開發人員指南

適用於 .NET 的 AWS SDK (V4)

Copyright © 2025 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

適用於 .NET 的 AWS SDK (V4): 開發人員指南

Copyright © 2025 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon 的商標和商業外觀不得用於任何非 Amazon 的產品或服務，也不能以任何可能造成客戶混淆、任何貶低或使 Amazon 名譽受損的方式使用 Amazon 的商標和商業外觀。所有其他非 Amazon 擁有的商標均為其各自擁有者的財產，這些擁有者可能附屬於 Amazon，或與 Amazon 有合作關係，亦或受到 Amazon 贊助。

Table of Contents

..... ix
什麼是 適用於 .NET 的 AWS SDK 1
關於此版本 1
開發套件主要版本的維護與支援 1
常用案例 2
本區段的其他主題 2
相關 AWS 工具 2
適用於 Windows PowerShell 的工具和適用於 PowerShell Core 的工具 2
適用於 VS 程式碼的工具組 3
適用於 Visual Studio 的 Toolkit 3
Toolkit for Azure DevOps 3
支援平台 3
.NET Core 4
.NET 標準 2.0 4
.NET Framework 4.5 4
.NET Framework 3.5 4
可攜式類別程式庫和 Xamarin 4
Unity 支援 4
SDKs與工具參考 4
修訂歷史記錄 5
最新消息 5
其他資源 6
開始使用 9
安裝和設定您的工具鏈 9
跨平台開發 9
Windows 搭配 Visual Studio 和 .NET Core 10
下一步驟 10
使用 驗證 AWS 10
啟用和設定 IAM Identity Center 11
設定 SDK 以使用 IAM Identity Center。 11
啟動 AWS 存取入口網站工作階段 12
其他資訊 12
建立簡單的應用程式 13
簡單的跨平台應用程式 13

簡單的 Windows 型應用程式 19
後續步驟 24
設定 25
啟動新的專案 25
使用 NuGet 安裝 AWSSDK 套件 26
從命令提示字元或終端機使用 NuGet 27
從 Visual Studio Solution Explorer 使用 NuGet 28
使用套件管理員主控台的 NuGet 28
安裝不含 NuGet 的 AWSSDK 組件 28
AWS 區域 30
使用特定區域建立服務用戶端 30
為所有服務用戶端指定區域 31
區域解析 32
中國 （北京） 區域的特殊資訊 32
新 AWS 服務的特殊資訊 33
憑證和設定檔解析 33
設定檔解析 34
使用聯合使用者帳戶登入資料 34
指定角色或臨時登入資料 35
使用代理登入資料 35
重試和逾時 35
重試 36
逾時 37
可觀測性 39
其他資源 40
設定 TelemetryProvider 40
指標 42
遙測供應商 43
使用者和角色 44
使用者和許可集合 45
服務角色 45
進階組態 46
AWSSDK.Extensions.NETCore.Setup 和 IConfiguration 46
設定其他應用程式參數 50
的組態檔案參考 適用於 .NET 的 AWS SDK 58
使用舊版憑證 69

憑證的重要警告和指引 70
使用共用 AWS 登入資料檔案 71
使用 SDK 存放區 （僅限 Windows) 74
使用開發套件 78
非同步程式設計 78
分頁 79
在哪裡可以找到分頁程式？ 80
分頁程式提供什麼？ 80
同步與非同步分頁 80
範例 80
分頁程式的其他考量事項 84
支援 HTTP 2 85
其他考量 87
其他工具 87
AWS 部署工具 87
AWS 適用於 .NET 的訊息處理架構 87
與 .NET Acover 整合 88
進階身分驗證 89
單一登入 89
先決條件 90
設定 SSO 設定檔 90
產生和使用 SSO 權杖 91
其他資源 95
教學課程 95
教學課程：僅限 .NET 應用程式 95
教學課程： AWS CLI 和 .NET 應用程式 104
部署至 AWS 112
從 .NET CLI 部署 112
從 IDE 工具組部署 112
使用案例 113
ASP.NET Core 應用程式 113
.NET 主控台應用程式 114
Blazor WebAssembly 應用程式 114
AWS Lambda 專案 115
先決條件 115
可用的 Lambda 命令 116

部署步驟 116
呼叫 AWS 服務 118
引導式程式碼範例 118
AWS CloudFormation 119
Amazon Cognito 123
DynamoDB 131
Amazon EC2 159
IAM 218
Amazon S3 237
Amazon SNS 247
Amazon SQS 251
AWS Lambda 283
API 283
先決條件 283
其他資訊 283
主題 283
Lambda 註釋 284
高階程式庫和架構 285
訊息處理架構 286
AWS 與 .NET Acover 整合 304
AWS OpsWorks 305
API 306
先決條件 306
其他服務和組態 306
程式碼範例 307
Aurora 307
基本概念 309
動作 334
Auto Scaling 349
基本概念 309
動作 334
Amazon Bedrock 384
動作 334
Amazon Bedrock 執行時間 388
Amazon Titan 文字 389
Anthropic Claude 394

Cohere Command 399
Meta Llama 403
混合式 AI 407
AWS CloudFormation 410
CloudWatch 414
Amazon Cognito 身分提供者 415
動作 334
AWS Control Tower 417
基本概念 309
動作 334
DynamoDB 456
基本概念 309
動作 334
Amazon EC2 495
Amazon ECS 497
Amazon S3 499
動作 334
案例 501
遷移您的專案 510
遷移至第 4 版 510
.NET Framework 510
值類型 510
集合 511
AWS Security Token Service (STS) 511
與 相關的變更 ClientConfig 512
AWSSDK.Extensions.NETCore.Setup NuGet 套件 512
CookieSigner 和 UrlSigner 513
DateTime 與 UTC DateTime 513
DateTime 剖析 514
ConvertFromUnixEpochSeconds 和 ConvertFromUnixEpochMilliseconds 514
日誌 514
支援 HTTP 2 515
單一登入 515
DynamoDB 特定的變更 515
EC2 特定的變更 519
S3 特定的變更 519

已移除的程式設計元素 ... 522
從 .NET Standard 1.3 遷移 ... 525
安全 ... 526
資料保護 ... 526
身分和存取權管理 ... 527
目標對象 ... 528
使用身分驗證 ... 528
使用政策管理存取權 ... 531
AWS 服務 如何使用 IAM ... 533
對 AWS 身分和存取進行故障診斷 ... 533
合規驗證 ... 534
恢復能力 ... 535
基礎設施安全性 ... 536
強制執行最低 TLS 版本 ... 536
.NET Core ... 536
.NET Framework ... 537
AWS Tools for PowerShell ... 538
Xamarin ... 539
Unity ... 539
瀏覽器 (適用於 Blazor WebAssembly) ... 540
S3 加密用戶端遷移 ... 540
遷移概觀 ... 540
將現有用戶端更新為 V1-transitional用戶端以讀取新格式 ... 541
將 V1-transitional用戶端遷移至 V2 用戶端以寫入新格式 ... 541
將 V2 用戶端更新為不再讀取 V1 格式 ... 545
特殊考量 ... 546
取得 AWSSDK 組件 ... 546
下載並擷取 ZIP 檔案 ... 546
在應用程式中存取登入資料和設定檔 ... 547
類別 CredentialProfileStoreChain 的範例 ... 547
SharedCredentialsFile 和 AWSCredentialsFactory 類別的範例 ... 549
Unity 支援 ... 549
Xamarin 支援 ... 550
API 參考 ... 551
關於 API 參考版本 ... 551
文件歷史紀錄 ... 553

第 4 版 (V4) 適用於 .NET 的 AWS SDK 已發行！

如需有關中斷變更和遷移應用程式的資訊，請參閱遷移主題。

本文為英文版的機器翻譯版本，如內容有任何歧義或不一致之處，概以英文版為準。

什麼是 適用於 .NET 的 AWS SDK

適用於 .NET 的 AWS SDK 可讓您更輕鬆地建置 .NET 應用程式，以利用經濟實惠、可擴展且可靠的 AWS 服務，例如 Amazon Simple Storage Service (Amazon S3) 和 Amazon Elastic Compute Cloud (Amazon EC2)。開發套件提供一組對 .NET 開發人員一致且熟悉的 AWS 程式庫，可簡化 服務的使用。

(好的，我知道了！ 我已準備好設定並快速導覽。)

關於此版本

目前和過去的文件

本文件適用於 的 4.0 版和更新版本 (V4) 適用於 .NET 的 AWS SDK。它主要以 .NET Core 和 ASP.NET Core 為中心，但也包含 .NET Framework 和 ASP.NET 4.x 的相關資訊。除了 Windows 和 Visual Studio 之外，它還提供了跨平台開發的同等考量。
如需從舊版 SDK 遷移的資訊，請參閱 遷移您的專案。
如需將在未來棄用之舊版 SDK V3 的相關資訊，請參閱 適用於 .NET 的 AWS SDK (V3) 開發人員指南。
若要尋找舊版 SDK 的已棄用內容，請參閱下列項目 （項目）：

- 適用於 .NET 的 AWS SDK （第 2 版，已棄用） 開發人員指南
- 已棄用 的 API 參考 適用於 .NET 的 AWS SDK

開發套件主要版本的維護與支援

如需開發套件主要版本及其基礎相依性之維護與支援的相關資訊，請參閱《AWS 開發套件及工具參考指南》中的以下內容：

- AWS SDKs和工具維護政策
- AWS SDKs和工具版本支援矩陣

常用案例

適用於 .NET 的 AWS SDK 可協助您實現數個令人信服的使用案例，包括下列項目：

- 使用 AWS Identity and Access Management (IAM) 管理使用者和角色。
- 存取 Amazon Simple Storage Service (Amazon S3) 以建立儲存貯體和存放物件。
- 管理主題的 Amazon Simple Notification Service (Amazon SNS) HTTP 訂閱。
- 使用 S3 傳輸公用程式，將檔案從 Xamarin 應用程式傳輸到 Amazon S3。
- 使用 Amazon Simple Queue Service (Amazon SQS) 來處理系統中元件之間的訊息和工作流程。
- 透過將 SQL 陳述式傳送至 Amazon S3 Select，執行高效的 Amazon S3 傳輸。
- 建立和啟動 Amazon EC2 執行個體，以及設定和請求 Amazon EC2 Spot 執行個體。

本區段的其他主題

- AWS 與 相關的工具 適用於 .NET 的 AWS SDK
- 支援的平台 適用於 .NET 的 AWS SDK
- AWS SDKs和工具參考指南
- 修訂歷史記錄
- 中的新功能 適用於 .NET 的 AWS SDK
- 其他資源

AWS 與 相關的工具 適用於 .NET 的 AWS SDK

適用於 Windows PowerShell 的工具和適用於 PowerShell Core 的工具

AWS Tools for Windows PowerShell 和 AWS Tools for PowerShell Core 是建置在 公開功能上的 PowerShell 模組 適用於 .NET 的 AWS SDK。 AWS PowerShell 工具可讓您從 PowerShell 提示字元編寫 AWS 資源操作的指令碼。雖然 cmdlet 會使用軟體開發套件的服務用戶端與方法來進行實作，但使用者可以享有 cmdlet 所提供的 PowerShell 體驗，進而採用慣常的方式來指定參數與處理結果。

若要開始使用，請參閱AWS Tools for Windows PowerShell。

適用於 VS 程式碼的工具組

AWS Toolkit for Visual Studio Code 是適用於 Visual Studio 程式碼 (VS 程式碼) 編輯器的外掛程式。此工具組可讓您更輕鬆地開發、除錯和部署使用 AWS的應用程式。

透過此工具組，您可以執行下列操作：

- 建立包含 AWS Lambda 函數的無伺服器應用程式，然後將應用程式部署到 AWS CloudFormation 堆疊。
- 使用 Amazon EventBridge 結構描述。
- 使用 Amazon ECS 任務定義檔案時，請使用 IntelliSense。
- 視覺化 AWS Cloud Development Kit (AWS CDK) 應用程式。

適用於 Visual Studio 的 Toolkit

AWS Toolkit for Visual Studio 是 Visual Studio IDE 的外掛程式，可讓您更輕鬆地開發、偵錯和部署使用 Amazon Web Services 的 .NET 應用程式。Toolkit for Visual Studio 為 Lambda 和 Web 應用程式和無伺服器應用程式的部署精靈等服務提供 Visual Studio 範本。您可以使用 AWS Explorer 來管理 Amazon EC2 執行個體、使用 Amazon DynamoDB 資料表、將訊息發佈至 Amazon Simple Notification Service (Amazon SNS) 佇列等，全部都在 Visual Studio 中。

若要開始使用，請參閱設定 AWS Toolkit for Visual Studio。

Toolkit for Azure DevOps

新增 AWS Toolkit for Microsoft Azure DevOps 任務，以輕鬆啟用 Azure DevOps 和 Azure DevOps Server 中的建置和發行管道，以使用 AWS 服務。您可以使用 Amazon S3、 AWS Elastic Beanstalk AWS CodeDeploy、Lambda、 AWS CloudFormation、Amazon Simple Queue Service (Amazon SQS) 和 Amazon SNS。您也可以使用 Windows PowerShell 模組和 AWS Command Line Interface () 執行命令AWS CLI。

若要開始使用 AWS Toolkit for Azure DevOps，請參閱 AWS Toolkit for Microsoft Azure DevOps 使用者指南。

支援的平台 適用於 .NET 的 AWS SDK

為開發人員 適用於 .NET 的 AWS SDK 提供不同的組件群組，以鎖定不同的平台。不過，在這些平台上，並非所有開發套件的功能都相同。此主題說明每個平台支援的差異。

.NET Core

適用於 .NET 的 AWS SDK 支援針對 .NET Core (.NET Core 3.1、.NET 5、.NET 6 等） 撰寫的應用程式。 AWS 服務用戶端僅支援 .NET Core 中的非同步呼叫模式。這也會影響許多建置在服務用戶端上的高階抽象，例如 Amazon S3`TransferUtility`，這只會支援 .NET Core 環境中的非同步呼叫。

.NET 標準 2.0

的非影格變化 適用於 .NET 的 AWS SDK 符合 .NET Standard 2.0。僅針對針對 .NET Standard 寫入的應用程式 適用於 .NET 的 AWS SDK 提供非同步方法。

.NET Framework 4.5

此版本的 適用於 .NET 的 AWS SDK 是根據 .NET Framework 4.7.2 編譯，並在 .NET 4.0 執行期. AWS service 用戶端中執行，支援同步和非同步呼叫模式，並使用 非同步並等待 C# 5.0 中引入的關鍵字。

.NET Framework 3.5

第 4 版 適用於 .NET 的 AWS SDK 不支援 .NET Framework 3.5。

可攜式類別程式庫和 Xamarin

適用於 .NET 的 AWS SDK 也包含可攜式類別程式庫實作。可攜式類別程式庫實作可以鎖定多個平台，包括 iOS 和 Android 上的 Universal Windows Platform (UWP) 和 Xamarin。如需詳細資訊，請參閱適用於 .NET 和 Xamarin 的 Mobile SDK。 AWS 服務用戶端僅支援非同步呼叫模式。

Unity 支援

如需 Unity 支援的相關資訊，請參閱 Unity 支援的特殊考量。

AWS SDKs和工具參考指南

AWS SDKs和工具參考指南包含許多 AWS SDKs 和工具組以及 相關且重要的資訊 AWS CLI。以下是參考包含的一些資訊範例：

- 共用 AWSconfig和 `credentials` 檔案及其位置的相關資訊。
- 設定 AWS 帳戶、使用者和角色

- 組態和身分驗證設定參考
- AWS 通用執行期 (CRT) 程式庫
- AWS SDKs和工具維護政策
- AWS SDKs和工具版本支援矩陣

修訂歷史記錄

若要了解各種版本中的變更，請參閱以下內容：

- SDK 變更日誌
- 中的新功能 適用於 .NET 的 AWS SDK
- 文件歷史紀錄

中的新功能 適用於 .NET 的 AWS SDK

如需 相關新開發的高階資訊 適用於 .NET 的 AWS SDK，請參閱位於 https：//https://aws.amazon.com/sdk-for-net/ 的產品頁面和 SDK 變更日誌。

以下是 中的新功能 適用於 .NET 的 AWS SDK。

2025 年 8 月 21 日： 第 3 版即將end-of-support 適用於 .NET 的 AWS SDK

適用於 .NET 的 AWS SDK 已宣布終止對 第 end-of-support。請參閱遷移指南以遷移 V3 應用程式並避免中斷。如需詳細資訊，請參閱部落格文章宣布 適用於 .NET 的 AWS SDK v3 end-of-support。

2025 年 7 月 2 日：分散式快取提供者的一般可用性

適用於 Amazon DynamoDB 的 AWS .NET 分散式快取提供者已全面推出！此提供者實作 ASP.NET Core IDistributedCache 介面，可讓您將 DynamoDB 的全受管且耐用的基礎設施整合到您的快取層中，並將程式碼變更降至最低。分散式快取提供者可透過 NuGet 上的 AWS.AspNetCore.DistributedCacheProvider 套件取得。如需詳細資訊，請參閱部落格文章 .AWS NET Distributed Cache Provider for Amazon DynamoDB 現在已全面推出。

> **Note**
>
> 只有 的 V4 才支援分散式快取提供者的發行版本 適用於 .NET 的 AWS SDK。如果您嘗試了分散式快取提供者的預覽版本，您可能需要更新應用程式中的套件相依性。

2025 年 5 月 5 日：適用於 .NET AWS 的訊息處理架構的一般可用性

AWS 適用於 .NET 的訊息處理架構已全面推出！框架是 AWS原生架構，可簡化開發 .NET 訊息處理應用程式，這些應用程式使用 Amazon Simple Queue Service (SQS)、Amazon Simple Notification Service (SNS) 和 Amazon EventBridge 等 AWS 服務。

2025 年 4 月 28 日：第 4 版 適用於 .NET 的 AWS SDK

第 4 版 適用於 .NET 的 AWS SDK 已全面推出！如需將應用程式遷移至 V4 的資訊，請參閱 遷移至第 4 版。另請參閱發佈一般可用性的 V 適用於 .NET 的 AWS SDK V40 一般可用性部落格文章。

2025 年 2 月 15 日：與 .NET Aspire 整合

已發佈與 .NET ACache 的整合，以改善內部開發迴圈。如需相關資訊，請參閱在 中 AWS 與 .NET ACache 整合 適用於 .NET 的 AWS SDK。

2025 年 2 月 10 日：GA 發行以獲得可觀測性

可觀測性是從系統發出的資料推斷系統目前狀態的程度。可觀測性已新增至 適用於 .NET 的 AWS SDK，包括遙測供應商的實作。如需詳細資訊，請參閱本指南可觀測性中的 和部落格文章宣布 AWS .NET OpenTelemetry 程式庫的一般可用性。

2025 年 1 月 15 日：新的完整性保護預設行為

從 3.7.412.0 版開始 適用於 .NET 的 AWS SDK，軟體開發套件會自動計算上傳的CRC32檢查總和，以提供預設完整性保護。如需詳細資訊，請參閱 GitHub 上的公告，網址為 https://github.com/aws/aws-sdk-net/issues/3610。軟體開發套件也提供全域設定，用於外部設定的資料完整性保護，您可以在軟體AWS SDKs和工具參考指南中的資料完整性保護中閱讀這些設定。

2024 年 11 月 15 日：第 4 版預覽版的部落格文章

第 4 適用於 .NET 的 AWS SDK 版的預覽版 4 已於 2024 年 11 月 15 日發行。如需此預覽的詳細資訊，請參閱部落格文章 Preview 4 of 適用於 .NET 的 AWS SDK V4。

2024 年 8 月 16 日：第 4 版預覽版的部落格文章

第 4 適用於 .NET 的 AWS SDK 版已於 2024 年 8 月 16 日作為第一次預覽發行。如需此預覽的詳細資訊，請參閱部落格文章預覽 1 of 適用於 .NET 的 AWS SDK V4。

其他資源

支援的服務

適用於 .NET 的 AWS SDK 支援大多數 AWS 基礎設施產品，並經常新增更多服務。如需 SDK 支援 AWS 的服務清單，請參閱 SDK README 檔案。

的首頁 適用於 .NET 的 AWS SDK

如需 的詳細資訊 適用於 .NET 的 AWS SDK，請參閱 SDK 的首頁，網址為 https：//https://aws.amazon.com/sdk-for-net/。

SDK 參考文件

軟體開發套件參考文件可讓您瀏覽和搜尋軟體開發套件隨附的所有程式碼。它提供完整的文件和使用範例。如需詳細資訊，請參閱 適用於 .NET 的 AWS SDK API 參考。

AWS re：Post （先前 AWS 為論壇）

請造訪 AWS re：Post，特別是 的主題 適用於 .NET 的 AWS SDK，以提出問題或提供意見回饋 AWS。每個文件頁面底部都有 Try AWS re：Post 連結，可帶您前往相關聯的 re：Post 主題。 AWS 工程師會監控主題並回應問題、意見回饋和問題。

如果您已登入 re：Post，您也可以遵循主題。若要遵循 的主題 適用於 .NET 的 AWS SDK，請前往所有主題頁面，尋找開啟的「.NET AWS」，然後選取遵循按鈕。

工具組

- AWS Toolkit for Visual Studio：如果您使用 Microsoft Visual Studio IDE，您應該參閱 AWS Toolkit for Visual Studio 使用者指南。
- AWS Toolkit for Visual Studio Code：如果您使用 Microsoft Visual Studio IDE，您應該查看 AWS Toolkit for Visual Studio Code 使用者指南。

實用的程式庫、擴充功能和工具

請造訪 GitHub 網站上的 aws/dotnet 和 aws/aws-sdk-net 儲存庫，以取得可用於在 上建置 .NET 應用程式和服務之程式庫、工具和資源的連結 AWS。

下列是一些範例：

- AWS Systems Manager 的 .NET 組態延伸模組
- AWS 延伸模組 .NET Core 設定
- AWS 記錄 .NET
- Amazon Cognito Authentication Extension Library

- 適用於 .NET 的 AWS X-Ray SDK

其他資源

以下是其他可能很有用的資源：

- 開發人員網路
- AWS 雲端上的 .NET 開發環境 - 快速入門參考部署
- 您好，Cloud！ 部落格
- AWS 白皮書：在 上開發和部署 .NET 應用程式 AWS
- AWS Microservice Extractor for .NET
- 適用於 .NET 的移植助理
- AWS SDKs和工具參考指南

開始使用 適用於 .NET 的 AWS SDK

了解如何安裝、設定和使用 適用於 .NET 的 AWS SDK。您必須安裝工具鏈，並設定應用程式存取 AWS 服務所需的許多基本項目。其中包含：

- 適當的使用者帳戶或角色
- 該使用者帳戶或 擔任該角色的身分驗證資訊

如需如何設定專案和其他關鍵組件的資訊，請參閱 設定 適用於 .NET 的 AWS SDK。

主題

- 安裝和設定 的工具鏈 適用於 .NET 的 AWS SDK
- 適用於 .NET 的 AWS SDK 使用 驗證 AWS
- 使用 建立簡單的應用程式 適用於 .NET 的 AWS SDK

安裝和設定 的工具鏈 適用於 .NET 的 AWS SDK

若要使用 適用於 .NET 的 AWS SDK，您必須安裝特定開發工具。

跨平台開發

在 Windows、Linux 或 macOS 上進行跨平台 .NET 開發時需要下列項目：

- Microsoft .NET Core 軟體開發套件 2.1 版、3.1 版或更新版本，包括 .NET 命令列界面 (CLI) (**`dotnet`**) 以及 .NET Core 執行時間。

- 適合您作業系統和需求的程式碼編輯器或整合式開發環境 (IDE)。這通常是為 .NET Core 提供一些支援。

  舉例來說，包括 Microsoft Visual Studio 程式碼 (VS 程式碼)、JetBrains Rider 和 Microsoft Visual Studio。

- （選用）如果 AWS 工具組可用於您選擇的編輯器和作業系統。

  範例包括 AWS Toolkit for Visual Studio Code、 AWS Toolkit for JetBrains和 AWS Toolkit for Visual Studio。

Windows 搭配 Visual Studio 和 .NET Core

在 Windows 上使用 Visual Studio 和 .NET Core 進行開發時需要下列項目：

- Microsoft Visual Studio

- Microsoft .NET Core 2.1、3.1 或更新版本

  根據預設，安裝最新版本的 Visual Studio 時通常會包含此項目。

- （選用）是一種 AWS Toolkit for Visual Studio外掛程式，提供用於從 Visual Studio 管理 AWS 資源和本機設定檔的使用者介面。若要安裝工具組，請參閱設定 AWS Toolkit for Visual Studio。

  如需詳細資訊，請參閱「AWS Toolkit for Visual Studio 使用者指南」。

下一步驟

適用於 .NET 的 AWS SDK 使用 驗證 AWS

適用於 .NET 的 AWS SDK 使用 驗證 AWS

在使用 進行開發 AWS 時，您必須建立程式碼向 進行身分驗證的方式 AWS 服務。視環境和您可用的存取權而定，您可以透過不同方式設定 AWS 資源的程式設計 AWS 存取。

若要查看 SDK 的各種身分驗證方法，請參閱 AWS SDKs和工具參考指南中的身分驗證和存取。

本主題假設新的使用者正在本機開發，其雇主尚未提供身分驗證方法，且將使用 AWS IAM Identity Center 取得臨時登入資料。如果您的環境不適用這些假設，則本主題中的有些資訊可能不適用您的環境，或者有些資訊可能已經提供給您。

設定此環境需要幾個步驟，總結如下：

1. 啟用和設定 IAM Identity Center
2. 設定 SDK 以使用 IAM Identity Center。
3. 啟動 AWS 存取入口網站工作階段

啟用和設定 IAM Identity Center

若要使用 IAM Identity Center，必須先啟用和設定它。若要查看如何為 SDK 執行此操作的詳細資訊，請參閱 SDK 和工具參考指南中的 IAM Identity Center 身分驗證主題中的步驟 1。 https://docs.aws.amazon.com/sdkref/latest/guide/access-sso.html AWS SDKs 具體來說，請遵循我沒有透過 IAM Identity Center 建立存取權限下的任何必要說明。

設定 SDK 以使用 IAM Identity Center。

有關如何設定 SDK 以使用 IAM Identity Center 的資訊，請參閱 SDK AWS SDKs 和工具參考指南中 IAM Identity Center 身分驗證主題的步驟 2。完成此組態之後，您的系統應該包含下列元素：

- 在執行應用程式之前 AWS CLI，您用來啟動 AWS 存取入口網站工作階段的 。
- 包含[default]設定檔的共用 AWS config檔案，其中包含一組可從 SDK 參考的組態值。若要尋找此檔案的位置，請參閱 AWS SDK 和工具參考指南中的共用檔案位置。 適用於 .NET 的 AWS SDK 使用設定檔的 SSO 權杖提供者，在傳送請求至 之前取得登入資料 AWS。該 sso_role_name 值是連接到 IAM Identity Center 許可集合的 IAM 角色，應該允許存取應用程式中使用的 AWS 服務 。

  下列範例 config 檔案顯示使用 SSO 權杖提供者設定的預設設定檔。設定檔的 sso_session 設定是指已命名的 sso-session 區段。sso-session 區段包含啟動 AWS 存取入口網站工作階段的設定。

```
[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 進行身分驗證，您的應用程式必須參考下列 NuGet 套件，以便 SSO 解析能夠運作：

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

未參考這些套件將導致執行時間例外狀況。

啟動 AWS 存取入口網站工作階段

在執行存取的應用程式之前 AWS 服務，您需要 SDK 的作用中 AWS 存取入口網站工作階段，才能使用 IAM Identity Center 身分驗證來解析登入資料。視您設定的工作階段長度而定，您的存取最終會過期，而且 SDK 會遇到身分驗證錯誤。若要登入 AWS 存取入口網站，請在 中執行下列命令 AWS CLI。

```
aws sso login
```

由於您有預設的設定檔設定，因此您不需要使用 `--profile` 選項呼叫指令。如果您的 SSO 權杖提供者組態使用已命名的設定檔，則命令為 `aws sso login --profile` *`named-profile`*。

若要測試您是否已經有作用中的工作階段，請執行下列 AWS CLI 命令。

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

對此命令的回應，應報告共用 `config` 檔案中設定的 IAM Identity Center 帳戶和許可集合。

> **Note**
>
> 如果您已有作用中的 AWS 存取入口網站工作階段並執行 `aws sso login`，則不需要提供登入資料。
> 登入程序可能會提示您允許 AWS CLI 存取您的資料。由於 AWS CLI 建置在適用於 Python 的 SDK 之上，因此許可訊息可能包含`botocore`名稱的變化。

其他資訊

- 如需在開發環境中使用 IAM Identity Center 和 SSO 的詳細資訊，請參閱 進階身分驗證 一節單一登入中的 。此資訊包括替代和更進階的方法，以及向您展示如何使用這些方法的教學課程。

- 如需 SDK 身分驗證的更多選項，例如使用設定檔和環境變數，請參閱 AWS SDKs和工具參考指南中的組態章節。
- 如需了解有關最佳實務的資訊，請參閱 IAM 使用者指南中的 IAM 安全最佳實務。
- 若要建立短期 AWS 登入資料，請參閱《IAM 使用者指南》中的暫時安全登入資料。
- 若要瞭解其他憑證提供者，請參閱 AWS SDK 和工具參考指南中的標準化憑證提供者。

使用 建立簡單的應用程式 適用於 .NET 的 AWS SDK

本節為初次使用 的開發人員提供基本教學課程 適用於 .NET 的 AWS SDK。

> **Note**
>
> 使用這些教學課程之前，您必須先安裝工具鏈並設定 SDK 身分驗證。

如需為特定 AWS 服務開發軟體以及程式碼範例的詳細資訊，請參閱 呼叫 AWS 服務。如需其他程式碼範例，請參閱 適用於 .NET 的 SDK (v4) 程式碼範例。

主題

- 使用 的簡單跨平台應用程式 適用於 .NET 的 AWS SDK
- 使用 的簡易 Windows 應用程式 適用於 .NET 的 AWS SDK
- 後續步驟

使用 的簡單跨平台應用程式 適用於 .NET 的 AWS SDK

本教學課程使用 適用於 .NET 的 AWS SDK 和 .NET Core 進行跨平台開發。本教學課程說明如何使用 SDK 列出您擁有的 Amazon S3 儲存貯體，並選擇性地建立儲存貯體。

您將使用 .NET 命令列界面 (CLI) 等跨平台工具來執行此教學課程。如需設定開發環境的其他方式，請參閱 安裝和設定 的工具鏈 適用於 .NET 的 AWS SDK。

在 Windows、Linux 或 macOS 上進行跨平台 .NET 開發時需要：

- Microsoft .NET Core 軟體開發套件 2.1 版、3.1 版或更新版本，包括 .NET 命令列界面 (CLI) (**`dotnet`**) 以及 .NET Core 執行時間。

- 程式碼編輯器或整合式開發環境 (IDE)，適用於您的作業系統和需求。這通常是為 .NET Core 提供一些支援。

  舉例來說，包括 Microsoft Visual Studio 程式碼 (VS 程式碼)、JetBrains Rider 和 Microsoft Visual Studio。

> **Note**
>
> 使用這些教學課程之前，您必須先安裝工具鏈並設定 SDK 身分驗證。

步驟

- 建立專案
- 建立程式碼
- 執行應用程式
- 清除

建立專案

1. 開啟命令提示字元或終端機。尋找或建立可在其下建立 .NET 專案的作業系統資料夾。
2. 在該資料夾中，執行下列命令以建立 .NET 專案。

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

3. 移至新建立的`S3CreateAndList`資料夾並執行下列命令：

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

   上述命令會從 NuGet 套件管理員安裝 NuGet 套件。由於我們確切知道本教學課程所需的 NuGet 套件，因此現在可以執行此步驟。在開發期間，所需的套件也很常見。發生這種情況時，即可執行類似的命令。

建立程式碼

1. 在 S3CreateAndList 資料夾中，在程式碼編輯器中尋找並開啟 Program.cs。
2. 以下列程式碼取代內容，並儲存檔案。

```
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;
        }
    }
}
```

執行應用程式

1. 執行下列命令。

```
dotnet run
```

2. 檢查輸出以查看您擁有的 Amazon S3 儲存貯體數量，如果有的話，及其名稱。
3. 選擇新 Amazon S3 儲存貯體的名稱。使用「dotnet-quicktour-s3-1-cross-」作為基礎，並為其新增唯一項目，例如 GUID 或您的名稱。請務必遵循儲存貯體名稱的規則，如 Amazon S3 使用者指南中的儲存貯體命名規則所述。
4. 執行下列命令，以您選擇的儲存貯體名稱取代 *amzn-s3-demo-bucket*。

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

5. 檢查輸出以查看建立的新儲存貯體。

清除

執行本教學課程時，您建立了一些資源，此時可以選擇清除。

- 如果您不想保留應用程式在先前步驟中建立的儲存貯體，請使用 Amazon S3 主控台將其刪除，網址為 https：//https://console.aws.amazon.com/s3/。
- 如果您不想保留 .NET 專案，請從開發環境中刪除該 `S3CreateAndList` 資料夾。

後續作業

返回快速導覽功能表或直接前往此快速導覽的結尾。

使用 的簡易 Windows 應用程式 適用於 .NET 的 AWS SDK

本教學課程使用 Windows 適用於 .NET 的 AWS SDK 上的 搭配 Visual Studio 和 .NET Core。本教學課程說明如何使用 SDK 列出您擁有的 Amazon S3 儲存貯體，並選擇性地建立儲存貯體。

您將使用 Visual Studio 和 .NET Core 在 Windows 上執行此教學課程。如需設定開發環境的其他方式，請參閱 安裝和設定 的工具鏈 適用於 .NET 的 AWS SDK。

在 Windows 上使用 Visual Studio 和 .NET Core 進行開發時需要：

- Microsoft Visual Studio
- Microsoft .NET Core 2.1、3.1 或更新版本

  在安裝最新版本的 Visual Studio 時，這通常是預設包含的。

> **Note**
>
> 使用這些教學課程之前，您必須先安裝工具鏈並設定 SDK 身分驗證。

步驟

- 建立專案
- 建立程式碼
- 執行應用程式
- 清除

建立專案

1. 開啟 Visual Studio 並建立新的專案，使用主控台應用程式範本的 C# 版本；亦即，描述為「...用於建立可在 .NET... 上執行的命令列應用程式」。命名專案 `S3CreateAndList`。

   > **Note**
   >
   > 請勿選擇主控台應用程式範本的 .NET Framework 版本，或者，如果您選擇，請務必使用 .NET Framework 4.7.2 或更新版本。

2. 載入新建立的專案後，選擇工具、NuGet Package Manager、管理 NuGet Packages for Solution。
3. 瀏覽下列 NuGet 套件，並將其安裝到專案中：AWSSDK.S3、AWSSDK.SSO、AWSSDK.SecurityToken和 AWSSDK.SSOOIDC

   此程序會從 NuGet 套件管理員安裝 NuGet 套件。由於我們確切知道本教學課程所需的 NuGet 套件，因此現在可以執行此步驟。在開發期間，所需的套件也很常見。發生這種情況時，請按照類似的過程進行安裝。
4. 如果您想要從命令提示字元執行應用程式，請現在開啟命令提示字元，然後導覽至將包含建置輸出的資料夾。這通常類似 S3CreateAndList\S3CreateAndList\bin\Debug\net6.0，但取決於您的環境。

建立程式碼

1. 在 S3CreateAndList 專案中，尋找並在 IDE 中開啟 Program.cs。
2. 以下列程式碼取代內容，並儲存檔案。

```
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. 建置應用程式。

> **Note**
>
> 如果您使用的是較舊版本的 Visual Studio，您可能會收到類似以下的建置錯誤：
> C# 7.0 不提供「功能」非同步主節點」。請使用語言版本 7.1 或更新版本。」
> 如果您收到此錯誤，請將您的專案設定為使用較新版本的語言。這通常在專案屬性、建置、進階中完成。

執行應用程式

1. 執行沒有命令列引數的應用程式。在命令提示字元（如果您先前已開啟）或從 IDE 中執行此操作。
2. 檢查輸出以查看您擁有的 Amazon S3 儲存貯體數量，如果有的話，及其名稱。
3. 選擇新 Amazon S3 儲存貯體的名稱。使用「dotnet-quicktour-s3-1-winvs-」作為基礎，並為其新增唯一項目，例如 GUID 或您的名稱。請務必遵循 Amazon S3 使用者指南中儲存貯體命名規則中所述的儲存貯體名稱規則。

4. 再次執行應用程式，這次提供存儲存貯體名稱。

   在命令列中，使用您選擇的儲存貯體名稱取代下列命令中的 *amzn-s3-demo-bucket*。

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

   或者，如果您在 IDE 中執行應用程式，請選擇專案、S3CreateAndList 屬性、偵錯，然後在該處輸入儲存貯體名稱。
5. 檢查輸出以查看建立的新儲存貯體。

清除

執行本教學課程時，您建立了一些資源，此時可以選擇清除。

- 如果您不想保留應用程式在先前步驟中建立的儲存貯體，請使用 Amazon S3 主控台將其刪除，網址為 https：//https://console.aws.amazon.com/s3/。
- 如果您不想保留 .NET 專案，請從開發環境中刪除該 S3CreateAndList 資料夾。

後續作業

返回快速導覽功能表或直接前往此快速導覽的結尾。

後續步驟

請務必清除您執行這些教學課程時所建立的剩餘資源。這些可能是開發環境中 AWS 的資源，例如檔案和資料夾。

現在您已瀏覽 適用於 .NET 的 AWS SDK，您可能想要啟動專案。

設定 適用於 .NET 的 AWS SDK

若要使用 適用於 .NET 的 AWS SDK，您需要設定您的環境，並設定應用程式存取 AWS 服務所需的許多基本項目。其中包含：

- AWS 區域的規格
- AWSSDK 套件或組件

本節中的某些主題提供如何設定這些基本物件的資訊。

本節和其他章節中的其他主題提供有關您可設定專案之更進階方式的資訊。

主題

- 啟動新的專案
- 使用 NuGet 安裝 AWSSDK 套件
- 安裝不含 NuGet 的 AWSSDK 組件
- 設定 AWS 的區域 適用於 .NET 的 AWS SDK
- 憑證和設定檔解析
- 重試和逾時
- 可觀測性
- 有關使用者和角色的其他資訊
- 適用於 .NET 的 AWS SDK 專案的進階組態
- 使用舊版憑證

啟動新的專案

您可以使用數種技術來啟動新專案以存取 AWS 服務。以下是其中一些技術：

- 如果您是初次使用 上的 .NET 開發， AWS 或至少初次使用 適用於 .NET 的 AWS SDK，則可以在中查看完整的範例建立簡單的應用程式。它為您提供 SDK 的簡介。

- 您可以使用 .NET CLI 啟動基本專案。若要查看此範例，請開啟命令提示字元或終端機、建立資料夾或目錄並導覽至其中，然後輸入以下內容。

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

建立空專案，您可以將程式碼和 NuGet 套件新增至其中。如需詳細資訊，請參閱 .NET Core 指南。

若要查看專案範本清單，請使用下列各項： `dotnet new --list`

- AWS Toolkit for Visual Studio 包含各種 AWS 服務的 C# 專案範本。在 Visual Studio 中安裝工具組後，您可以在建立新專案時存取範本。

  若要查看，請前往AWS Toolkit for Visual Studio 《 使用者指南》中的使用 AWS 服務。該區段中的數個範例會建立新專案。

- 如果您在 Windows 上使用 Visual Studio 開發 ，但沒有 AWS Toolkit for Visual Studio，請使用您的典型技術來建立新的專案。

  若要查看範例，請開啟 Visual Studio，然後選擇檔案、新增、專案。搜尋 ".net core"，然後選擇主控台應用程式 (.NET Core) 或 WPF 應用程式 (.NET Core) 範本的 C# 版本。建立空專案，您可以將程式碼和 NuGet 套件新增至其中。

您可以找到一些如何在 中使用 AWS 服務的範例引導式程式碼範例。

> **Important**
>
> 如果您使用 AWS IAM Identity Center 進行身分驗證，您的應用程式必須參考下列 NuGet 套件，以便 SSO 解析能夠運作：
>
> - `AWSSDK.SSO`
> - `AWSSDK.SSOOIDC`
>
> 未參考這些套件將導致執行時間例外狀況。

使用 NuGet 安裝 AWSSDK 套件

NuGet 是適用於 .NET 平台的套件管理系統。使用 NuGet，您可以將 AWSSDK 套件以及其他數個擴充功能安裝到您的專案。如需詳細資訊，請參閱 GitHub 網站上的 aws/dotnet 儲存庫。

NuGet 一律具有最新版本的 AWSSDK 套件，以及先前的版本。NuGet 知道套件之間的相依性，並自動安裝所有必要的套件。

> **Warning**
>
> NuGet 套件清單可能包含一個名為「AWSSDK」的套件（沒有附加識別符）。請勿安裝此 NuGet 套件；它是舊版的，不應用於新專案。

與 NuGet 一起安裝的套件會與您的專案一起存放，而不是存放在中央位置。這可讓您安裝給定應用程式的特定組件版本，無需為其他應用程式產生相容性問題。如需 NuGet 的詳細資訊，請參閱 NuGet 說明文件。

> **Note**
>
> 如果您無法或無法依專案下載並安裝 NuGet 套件，您可以取得 AWSSDK 組件並將其存放在本機（或內部部署）。
> 如果這適用於您，而且您尚未取得 AWSSDK 組件，請參閱 取得 AWSSDK 組件。若要了解如何使用本機存放的組件，請參閱 安裝不含 NuGet 的 AWSSDK 組件。

從命令提示字元或終端機使用 NuGet

1. 前往 NuGet 上的 AWSSDK 套件，並判斷專案中需要哪些套件；例如 AWSSDK.S3。
2. 從該套件的網頁複製 .NET CLI 命令，如下列範例所示。

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

3. 在專案的 目錄中，執行該 .NET CLI 命令。NuGet 也可以安裝任何的相依性，如 AWSSDK.Core。

> **Note**
>
> 如果您只想要最新版本的 NuGet 套件，您可以從 命令中排除版本資訊，如下列範例所示。
> `dotnet add package AWSSDK.S3`

從 Visual Studio Solution Explorer 使用 NuGet

1. 在方案總管，按右鍵選取您的專案，從內容功能表中選擇管理 NuGet 套件。
2. 在 NuGet Package Manager (NuGet 套件管理員) 的左窗格中，選擇 Browse (瀏覽)。然後，您可以使用搜尋方塊來搜尋要安裝的套件。NuGet 也可以安裝任何的相依性，如 AWSSDK.Core。

   下圖顯示 AWSSDK.S3 套件的安裝。

使用套件管理員主控台的 NuGet

在 Visual Studio 中，選擇工具、NuGet 套件管理員、套件管理員主控台。

您可以使用 **`Install-Package`**命令，從套件管理員主控台安裝您想要的 AWSSDK 套件。例如，若要安裝 AWSSDK.S3，請使用下列命令。

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

NuGet 也可以安裝任何的相依性，如 AWSSDK.Core。

如果您需要安裝舊版的套件，請使用 `-Version`選項並指定您想要的套件版本，如下列範例所示。

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

如需套件管理員主控台命令的詳細資訊，請參閱 Microsoft NuGet 文件中的 PowerShell 參考。

安裝不含 NuGet 的 AWSSDK 組件

本主題說明如何使用您在本機（或內部部署）取得和存放的 AWSSDK 組件，如中所述取得 AWSSDK 組件。這不是處理 SDK 參考的建議方法，但在某些環境中是必要的。

> **Note**
>
> 處理 SDK 參考的建議方法是僅下載並安裝每個專案所需的 NuGet 套件。該方法如 中所述使用 NuGet 安裝 AWSSDK 套件。

安裝 AWSSDK 組件

1. 在專案區域中為所需的 AWSSDK 組件建立資料夾。例如，您可以呼叫此資料夾 `AwsAssemblies`。
2. 如果您尚未這麼做，請取得 AWSSDK 組件，將組件放在一些本機下載或安裝資料夾中。將所需組件的 DLL 檔案從該下載資料夾複製到您的專案（在我們的範例中為 `AwsAssemblies` 資料夾）。

   也請務必複製任何相依性。您可以在 GitHub 網站上找到相依性的相關資訊。
3. 請參考必要的組件，如下所示。

   Cross-platform development

   1. 開啟專案`.csproj`的檔案並新增 `<ItemGroup>`元素。
   2. 在 `<ItemGroup>`元素中，為每個必要的組件新增具有 `Include` 屬性的`<Reference>`元素。

      例如，對於 Amazon S3，您可以將以下幾行新增至專案的 `.csproj` 檔案。

      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. 儲存專案`.csproj`的檔案。

   Windows with Visual Studio and .NET Core

   1. 在 Visual Studio 中，載入專案並開啟專案、新增參考。

2. 選擇對話方塊底部的瀏覽按鈕。導覽至專案的資料夾，以及您複製所需 DLL 檔案的子資料夾（例如AwsAssemblies)。
3. 選取所有 DLL 檔案，選擇新增，然後選擇確定。
4. 儲存您的專案。

設定 AWS 的區域 適用於 .NET 的 AWS SDK

AWS 區域可讓您存取 AWS 實際位於特定地理區域的服務。這對於備援以及讓您的資料和應用程式，在靠近您和您的使用者存取位置附近執行，都很有用。

若要檢視每個 AWS 服務所有支援的區域和端點的目前清單，請參閱 中的服務端點和配額AWS 一般參考。若要檢視現有區域端點的清單，請參閱 AWS 服務端點。若要查看區域的詳細資訊，請參閱指定 AWS 您的帳戶可以使用的區域。

您可以建立前往特定區域的 AWS 服務用戶端。您也可以使用將用於所有 AWS 服務用戶端的區域來設定應用程式。接下來會說明這兩個案例。

使用特定區域建立服務用戶端

您可以為應用程式中的任何 AWS 服務用戶端指定 區域。以這種方式設定區域優先於該特定服務用戶端的任何全域設定。

現有區域

此範例說明如何在現有區域中執行個體化 Amazon EC2 用戶端。它使用定義的 RegionEndpoint 欄位。

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

使用 RegionEndpoint 類別的新區域

此範例說明如何使用 RegionEndpoint.GetBySystemName 建構新的區域端點。

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

```
}
```

使用服務用戶端組態類別的新區域

此範例說明如何使用服務用戶端組態類別的 ServiceURL 屬性來指定 區域；在此情況下，請使用 AmazonEC2Config 類別。

即使區域端點未遵循一般區域端點模式，此技術仍然有效。

```
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
}
```

為所有服務用戶端指定區域

您可以透過多種方式為您的應用程式建立的所有 AWS 服務用戶端指定區域。此區域用於未以特定區域建立的服務用戶端。

會依下列順序 適用於 .NET 的 AWS SDK 尋找區域值。

描述檔

在應用程式或 SDK 已載入的設定檔中設定 。如需詳細資訊，請參閱憑證和設定檔解析。

環境變數

在AWS_REGION環境變數中設定 。

在 Linux 或 macOS 上：

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

在 Windows 上：

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

> **Note**
>
> 如果您為整個系統（使用 `export`或 `setx`) 設定此環境變數，它會影響所有 SDKs 和工具組，而不只是 適用於 .NET 的 AWS SDK。

AWSConfigs 類別

設定為 AWSConfigs.AWSRegion 屬性。

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

區域解析

如果上述方法都未用於指定 AWS 區域，則 會 適用於 .NET 的 AWS SDK 嘗試尋找 AWS 服務用戶端要操作的區域。

區域解析順序

1. 應用程式組態檔案，例如 `app.config`和 `web.config`。
2. 環境變數 (`AWS_REGION` 和 `AWS_DEFAULT_REGION`)。
3. 描述檔，其名稱由 中的值指定`AWSConfigs.AWSProfileName`。
4. 具有 `AWS_PROFILE`環境變數指定名稱的設定檔。
5. `[default]` 設定檔。
6. Amazon EC2 執行個體中繼資料（如果在 EC2 執行個體上執行）。

如果找不到區域，軟體開發套件會擲回例外狀況，指出 AWS 服務用戶端沒有設定的區域。

中國（北京）區域的特殊資訊

若要使用中國 (北京) 地區中的服務，您必須有專屬於中國 (北京) 地區的帳戶和憑證。其他 AWS 區域的帳戶和登入資料不適用於中國（北京）區域。同樣地，中國（北京）區域的帳戶和登入資料不適用於其他 AWS 區域。如需中國（北京）區域中可用端點和通訊協定的相關資訊，請參閱北京區域端點。

新 AWS 服務的特殊資訊

新的 AWS 服務最初可以在幾個區域中啟動，然後在其他區域中支援。在這些情況下，您不需要安裝最新的 SDK 即可存取該服務的新區域。您可以依用戶端或全域指定新增的區域，如先前所示。

憑證和設定檔解析

會依特定順序 適用於 .NET 的 AWS SDK 搜尋登入資料，並使用目前應用程式的第一個可用集合。

登入資料搜尋順序

1. 在 AWS 服務用戶端上明確設定的登入資料，如中所述在應用程式中存取登入資料和設定檔。

   > **Note**
   >
   > 該主題位於 特殊考量區段中，因為它不是指定憑證的偏好方法。

2. 如果這三個變數都有值，則從 `AWS_ACCESS_KEY_ID`、 `AWS_SECRET_ACCESS_KEY`和 `AWS_SESSION_TOKEN`環境變數建立的 SessionAWSCredentials。
3. 如果兩個變數都有值，則從 `AWS_ACCESS_KEY_ID`和`AWS_SECRET_ACCESS_KEY`環境變數建立的 BasicAWSCredentials。
4. 如果兩個變數都有值，則從 `AWS_WEB_IDENTITY_TOKEN_FILE`和 `AWS_ROLE_ARN` 環境變數建立的 AssumeRoleWithWebIdentityCredentials。
5. 登入資料設定檔，其名稱由 AWSConfigs.AWSProfileName 中的值指定。
6. 登入資料設定檔，名稱為 `AWS_PROFILE`環境變數指定的名稱。
7. `[default]` 登入資料設定檔。
8. 容器憑證提供者。
9. Amazon EC2 執行個體中繼資料。

如果您的應用程式在 Amazon EC2 執行個體上執行，例如在生產環境中，請使用 中所述的 IAM 角色使用 IAM 角色授予存取權。否則，例如在發行前測試中，將您的登入資料存放在使用您 Web 應用程式可在伺服器上存取之 AWS 登入資料檔案格式的檔案中。

如需環境變數和登入資料設定檔的詳細資訊，請參閱 AWS SDKs和工具參考指南中的下列主題：環境變數、環境變數清單，以及共用組態和登入資料檔案。

設定檔解析

使用兩種不同的憑證儲存機制，了解如何設定 適用於 .NET 的 AWS SDK 來使用它們非常重要。AWSConfigs.AWSProfilesLocation 屬性會控制 如何 適用於 .NET 的 AWS SDK 尋找登入資料設定檔。

AWSProfilesLocation	設定檔解析行為
null (未設定) 或為空	如果平台支援，請搜尋 SDK Store，然後在預設位置搜尋共用 AWS 的登入資料檔案。如果設定檔不在這些位置，請搜尋 `~/.aws/config` (Linux 或 macOS) 或 `%USERPROFILE%\.aws\config` (Windows)。
AWS 登入資料檔案格式的檔案路徑	只 搜尋指定的檔案中是否有特定名稱的設定檔。

使用聯合使用者帳戶登入資料

使用 適用於 .NET 的 AWS SDK (AWSSDK.Core 3.1.6.0 版及更新版本） 的應用程式可以透過 Active Directory Federation Services (AD FS) 使用聯合使用者帳戶，透過安全聲明標記語言 (SAML) 存取 AWS 服務。

聯合身分存取權支援表示使用者可以使用 Active Directory 進行身分驗證。自動授與臨時登入資料給使用者。這些臨時登入資料在一小時內有效，會在您的應用程式叫用 AWS 服務時使用。開發套件會管理臨時登入資料。對於網域加入的使用者帳戶，如果您的應用程式呼叫過期，則會自動重新對使用者進行身分驗證和重新整理授與登入資料。（對於未加入網域的帳戶，系統會提示使用者在重新驗證之前輸入登入資料。）

若要在您的 .NET 應用程式中使用此支援，您必須先使用 PowerShell cmdlet 建立角色設定檔。若要了解如何操作，請參閱 AWS Tools for Windows PowerShell 文件。

設定角色描述檔之後，請在應用程式中參考描述檔。有多種方法可以執行此操作，其中一個是使用 AWSConfigs.AWSProfileName 屬性，就像使用其他登入資料描述檔一樣。

AWS Security Token Service 組件 (AWSSDK.SecurityToken) 提供 SAML 支援以取得 AWS 登入資料。若要使用聯合使用者帳戶登入資料，請確定您的應用程式可以使用此組件。

指定角色或臨時登入資料

對於在 Amazon EC2 執行個體上執行的應用程式，管理登入資料的最安全方法是使用 IAM 角色，如中所述使用 IAM 角色授予存取權。

對於您組織外部的使用者可使用軟體可執行檔的應用程式案例，我們建議您設計軟體以使用臨時安全登入資料。除了提供對 AWS 資源的限制存取之外，這些登入資料還可以在指定的一段時間後過期。如需臨時安全登入資料的詳細資訊，請參閱以下：

- 臨時安全登入資料
- Amazon Cognito 身分集區

使用代理登入資料

如果您的軟體 AWS 透過代理與 通訊，您可以使用 服務Config類別的 ProxyCredentials 屬性來指定代理的登入資料。服務的Config類別通常是服務主要命名空間的一部分。範例包括：AmazonCloudDirectoryConfig中的 AmazonCloudDirectory Config和 AmazonGameLiftConfig中的 AmazonGameLiftConfig。

例如，對於 Amazon S3，您可以使用類似以下的程式碼，其中 SecurelyStoredUserName和 SecurelyStoredPassword是 NetworkCredential 物件中指定的代理使用者名稱和密碼。

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

> **Note**
>
> 舊版的開發套件使用 ProxyUsername 和 ProxyPassword，但這些屬性都已停用。

重試和逾時

適用於 .NET 的 AWS SDK 可讓您設定對 AWS 服務的 HTTP 請求的重試次數和逾時值。如果重試和逾時的預設不適用於您的應用程式，您可以將它們兩者適時調整以配合您的需求，但請務必了解如此一來對應用程式產生的影響程度。

若要判斷哪些值用於重試和逾時，請考慮以下資訊：

- 當網路連線降級或 AWS 服務無法連線時， 適用於 .NET 的 AWS SDK 和您的應用程式應如何回應？ 您是否希望快速呼叫失敗，還是應該呼叫保持重試？
- 您的應用程式是使用者接觸應用程式，或是必須要有所回應的網站，或者是對延長的延遲時間具有更高容忍力的背景執行工作？
- 應用程式部署在低延遲的可靠網路上，還是部署在連線不可靠的遠端位置？

重試

概觀

適用於 .NET 的 AWS SDK 可以重試因伺服器端限流或連線中斷而失敗的請求。服務組態類別有兩個屬性，可用來指定服務用戶端的重試行為。服務組態類別會從 適用於 .NET 的 AWS SDK API 參考的抽象 Amazon.Runtime.ClientConfig 類別繼承這些屬性：

- `RetryMode` 指定三種重試模式之一，這些模式在 Amazon.Runtime.RequestRetryMode 列舉中定義。

  您可以使用`AWS_RETRY_MODE`環境變數或共用組態檔案中的 retry_mode AWS 設定來控制應用程式的預設值。

- `MaxErrorRetry` 指定服務用戶端層級允許的重試次數；軟體開發套件會在失敗和擲回例外狀況之前，以指定的次數重試操作。

  您可以使用`AWS_MAX_ATTEMPTS`環境變數或共用組態檔案中的 max_attempts AWS 設定來控制應用程式的預設值。

您可以在 適用於 .NET 的 AWS SDK API 參考的摘要 Amazon.Runtime.ClientConfig 類別中找到這些屬性的詳細說明。根據預設， 的每個值都會`RetryMode`對應至 的特定值`MaxErrorRetry`，如下表所示。

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

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

Behavior (行為)

您的應用程式啟動時

當您的應用程式啟動時，軟體開發套件`MaxErrorRetry`會設定 `RetryMode`和 的預設值。除非您指定其他值，否則建立服務用戶端時會使用這些預設值。

- 如果您的環境中未設定屬性，則 的預設值`RetryMode`會設定為 Legacy，而 的預設值`MaxErrorRetry`會設定為上表中的對應值。
- 如果您的環境中已設定重試模式，則該值會用作 的預設值`RetryMode`。除非已在您的環境中設定最大錯誤的值（接下來說明），否則 的預設值`MaxErrorRetry`會使用上表中的對應值進行設定。
- 如果您的環境中已設定最大錯誤的值，則該值會用作 的預設值`MaxErrorRetry`。Amazon DynamoDB 是此規則的例外狀況； 的預設 DynamoDB 值一律`MaxErrorRetry`是上表中的值。

當您的應用程式執行時

建立服務用戶端時，您可以使用 `RetryMode`和 的預設值`MaxErrorRetry`，如先前所述，也可以指定其他值。若要指定其他值，請在建立服務用戶端時建立並包含服務組態物件，例如 AmazonDynamoDBConfig 或 AmazonSQSConfig。

建立服務用戶端之後，就無法變更這些值。

考量

重試時，請求的延遲會增加。您應該設定您的重試，依據您的應用程式限制總請求延遲和錯誤率而設定。

逾時

適用於 .NET 的 AWS SDK 可讓您在服務用戶端層級和每個方法呼叫設定請求逾時。設定逾時有兩種機制，這些機制涵蓋在後續章節中：

- 如果您使用的是非同步呼叫，則可以使用 方法的 `CancellationToken` 參數。

- 如果您在 .NET Framework 中使用同步呼叫，則可以使用抽象 Amazon.Runtime.ClientConfig 類別的 Timeout和 ReadWriteTimeout 屬性。

針對逾時使用 CancellationToken 參數

適用於 .NET 的 AWS SDK 可讓您使用 CancellationToken 參數，在非同步呼叫上設定請求逾時。下列程式碼片段顯示範例。System.Threading.Tasks.TaskCanceledException 如果未在 10 秒內完成請求，程式碼會擲回。

```
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);
}
```

將 Timeout和 ReadWriteTimeout 屬性用於逾時

> **Note**
>
> Timeout 屬性不會影響非同步呼叫。如果您使用的是非同步呼叫，請針對逾時使用 CancellationToken 參數改為參閱 。

適用於 .NET 的 AWS SDK 可讓您在服務用戶端層級設定請求逾時和通訊端讀取/寫入逾時值。這些值是在 Timeout和抽象 Amazon.Runtime.ClientConfig 類別的 ReadWriteTimeout 屬性中指定。這些值會做為 AWS 服務用戶端物件所建立 HttpWebRequest 物件的 Timeout和 ReadWriteTimeout 屬性傳遞。在預設情況下，Timeout 值為 100 秒，ReadWriteTimeout 值為 300 秒。

當您的網路有高延遲或存在會造成操作重試的條件，使用長逾時值和次數高的重試，會導致某些開發套件的操作看起來沒有回應。

> **Note**
>
> 以可攜式類別程式庫 (PCL) 為 適用於 .NET 的 AWS SDK 目標的 版本會使用 HttpClient 類別而非 HttpWebRequest類別，並且僅支援逾時屬性。

以下為例外狀況的預設逾時值。這些值時在您明確設定逾時值時會被覆寫。

- Timeout 如果呼叫的方法上傳串流，例如 AmazonS3Client.PutObjectAsync()、AmazonS3Client.UploadPartAsync()、AmazonGlacierClient.UploadA 等，則 和 ReadWriteTimeout會設定為最大值。
- 適用於 .NET 的 AWS SDK 目標為 .NET Framework 設定的 版本ReadWriteTimeout，Timeout以及所有 AmazonS3Client 和 AmazonGlacierClient 物件的最大值。
- 適用於 .NET 的 AWS SDK 將可攜式類別程式庫 (PCL) 和 .NET Core 設為所有 AmazonS3Client 和 AmazonGlacierClient 物件Timeout最大值的 版本。

下列範例說明如何指定標準重試模式、最多 3 次重試、逾時 10 秒，以及讀取/寫入逾時 10 秒 （ 如適用 ）。AmazonS3Client 建構函數會獲得 AmazonS3Config 物件。

```
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
  });
```

可觀測性

可觀測性是從系統發出的資料推斷系統目前狀態的程度。發出的資料通常稱為遙測。

適用於 .NET 的 AWS SDK 可以提供兩種常見的遙測訊號、指標和追蹤以及記錄。您可以連接 TelemetryProvider，將遙測資料傳送至可觀測性後端 （ 例如 AWS X-Ray或 Amazon CloudWatch)，然後對其採取行動。

根據預設，在 SDK 中會停用遙測訊號。本主題說明如何啟用和設定遙測輸出。

其他資源

如需啟用和使用可觀測性的詳細資訊，請參閱下列資源：

- OpenTelemetry
- 部落格文章適用於 .NET 的 AWS SDK 使用 OpenTelemetry 在 中增強可觀測性。
- 部落格文章宣布 AWS .NET OpenTelemetry 程式庫的一般可用性。
- OpenTelemetry 的匯出工具
- 如需 中可觀測性的範例 AWS Tools for PowerShell，請參閱《Tools for PowerShell 使用者指南》中的可觀測性。

設定 `TelemetryProvider`

您可以針對所有服務用戶端或個別用戶端`TelemetryProvider`，在應用程式中全域設定 ，如下列範例所示。the section called “遙測供應商” 本節包含遙測實作的相關資訊，包括 SDK 隨附的實作相關資訊。

設定預設全域遙測供應商

根據預設，每個服務用戶端都會嘗試使用全球可用的遙測提供者。如此一來，您可以設定一次提供者，所有用戶端都會使用它。在您建立任何服務用戶端之前，這應該只完成一次。

下列程式碼片段說明如何設定全域遙測供應商。然後，它會建立 Amazon S3 服務用戶端，並嘗試執行失敗的操作。程式碼會將追蹤和指標同時新增至應用程式。此程式碼使用以下 NuGet 套件：`OpenTelemetry.Exporter.Console`和 `OpenTelemetry.Instrumentation.AWS`。

> **Note**
>
> 如果您使用 AWS IAM Identity Center 進行身分驗證，請務必同時新增 `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();
```

為特定服務用戶端設定遙測供應商

您可以使用特定的遙測供應商（全域提供者除外）來設定個別服務用戶端。若要這樣做，請使用服務用戶端建構函數的 Config 物件`TelemetryProvider`類別。例如，請參閱 AmazonS3Config 並尋找 `TelemetryProvider` 屬性。如需自訂遙測實作的相關資訊the section called “遙測供應商”，請參閱 。

主題

- 指標

- 遙測供應商

指標

下表列出 SDK 發出的遙測指標。設定遙測提供者，讓指標可被觀察。

發出哪些指標？

指標名稱	單位	Type	Attributes	描述
client.call.duration	s	直方圖	rpc.service、rpc.method	整體通話持續時間（包括傳送或接收請求和回應內文的重試次數和時間）
client.uptime	s	直方圖	rpc.service	自建立用戶端以來的時間量
client.call.attempts	{attempt}	MonotonCounter	rpc.service、rpc.method	個別操作的嘗試次數
client.call.errors	{error}	MonotonCounter	rpc.service、rpc.method、 exception.type	操作的錯誤數目
client.call.attempt_duration	s	直方圖	rpc.service、rpc.method	連線到服務、傳送請求，以及取回 HTTP 狀態碼和標頭所需的時間（包括排入佇列等待傳送的時間）
client.call.resolve_endpoint_duration	s	直方圖	rpc.service、rpc.method	解析請求的端點（端點解析程式，而非 DNS) 所需的時間
client.call.serialization_duration	s	直方圖	rpc.service、rpc.method	序列化訊息內文所需的時間
client.call.deserialization_duration	s	直方圖	rpc.service、rpc.method	還原序列化訊息內文所需的時間
client.call.auth.signing_duration	s	直方圖	rpc.service、rpc.method	簽署請求所需的時間

指標名稱	單位	Type	Attributes	描述
client.call.auth.resolve_identity_duration	s	直方圖	rpc.service、rpc.method	從身分提供者取得身分（例如 AWS 憑證或承載字符）所需的時間
client.http.bytes_sent	根據	MonotonCounter	server.address	HTTP 用戶端傳送的位元組總數
client.http.bytes_received	根據	MonotonCounter	server.address	HTTP 用戶端收到的位元組總數

以下是欄描述：

- 指標名稱 – 發出的指標名稱。
- 單位 – 指標的度量單位。單位以區分https://unitsofmeasure.org/ucum大小寫 ("c/s") 表示法表示。
- 類型 – 用來擷取指標的檢測類型。
- 屬性 – 指標發出的一組屬性（維度）。
- 描述 – 指標測量內容的描述。

遙測供應商

開發套件提供 OpenTelemetry 做為遙測提供者的實作，下一節將說明。

如果您有特定的遙測需求、已有遙測解決方案，或需要精細控制遙測資料的擷取和處理方式，您也可以實作自己的遙測供應商。

向 TelemetryProvider 類別註冊您自己的實作。以下是如何註冊您自己的 `TracerProvider`和 的簡單範例`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());
```

主題

- 設定 OpenTelemetry 型遙測供應商

設定 OpenTelemetry 型遙測供應商

適用於 .NET 的 AWS SDK 包含 OpenTelemetry 型遙測供應商的實作。如需如何將此提供者設定為全域遙測提供者的詳細資訊，請參閱 設定 `TelemetryProvider`。若要使用此遙測提供者，您需要專案中的下列資源：

- OpenTelemetry.Instrumentation.AWS NuGet 套件。
- 遙測匯出工具，例如 OTLP 或主控台。如需詳細資訊，請參閱 OpenTelemetry 文件中的匯出工具。

開發套件隨附的 OpenTelemetry 實作可設定為減少 HTTPS 請求、憑證和壓縮的追蹤量。若要這樣做，請將 `SuppressDownstreamInstrumentation`選項設定為 `true`，如下所示：

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

如需有關此提供者的其他資訊，請參閱 中的部落格文章適用於 .NET 的 AWS SDK 使用 OpenTelemetry 增強可觀測性。

有關使用者和角色的其他資訊

若要在 上進行 .NET 開發， AWS 或在 上執行 .NET 應用程式 AWS，您需要結合一些適合這些任務的使用者、許可集和服務角色。

您建立的特定使用者、許可集和服務角色，以及您使用這些角色的方式，將取決於應用程式的需求。以下提供一些額外的資訊，幫助您了解可能使用它們的原因，以及建立的方法。

使用者和許可集合

雖然可以使用具有長期憑證的 IAM 使用者帳戶來存取 AWS 服務，但這不再是最佳實務，應該避免使用。即使在開發期間，最佳實務是在 中建立使用者和許可集， AWS IAM Identity Center 並使用身分來源提供的臨時憑證。

若是開發環境，您可以使用您在 使用 驗證 AWS 中建立或提供給您的使用者。如果您有適當的 AWS Management Console 許可，您也可以為該使用者建立具有最低權限的不同許可集，或為開發專案建立專門為開發專案建立新的使用者，以提供最低權限的許可集。您選擇的行動方式 (如有的話) 取決於您的情況。

如需有關這些使用者和許可集合，以及如何建立它們的詳細資訊，請參閱 AWS SDK 和工具參考指南中的身分驗證和存取和 AWS IAM Identity Center 使用者指南中的入門。

服務角色

您可以設定 AWS 服務角色，以代表使用者存取 AWS 服務。如果有多人將遠端執行您的應用程式，則此類型的存取是適當的；例如，在您為此目的建立的 Amazon EC2 執行個體上。

建立服務角色的程序會根據情況而有所不同，但基本上如下所示。

1. 登入 AWS Management Console ，並在 https://console.aws.amazon.com/iam/ ：//www. 開啟 IAM 主控台。
2. 選擇 Roles (角色)，然後選擇 Create role (建立角色)。
3. 選擇 AWS 服務，尋找並選取 EC2 (範例)，然後選擇 EC2 使用案例 (範例)。
4. 選擇下一步：許可，並為應用程式將使用 AWS 的服務選取適當的政策。

   > **Warning**
   >
   > 請勿選擇 AdministratorAccess 政策，因為該政策會啟用您帳戶中幾乎所有內容的讀取和寫入許可。

5. 選擇下一步：標籤，然後輸入任何您想要的標籤。

   您可以在《IAM 使用者指南》中找到使用 AWS 資源標籤控制存取中標籤的相關資訊。
6. 選擇下一步：檢閱並提供角色名稱和角色描述。然後選擇 Create role (建立角色)。

您可以在 IAM 使用者指南中的身分（使用者、群組和角色）中找到有關 IAM 角色的高階資訊。尋找該指南 IAM 角色主題中角色的詳細資訊。

角色的其他資訊

- 使用適用於 Amazon Elastic Container Service (Amazon ECS) 任務的任務 IAM 角色。
- 使用在 Amazon EC2 執行個體上執行的應用程式的 IAM 角色。

適用於 .NET 的 AWS SDK 專案的進階組態

本節中的主題包含您可能感興趣的其他組態任務和方法的相關資訊。

主題

- 使用 AWSSDK.Extensions.NETCore.Setup 和 IConfiguration 介面
- 設定其他應用程式參數
- 的組態檔案參考 適用於 .NET 的 AWS SDK

使用 AWSSDK.Extensions.NETCore.Setup 和 IConfiguration 介面

（本主題先前標題為「適用於 .NET 的 AWS SDK 使用 .NET Core 設定」）

.NET Core 中最大的變更之一是移除 `ConfigurationManager`以及與 .NET Framework `app.config`和 ASP.NET 應用程式搭配使用的標準 和 `web.config` 檔案。

.NET Core 中的組態是以組態提供者建立的鍵值對為基礎。組態提供者會將組態資料從各種組態來源讀入鍵/值組，包括命令列引數、目錄檔案、環境變數和設定檔案。

> **Note**
>
> 如需詳細資訊，請參閱 ASP.NET Core 中的組態。

若要輕鬆 適用於 .NET 的 AWS SDK 搭配 .NET Core 使用 ，您可以使用 AWSSDK.Extensions.NETCore.Setup NuGet 套件。與許多 .NET Core 程式庫一樣，它會將延伸方法新增至`IConfiguration`界面，讓 AWS 組態無縫取得。

此套件的原始碼位於 GitHub，網址為 https://github.com/aws/aws-sdk-net/tree/main/extensions/src/AWSSDK.Extensions.NETCore.Setup。

使用 AWSSDK.Extensions.NETCore.Setup

假設您建立 ASP.NET Core Model-View-Controller (MVC) 應用程式，這可以透過 Visual Studio 中的 ASP.NET Core Web 應用程式範本或在 .NET Core CLI `dotnet new mvc ...`中執行來完成。當您建立這類應用程式時， 的建構函數會透過從`Startup.cs`組態提供者讀取各種輸入來源來處理組態，例如 `appsettings.json`。

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

若要使用 `Configuration` 物件取得AWS選項，請先新增 `AWSSDK.Extensions.NETCore.Setup` NuGet 套件。然後，將您的選項新增至組態檔案，如下所述。

請注意，新增至專案的其中一個檔案是 `appsettings.Development.json`。這相當於`EnvironmentName`設定為開發。在開發期間，您會將組態放入此檔案中，這只會在本機測試期間讀取。當您部署已`EnvironmentName`設定為生產的 Amazon EC2 執行個體時，會忽略此檔案，而 適用於 .NET 的 AWS SDK 會回到為 Amazon EC2 執行個體設定的 IAM 登入資料和區域。

下列組態設定顯示您可以在專案的 `appsettings.Development.json` 檔案中新增以提供 AWS 設定的值範例。

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

若要存取 CSHTML 檔案中的設定，請使用 `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>
```

若要從程式碼存取檔案中設定 AWS 的選項，請呼叫新增至 的GetAWSOptions擴充功能方法IConfiguration。

若要建構這些選項的服務用戶端，請呼叫 CreateServiceClient。下列範例示範如何建立 Amazon S3 服務用戶端。（請務必將 AWSSDK.S3 NuGet 套件新增至您的專案。）

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

您也可以使用 appsettings.Development.json 檔案中的多個項目建立具有不相容設定的多個服務用戶端，如下列範例所示，其中 的組態service1包含 us-west-2區域，而 的組態service2包含特殊端點 URL。

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

然後，您可以使用 JSON 檔案中的項目，取得特定服務的選項。例如，若要取得 的設定，service1請使用下列項目。

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

Appsettings 檔案中允許的值

以下應用程式組態的值可以在 appsettings.Development.json 檔案中做設定。欄位名稱必須使用顯示的大小寫。如需這些設定的詳細資訊，請參閱 AWS.Runtime.ClientConfig類別。

- 區域
- 設定檔
- ProfilesLocation

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

ASP.NET Core 相依性插入

AWSSDK.Extensions.NETCore.Setup NuGet 套件也能在 ASP.NET 核心中使用新的相依性注入系統。您應用程式`Startup`類別中新增 MVC 服務`ConfigureServices`的方法。如果應用程式是使用實體架構，那麼這也會是初始化的位置。

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

> **Note**
>
> .NET Core 中的相依性注入背景可在 .NET Core 文件網站上取得。

`AWSSDK.Extensions.NETCore.Setup` NuGet 套件會將新的延伸方法新增至 `IServiceCollection` ，您可以用來將 AWS 服務新增至相依性插入。下列程式碼說明如何新增讀

取自 AWS 的選項IConfiguration，以將 Amazon S3 和 DynamoDB 新增至 服務清單。（請務必將 AWSSDK.S3 和 AWSSDK.DynamoDBv2 NuGet 套件新增至您的專案。）

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

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

現在，如果您的 MVC 控制器使用 IAmazonS3 或 IAmazonDynamoDB 參數做為建構函數裡的參數，相依性注入系統傳入這些服務。

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

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

    ...

}
```

設定其他應用程式參數

> **Note**
>
> 本主題中的資訊專屬於以 .NET Framework 為基礎的專案。根據預設， App.config和 Web.config 檔案不會出現在以 .NET Core 為基礎的專案中。

開啟以檢視 .NET Framework 內容

您可以設定許多應用程式參數：

- AWSLogging
- AWSLogMetrics
- AWSRegion
- AWSResponseLogging
- AWS.DynamoDBContext.TableNamePrefix
- AWS.S3.UseSignatureVersion4
- AWSEndpointDefinition
- AWS 服務產生的端點

這些參數可以在應用程式的 App.config 或 Web.config 檔案中設定。雖然您也可以使用 適用於 .NET 的 AWS SDK API 設定這些項目，但我們建議您使用應用程式的 .config 檔案。兩種方法皆在這裡有所描述。

如需有關如本主題稍後所述使用 <aws>元素的詳細資訊，請參閱 的組態檔案參考 適用於 .NET 的 AWS SDK。

AWSLogging

設定開發套件應如何記錄事件，如果有的話。例如，建議使用的方法是使用 <logging> 元素，此為 <aws> 元素的子元素：

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

或使用：

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

可能值如下：

None

關閉 事件日誌。此為預設值。

log4net

使用 log4net 記錄。

SystemDiagnostics

使用 System.Diagnostics 類別記錄。

您可以為 logTo 屬性設定多個值，以逗號分隔即可。以下示範設定 .config 檔案的 log4net 和 System.Diagnostics 記錄在：

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

或使用：

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

或者，使用 適用於 .NET 的 AWS SDK API，結合 LoggingOptions 列舉的值，並設定 AWSConfigs.Logging 屬性：

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

此設定的變更只會對新的 AWS 用戶端執行個體生效。

AWSLogMetrics

指定開發套件是否應該記錄效能指標。若要在 .config 檔案裡設定指標記錄組態，請設定 <logging> 元素的 logMetrics 屬性值，此元素為 <aws> 元素的子元素：

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

或者，在 <appSettings> 設定 AWSLogMetrics 金鑰：

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

或者，若要使用 適用於 .NET 的 AWS SDK API 設定指標記錄，請設定 AWSConfigs.LogMetrics 屬性：

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

這個設定為所有用戶端/組態設定預設的 LogMetrics 屬性。此設定的變更只會對新的 AWS 用戶端執行個體生效。

AWSRegion

為未明確指定 AWS 區域之用戶端設定預設區域。若要在 .config 檔案中設定區域，建議的方法為設定 aws 元素裡的 region 屬性值：

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

或者，在 <appSettings>區段中設定 AWSRegion 金鑰：

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

或者，若要使用 適用於 .NET 的 AWS SDK API 設定區域，請設定 AWSConfigs.AWSRegion 屬性：

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

如需為特定區域建立 AWS 用戶端的詳細資訊，請參閱AWS 區域選擇。此設定的變更只會對新的 AWS 用戶端執行個體生效。

AWSResponseLogging

設定開發套件應何時記錄服務回應。可能值如下：

Never

永遠不用記錄服務回應。此為預設值。

Always

一直記錄服務回應。

OnError

只在發生錯誤時記錄服務回應。

若要在 .config 檔案裡設定服務記錄組態，建議方式為設定 <logging> 元素的 logResponses 屬性值，此元素為 <aws> 元素的子元素：

```
<aws>
```

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

或者，在 <appSettings>區段中設定 AWSResponseLogging 金鑰：

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

或者，若要使用 適用於 .NET 的 AWS SDK API 設定服務記錄，請將 AWSConfigs.ResponseLogging 屬性設定為 ResponseLoggingOption 列舉的其中一個值：

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

這些設定的變更會立即生效。

AWS.DynamoDBContext.TableNamePrefix

設定預設 TableNamePrefix DynamoDBContext 將會在未手動設定的情況下。

若要設定 .config 檔案的表格的前綴名稱，建議的方法為設定 <dynamoDB> 元素的子元素 <dynamoDBContext> 元素之 tableNamePrefix 屬性值，此元素其本身又為 <aws> 的子元素：

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

或者，在 <appSettings> 設定 AWS.DynamoDBContext.TableNamePrefix 金鑰：

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

或者，若要使用 適用於 .NET 的 AWS SDK API 設定資料表名稱字首，請設定 AWSConfigs.DynamoDBContextTableNamePrefix 屬性：

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

此設定的變更僅對 DynamoDBContextConfig 和 DynamoDBContext 新建立的執行個體生效。

AWS.S3.UseSignatureVersion4

設定 Amazon S3 用戶端是否應使用簽章第 4 版簽署請求。

若要在 .config 檔案中設定 Amazon S3 的簽章第 4 版簽署，建議方法是設定 <s3>元素的useSignatureVersion4屬性，這是 <aws>元素的子元素：

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

或者，true在 <appSettings>區段中將AWS.S3.UseSignatureVersion4金鑰設定為 ：

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

或者，若要使用 適用於 .NET 的 AWS SDK API 設定簽章第 4 版簽署，請將 AWSConfigs.S3UseSignatureVersion4 屬性設定為 true：

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

在預設情況下，此設定為 false，但第四版的簽章可能在某些情況下或某些區域中會做為預設使用。當設定為 true，第四版的簽章將用於所有的請求。此設定的變更僅適用於新的 Amazon S3 用戶端執行個體。

AWSEndpointDefinition

設定開發套件是否應開始用自訂組態檔案定義區域和端點。

若要設定 .config 檔案的端點定義檔案，我們建議設定 <aws> 元素的 endpointDefinition 屬性值。

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

或者，您可以在 <appSettings>區段中設定 AWSEndpointDefinition 金鑰：

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

或者，若要使用 適用於 .NET 的 AWS SDK API 設定端點定義檔案，請設定 AWSConfigs.EndpointDefinition 屬性：

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

如果沒有提供檔案名稱，則無法使用自訂組態檔案。此設定的變更只會對新的 AWS 用戶端執行個體生效。endpoints.json 檔案可從 取得https://github.com/aws/aws-sdk-net/blob/main/sdk/src/Core/endpoints.json。

AWS 服務產生的端點

有些 AWS 服務會產生自己的端點，而不是使用區域端點。這些服務的用戶端使用特定於該服務與特定於您的資源的服務 URL。這些服務的兩個範例是 Amazon CloudSearch 和 AWS IoT。以下範例說明如何取得這些服務的端點。

Amazon CloudSearch 端點範例

Amazon CloudSearch 用戶端用於存取 Amazon CloudSearch 組態服務。您可使用 Amazon CloudSearch 組態服務來建立、設定和管理搜尋網域。若要建立搜尋網域，請建立 CreateDomainRequest 物件並提供 DomainName 屬性。使用請求物件建立 AmazonCloudSearchClient 物件。呼叫 CreateDomain 方法。呼叫傳回的 CreateDomainResponse 物件包含 DomainStatus 屬性，同時具有 DocService 和 SearchService 端點。建立 AmazonCloudSearchDomainConfig 物件，並使用它來初始化 AmazonCloudSearchDomainClient 類別的 DocService和 SearchService執行個體。

```
// 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 端點範例

若要取得 的端點 AWS IoT，請建立 AmazonIoTClient 物件並呼叫 DescribeEndPoint 方法。傳回的 DescribeEndPointResponse 物件包含 `EndpointAddress`。建立 AmazonIotDataConfig 物件，設定 `ServiceURL` 屬性，並使用此物件執行個體化 AmazonIotDataClient 類別。

```
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");
}
```

的組態檔案參考 適用於 .NET 的 AWS SDK

> **Note**
>
> 本主題中的資訊專屬於以 .NET Framework 為基礎的專案。根據預設， App.config和 Web.config 檔案不會出現在以 .NET Core 為基礎的專案中。

開啟以檢視 .NET Framework 內容

您可以使用 .NET 專案的 App.config或 Web.config 檔案來指定 AWS 設定，例如 AWS 登入資料、記錄選項、 AWS 服務端點和 AWS 區域，以及一些 AWS 服務設定，例如 Amazon DynamoDB、Amazon EC2 和 Amazon S3。以下資訊說明如何正確地格式化 App.config 或 Web.config 檔案，以指定這些類型的設定。

> **Note**
>
> 雖然您可以繼續使用 App.config或 Web.config 檔案中的 <appSettings> 元素來指定 AWS 設定，但我們建議您如本主題稍後所述使用 <configSections>和 <aws>元素。如需 <appSettings>元素的詳細資訊，請參閱設定 適用於 .NET 的 AWS SDK 應用程式中的 <appSettings>元素範例。

> **Note**
>
> 雖然您可以繼續使用程式碼檔案中的下列AWSConfigs類別屬性來指定 AWS 設定，但下列屬性已棄用，未來版本可能不支援：
>
> - DynamoDBContextTableNamePrefix

- EC2UseSignatureVersion4
- LoggingOptions
- LogMetrics
- ResponseLoggingOption
- S3UseSignatureVersion4

一般而言，建議您不要在程式碼檔案中使用AWSConfigs類別屬性來指定 AWS 設定，而是使用 App.config或 Web.config 檔案中的 <configSections>和 <aws>元素來指定 AWS 設定，如本主題稍後所述。如需上述屬性的詳細資訊，請參閱設定 適用於 .NET 的 AWS SDK 應用程式中的AWSConfigs程式碼範例。

主題

- 宣告 AWS 設定區段
- 允許的元素
- 元素參考

宣告 AWS 設定區段

您可以從 <aws>元素的 App.config或 Web.config 檔案中指定 AWS 設定。在您開始使用 <aws> 元素前，您必須建立一個 <section> 元素 (為 <configSections> 元素的子元素)，並設其本身的 name 屬性為 aws，其本身的 type 屬性為 Amazon.AWSSection, AWSSDK.Core，如以下範例所示：

```
<?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 編輯器不提供 <aws> 元素或其子元素完成自動的程式碼 (IntelliSense)。

為了協助您建立 <aws> 元素的正確格式版本，請呼叫 Amazon.AWSConfigs.GenerateConfigTemplate 方法。此輸出 <aws> 元素的正式版本，適合列印字串的方式，可依您的需求調整。以下章節說明 <aws> 元素的屬性和子元素。

允許的元素

以下是設定區段中 AWS 允許元素之間的邏輯關係清單。您可以呼叫 Amazon.AWSConfigs.GenerateConfigTemplate 方法產生最新的版本，也會輸出 <aws> 元素的正式版本，以字串方式呈現，所以您可以依需求調整。

```
<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>
```

元素參考

以下是 AWS 設定區段中允許的元素清單。允許每個元素的屬性和其父子元素都已列出。

主題

- 別名
- aws
- dynamoDB
- dynamoDBContext
- ec2
- 日誌
- 映射
- 屬性
- proxy
- s3

別名

<alias> 元素表示在一個或多個從表格 (from-table) 至到表格 (to-table) 對應的單一項目，為類型設定指定不同的表格。此元素會對應到 適用於 .NET 的 AWS SDK 之 Amazon.AWSConfigs.DynamoDBConfig.Context.TableAliases 屬性中 Amazon.Util.TableAlias 類別的執行個體。在套用表格的前綴名稱前，會完成重新對應。

這個元素可以包含下列屬性：

fromTable

從表格 (from-table) 至到表格 (to-table) 對應的從表格 (from-table) 部分。此屬性對應到 適用於 .NET 的 AWS SDK中的 Amazon.Util.TableAlias.FromTable 屬性。

toTable

從表格 (from-table) 至到表格 (to-table) 對應的至表格 (to-table) 部分。此屬性對應到 適用於 .NET 的 AWS SDK中的 Amazon.Util.TableAlias.ToTable 屬性。

<alias> 元素的父系為 <tableAliases> 元素。

<alias> 元素不含子元素。

以下為範例使用的 <alias> 元素：

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

aws

<aws> 元素代表 AWS 設定區段中最上層的元素。這個元素可以包含下列屬性：

endpointDefinition

自訂組態檔案的絕對路徑，定義要使用 AWS 的區域和端點。此屬性對應到 適用於 .NET 的 AWS SDK中的 Amazon.AWSConfigs.EndpointDefinition 屬性。

profileName

用於進行服務呼叫之已儲存 AWS 登入資料的設定檔名稱。此屬性對應到 適用於 .NET 的 AWS SDK中的 Amazon.AWSConfigs.AWSProfileName 屬性。

profilesLocation

與其他 AWS SDKs 共用的登入資料檔案位置的絕對路徑。預設情況下，登入資料檔案存放於目前使用者主目錄的 .aws 目錄中。此屬性對應到 適用於 .NET 的 AWS SDK中的 Amazon.AWSConfigs.AWSProfilesLocation 屬性。

region

未明確指定 AWS 區域之用戶端的預設區域 ID。此屬性對應到 適用於 .NET 的 AWS SDK中的 Amazon.AWSConfigs.AWSRegion 屬性。

<aws> 元素沒有父元素。

<aws> 元素可以含有以下的子元素：

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

以下為範例使用的 <aws> 元素：

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

dynamoDB

<dynamoDB> 元素代表 AmazonDynamoDB 設定的設定集合。這個元素可以包含 conversionSchema 屬性，這代表使用 .NET 和 DynamoDB 物件之間轉換。允許值包含 V1 和 V2。此屬性對應到 適用於 .NET 的 AWS SDK中的 Amazon.DynamoDBv2.DynamoDBEntryConversion 類別。如需詳細資訊，請參閱 DynamoDB 系列 - 轉換結構描述。

<dynamoDB> 元素的父系為 <aws> 元素。

<dynamoDB> 元素可以含有 <dynamoDBContext> 子元素。

以下為範例使用的 <dynamoDB> 元素：

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

dynamoDBContext

<dynamoDBContext> 元素代表 Amazon DynamoDB 內容專屬的設定集合。這個元素可以包含 tableNamePrefix 屬性，這代表 DynamoDB 內容將使用預設的表格的前綴名稱，如果未手動設定的話。此屬性從 適用於 .NET 的 AWS SDK中的 Amazon.AWSConfigs.DynamoDBConfig.Context.TableNamePrefix 屬性對應到 Amazon.Util.DynamoDBContextConfig.TableNamePrefix 屬性。如需詳細資訊，請參閱 DynamoDB 開發套件的增強功能。

<dynamoDBContext> 元素的父系為 <dynamoDB> 元素。

<dynamoDBContext> 元素可以含有以下的子元素：

- <alias> (一個或多個執行個體)
- <map> (一個或多個執行個體)

以下為範例使用的 <dynamoDBContext> 元素：

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

ec2

<ec2> 元素代表 Amazon EC2 設定的集合。這個元素可以包含 useSignatureVersion4 屬性，指定第四版的簽章簽署將用於所有請求 (true) 或第四版的簽章簽署將不會用於所有請求 (false，預設)。此屬性從 適用於 .NET 的 AWS SDK中的 Amazon.AWSConfigs.EC2Config.UseSignatureVersion4 屬性對應到 Amazon.Util.EC2Config.UseSignatureVersion4 屬性。

<ec2> 的父系為此元素。

<ec2> 元素不含子元素。

以下為範例使用的 <ec2> 元素：

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

日誌

<logging> 元素表示回應日誌記錄和效能指標記錄的設定集合。這個元素可以包含下列屬性：

logMetrics

無論效能指標記錄是否適用於所有用戶端和組態 (true)，否則 false。此屬性從 適用於 .NET 的 AWS SDK中的 Amazon.AWSConfigs.LoggingConfig.LogMetrics 屬性對應到 Amazon.Util.LoggingConfig.LogMetrics 屬性。

logMetricsCustomFormatter

用於記錄指標的自訂格式化之資料類型和組件名稱。此屬性從 適用於 .NET 的 AWS SDK中的 Amazon.AWSConfigs.LoggingConfig.LogMetricsCustomFormatter 屬性對應到 Amazon.Util.LoggingConfig.LogMetricsCustomFormatter 屬性。

logMetricsFormat

記錄指標的顯示格式 (從 適用於 .NET 的 AWS SDK中的 Amazon.AWSConfigs.LoggingConfig.LogMetricsFormat 屬性對應到 Amazon.Util.LoggingConfig.LogMetricsFormat 屬性)。

允許數值包括：

JSON

使用 JSON 格式。

Standard

使用預設格。

logResponses

日誌服務回應的時機 (從 適用於 .NET 的 AWS SDK中的 Amazon.AWSConfigs.LoggingConfig.LogResponses 屬性對應到 Amazon.Util.LoggingConfig.LogResponses 屬性)。

允許數值包括：

Always

一直記錄服務回應。

Never

永遠不用記錄服務回應。

OnError

只在發生錯誤時，記錄服務回應。

logTo

登入位置 （ 從 中的 LogTo 屬性映射至 Amazon.AWSConfigs.LoggingConfig.LogTo 屬性 適用於 .NET 的 AWS SDK)。

允許的值包含一個或多個：

Log4Net

記錄到 log4net。

None

停用日誌記錄。

SystemDiagnostics

記錄到 System.Diagnostics。

<logging> 元素的父系為 <aws> 元素。

<logging> 元素不含子元素。

以下為範例使用的 <logging> 元素：

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

映射

<map> 元素表示從 .NET 類型對應到 DynamoDB 資料表 (從 適用於 .NET 的 AWS SDK中的 Amazon.AWSConfigs.DynamoDBConfig.Context.TypeMappings 屬性對應到 TypeMapping 類別的執行個體) 之類型到資料表 (type-to-table) 對應集合中的單一項目，如需詳細資訊，請參閱 DynamoDB 開發套件的增強功能。

這個元素可以包含下列屬性：

targetTable

對應套用的 DynamoDB 表格。此屬性對應到 適用於 .NET 的 AWS SDK中的 `Amazon.Util.TypeMapping.TargetTable` 屬性。

type

對應套用的類型和組件名稱。此屬性對應到 適用於 .NET 的 AWS SDK中的 `Amazon.Util.TypeMapping.Type` 屬性。

`<map>` 元素的父系為 `<dynamoDBContext>` 元素。

`<map>` 元素可以包含 `<property>` 子元素的一個或多個執行個體。

以下為範例使用的 `<map>` 元素：

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

屬性

`<property>` 元素表示 DynamoDB 屬性。（此元素對應至 中 AddProperty方法的 Amazon.Util.PropertyConfig 類別執行個體 適用於 .NET 的 AWS SDK) 如需詳細資訊，請參閱 DynamoDB SDK 和 DynamoDB 屬性的增強功能。 DynamoDB

這個元素可以包含下列屬性：

attribute

屬性的屬性名稱，例如範圍金鑰的名稱。此屬性對應到 適用於 .NET 的 AWS SDK中的 `Amazon.Util.PropertyConfig.Attribute` 屬性。

converter

此屬性應使用的轉換器類型。此屬性對應到 適用於 .NET 的 AWS SDK中的 `Amazon.Util.PropertyConfig.Converter` 屬性。

ignore

此關聯的屬性是否應該被忽略 (true)；否則為 false。此屬性對應到 適用於 .NET 的 AWS SDK中的 `Amazon.Util.PropertyConfig.Ignore` 屬性。

name

屬性的名稱。此屬性對應到 適用於 .NET 的 AWS SDK中的 Amazon.Util.PropertyConfig.Name 屬性。

version

此屬性是否應該存放項目版本編號 (true)；否則為 false。此屬性對應到 適用於 .NET 的 AWS SDK 中的 Amazon.Util.PropertyConfig.Version 屬性。

<property> 元素的父系為 <map> 元素。

<property> 元素不含子元素。

以下為範例使用的 <property> 元素：

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

proxy

<proxy> 元素表示用於設定 適用於 .NET 的 AWS SDK 要使用之代理程式的設定。這個元素可以包含下列屬性：

託管

代理伺服器的主機名稱或 IP 地址。此屬性從 適用於 .NET 的 AWS SDK中的 Amazon.AWSConfigs.ProxyConfig.Host 屬性對應到 Amazon.Util.ProxyConfig.Host 屬性。

密碼

此密碼使用代理伺服器的進行身分驗證。此屬性從 適用於 .NET 的 AWS SDK中的 Amazon.AWSConfigs.ProxyConfig.Password 屬性對應到 Amazon.Util.ProxyConfig.Password 屬性。

port

此代理的連接埠號碼。此屬性從 適用於 .NET 的 AWS SDK中的 Amazon.AWSConfigs.ProxyConfig.Port 屬性對應到 Amazon.Util.ProxyConfig.Port 屬性。

使用者名稱

使用代理伺服器進行身分驗證的使用者名稱。此屬性從 適用於 .NET 的 AWS SDK中的 Amazon.AWSConfigs.ProxyConfig.Username 屬性對應到 Amazon.Util.ProxyConfig.Username 屬性。

<proxy> 元素的父系為 <aws> 元素。

<proxy> 元素不含子元素。

以下為範例使用的 <proxy> 元素：

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

s3

<s3> 元素代表 Amazon S3 設定的集合。這個元素可以包含 useSignatureVersion4 屬性，指定第四版的簽章簽署將用於所有請求 (true) 或第四版的簽章簽署將不會用於所有請求 (false，預設)。此屬性對應到 適用於 .NET 的 AWS SDK中的 Amazon.AWSConfigs.S3Config.UseSignatureVersion4 屬性。

<s3> 元素的父系為 <aws> 元素。

<s3> 元素不含子元素。

以下為範例使用的 <s3> 元素：

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

使用舊版憑證

本節中的主題提供有關在不使用 AWS IAM Identity Center的情況下使用長期或短期憑證資訊。

> **Warning**
>
> 為避免安全風險，在開發專用軟體或使用真實資料時，請勿使用 IAM 使用者進行身分驗證。相反地，搭配使用聯合功能和身分提供者，例如 AWS IAM Identity Center。

> **Note**
>
> 本主題中的資訊適用於您需要手動取得及管理短期或長期憑證的情況。有關短期和長期憑證的其他資訊，請參閱 AWS SDK 和工具參考指南中的其他驗證方法。
> 如需最佳安全實務，請使用 AWS IAM Identity Center，如 中所述使用 驗證 AWS。

憑證的重要警告和指引

憑證警告

- 請勿使用您帳戶的根登入資料來存取 AWS 資源。這些登入資料可讓未管制的帳戶存取和很難撤銷這些帳戶。
- 請勿將常值存取金鑰或登入資料資訊放在應用程式檔案中。如果您不小心這麼做了，則會有暴露您登入資料的風險，例如，當您上傳專案到公有儲存庫時。
- 請勿在您的專案區域中包含包含登入資料的檔案。
- 請注意，存放在共用 AWS `credentials`檔案中的任何登入資料都會以純文字儲存。

安全管理憑證的其他指引

如需如何安全管理 AWS 登入資料的一般討論，請參閱《》中的AWS 安全登入資料AWS 一般參考和《IAM 使用者指南》中的安全最佳實務和使用案例。除了這些討論之外，請考慮下列事項：

- 建立其他使用者 (例如 IAM Identity Center 中的使用者)，並使用其憑證，而不是使用您的 AWS 根使用者憑證。如有必要，其他使用者的憑證可以被撤銷，或本質上是臨時的。此外，您可以將政策套用至每個使用者，以便僅存取特定資源和動作，從而採取最低權限許可的立場。

- 使用適用於 Amazon Elastic Container Service (Amazon ECS) 任務的任務 IAM 角色。

- 使用在 Amazon EC2 執行個體上執行的應用程式的 IAM 角色。

- 為組織外部使用者可用的應用程式使用臨時登入資料或環境變數。

主題

- 使用共用 AWS 登入資料檔案
- 使用 SDK 存放區 （僅限 Windows)

使用共用 AWS 登入資料檔案

（請務必檢閱登入資料的重要警告和指引。）

為您的應用程式提供登入資料的一種方法是在共用 AWS 登入資料檔案中建立設定檔，然後將登入資料存放在這些設定檔中。此檔案可供其他 AWS SDKs使用。它也可以由 AWS CLI、 和適用於 Visual StudioAWS Tools for Windows PowerShell、JetBrains 和 VS Code AWS 的工具組使用。

> **Warning**
>
> 為避免安全風險，在開發專用軟體或使用真實資料時，請勿使用 IAM 使用者進行身分驗證。相反地，搭配使用聯合功能和身分提供者，例如 AWS IAM Identity Center。

> **Note**
>
> 本主題中的資訊適用於您需要手動取得及管理短期或長期憑證的情況。有關短期和長期憑證的其他資訊，請參閱 AWS SDK 和工具參考指南中的其他驗證方法。
> 如需最佳實務，請使用 AWS IAM Identity Center，如 中所述使用 驗證 AWS。

一般資訊

根據預設，共用的 AWS 登入資料檔案位於主.aws目錄中的 目錄中，並命名為 `credentials`；也就是 `~/.aws/credentials`(Linux 或 macOS) 或 `%USERPROFILE%\.aws\credentials`(Windows)。如需有關替代位置的資訊，請參閱 AWS SDKs和工具參考指南中共用檔案的位置。另請參閱在應用程式中存取登入資料和設定檔。

共用的 AWS 登入資料檔案是純文字檔案，並遵循特定格式。如需 AWS 登入資料檔案格式的資訊，請參閱 AWS SDKs和工具參考指南中的登入資料檔案格式。

您可以透過多種方式管理共用 AWS 登入資料檔案中的設定檔。

- 使用任何文字編輯器來建立和更新共用的 AWS 登入資料檔案。
- 使用 適用於 .NET 的 AWS SDK API 的 Amazon.Runtime.CredentialManagement 命名空間，如本主題稍後所示。
- 使用適用於 Visual Studio、JetBrains 和 VS Code 的 AWS Tools for PowerShell和 AWS 工具組命令和程序。 https://docs.aws.amazon.com/toolkit-for-vscode/latest/userguide/setup-credentials.html
- 使用 AWS CLI 命令，例如 `aws configure set aws_access_key_id`和 `aws configure set aws_secret_access_key`。

設定檔管理的範例

下列各節顯示共用 AWS 登入資料檔案中的設定檔範例。部分範例顯示結果，可透過上述任何登入資料管理方法取得。其他範例示範如何使用特定方法。

預設設定檔

共用的 AWS 登入資料檔案幾乎一律會有名為預設值的設定檔。如果未定義其他設定檔，則 會在這裡適用於 .NET 的 AWS SDK 尋找登入資料。

`[default]` 設定檔通常如下所示。

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

以程式設計方式建立設定檔

此範例說明如何建立設定檔，並以程式設計方式將其儲存至共用的 AWS 登入資料檔案。它使用下列類別的 Amazon.Runtime.CredentialManagement 命名空間： CredentialProfileOptions、CredentialProfile 和 SharedCredentialsFile。

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

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

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

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

> **Warning**
>
> 這類程式碼通常不應在您的應用程式中。如果您將其包含在應用程式中，請採取適當的預防措施，以確保無法在程式碼中、透過網路或甚至電腦記憶體中看到純文字金鑰。

以下是此範例建立的設定檔。

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

以程式設計方式更新現有的設定檔

此範例說明如何以程式設計方式更新先前建立的設定檔。它使用下列類別的 Amazon.Runtime.CredentialManagement 命名空間： CredentialProfile 和 SharedCredentialsFile。它也會使用 Amazon 命名空間的 RegionEndpoint 類別。

```
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);
    }
}
```

以下是更新的設定檔。

```
[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設定 AWS 區域。如需詳細資訊，請參閱設定 AWS 的區域 適用於 .NET 的 AWS SDK。

使用 SDK 存放區 （僅限 Windows)

（請務必檢閱重要的警告和指導方針。）

在 Windows 上，開發套件存放區是建立設定檔和存放 適用於 .NET 的 AWS SDK 應用程式加密登入資料的另一個位置。它位於 中%USERPROFILE%\AppData\Local\AWSToolkit\RegisteredAccounts.json。您可以在開發期間使用 SDK Store 做為共用 AWS 登入資料檔案的替代方案。

> **Warning**
>
> 為避免安全風險，在開發專用軟體或使用真實資料時，請勿使用 IAM 使用者進行身分驗證。相反地，搭配使用聯合功能和身分提供者，例如 AWS IAM Identity Center。

> **Note**
>
> 本主題中的資訊適用於您需要手動取得及管理短期或長期憑證的情況。有關短期和長期憑證的其他資訊，請參閱 AWS SDK 和工具參考指南中的其他驗證方法。
> 如需最佳安全實務，請使用 AWS IAM Identity Center，如 中所述使用 驗證 AWS。

一般資訊

SDK 存放區提供下列優點：

- 軟體開發套件存放區中的登入資料會加密，而軟體開發套件存放區位於使用者的主目錄中。此可限制意外暴露您的登入資料的風險。
- 軟體開發套件存放區也提供登入資料給 AWS Tools for Windows PowerShell和 AWS Toolkit for Visual Studio。

軟體開發套件存放區描述檔是特定主機上特定使用者的特定描述檔。您無法複製它們給其他的主機或其他的使用者。這表示您無法為其他主機或開發人員機器重複使用開發機器上的 SDK Store 設定檔。這也表示您無法在生產應用程式中使用 SDK Store 設定檔。

您可以使用下列方式管理 SDK Store 中的設定檔：

- 使用 中的圖形使用者介面 (GUI)AWS Toolkit for Visual Studio。
- 使用 適用於 .NET 的 AWS SDK API 的 Amazon.Runtime.CredentialManagement 命名空間，如本主題稍後所示。
- 使用來自 的命令AWS Tools for Windows PowerShell，例如 `Set-AWSCredential`和 `Remove-AWSCredentialProfile`。

設定檔管理的範例

下列範例示範如何在 SDK Store 中以程式設計方式建立和更新設定檔。

以程式設計方式建立設定檔

此範例說明如何建立設定檔，並以程式設計方式將其儲存至 SDK 存放區。它使用下列類別的 Amazon.Runtime.CredentialManagement 命名空間： CredentialProfileOptions、CredentialProfile 和 NetSDKCredentialsFile。

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

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

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

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

> **Warning**
>
> 這類程式碼通常不應在您的應用程式中。如果您的應用程式包含 ，請採取適當的預防措施，以確保無法在程式碼中、透過網路或甚至電腦記憶體中看到純文字金鑰。

以下是此範例建立的設定檔。

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

以程式設計方式更新現有的設定檔

此範例說明如何以程式設計方式更新先前建立的設定檔。它使用下列類別的 Amazon.Runtime.CredentialManagement 命名空間： CredentialProfile 和 NetSDKCredentialsFile。它也會使用 Amazon 命名空間的 RegionEndpoint 類別。

```
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);
    }
}
```

以下是更新的設定檔。

```
"[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設定 AWS 區域。如需詳細資訊，請參閱設定 AWS 的區域 適用於 .NET 的 AWS SDK。

使用 適用於 .NET 的 AWS SDK

本節提供有關建立應用程式時 適用於 .NET 的 AWS SDK 可能需要考慮的 功能的資訊。

開始之前，請確定您已設定環境並設定專案。

如需針對特定 AWS 服務開發軟體以及程式碼範例的資訊，請參閱 呼叫 AWS 服務。如需其他程式碼範例，請參閱 適用於 .NET 的 SDK (v4) 程式碼範例。

主題

- 使用 以非同步方式進行程式設計 適用於 .NET 的 AWS SDK
- 在 中使用分頁結果 適用於 .NET 的 AWS SDK
- 在 中支援 HTTP 2 適用於 .NET 的 AWS SDK
- 其他工具

使用 以非同步方式進行程式設計 適用於 .NET 的 AWS SDK

適用於 .NET 的 AWS SDK 使用任務型非同步模式 (TAP) 進行非同步實作。若要進一步了解 TAP，請參閱 https：//docs.microsoft.com 上的任務型非同步模式 (TAP)。

本主題概述如何在呼叫 AWS 服務用戶端時使用 TAP。

適用於 .NET 的 AWS SDK API 中的非同步方法是根據 `Task`類別或 `Task<TResult>`類別的操作。如需這些類別的相關資訊，請參閱 docs.microsoft.com：//https://learn.microsoft.com/en-us/dotnet/api/system.threading.tasks.taskTResult。

當您的程式碼中呼叫這些 API 方法時，它們必須在使用 `async`關鍵字宣告的函數內呼叫，如下列範例所示。

```
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;
}
```

如上述程式碼片段所示，async宣告的偏好範圍是 Main函數。設定此async範圍可確保對 AWS 服務用戶端的所有呼叫都必須是非同步的。如果您因為某些原因而無法宣告Main為非同步，您可以在 以外的函數上使用 async關鍵字，Main然後從該處呼叫 API 方法，如下列範例所示。

```
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;
}
```

請注意，當您使用此模式Main時， 中需要的特殊Task<>語法。此外，您必須使用回應**Result**的成員來取得資料。

您可以在 建立簡單的應用程式區段 (簡單的跨平台應用程式 和 簡單的 Windows 型應用程式) 和 中查看非同步呼叫 AWS 服務用戶端的完整範例引導式程式碼範例。

在 中使用分頁結果 適用於 .NET 的 AWS SDK

有些 AWS 服務會收集和存放大量資料，您可以使用 的 API 呼叫來擷取這些資料 適用於 .NET 的 AWS SDK。如果您想要擷取的資料量對於單一 API 呼叫變得太大，您可以透過使用分頁將結果分成更易於管理的部分。

為了讓您能夠執行分頁，軟體開發套件中許多服務用戶端的請求和回應物件會提供接續字符（通常名為 NextToken)。其中一些服務用戶端也提供分頁程式。

分頁程式可讓您避免接續字符的額外負荷，這可能涉及迴圈、狀態變數、多個 API 呼叫等。當您使用分頁程式時，您可以透過迴圈foreach的宣告單行程式碼，從 AWS 服務擷取資料。如果需要多個 API 呼叫來擷取資料，分頁器會為您處理。

在哪裡可以找到分頁程式？

並非所有 服務都提供分頁程式。判斷服務是否為特定 API 提供分頁器的一種方法是查看 適用於 .NET 的 AWS SDK API 參考中服務用戶端類別的定義。

例如，如果您檢查 AmazonCloudWatchLogsClient 類別的定義，您會看到 Paginators 屬性。這是為 Amazon CloudWatch Logs 提供分頁器的 屬性。

分頁程式提供什麼？

分頁程式包含可讓您查看完整回應的屬性。它們通常也包含一或多個屬性，可讓您存取最有趣的回應部分，我們將呼叫關鍵結果。

例如，在先前AmazonCloudWatchLogsClient提到的 中， Paginator 物件包含具有 API 呼叫中完整 DescribeLogGroupsResponse 物件的 Responses 屬性。此Responses屬性包含日誌群組的集合。

Paginator 物件也包含一個名為 的金鑰結果LogGroups。此屬性只會保留回應的日誌群組部分。擁有此金鑰結果可讓您在許多情況下減少和簡化程式碼。

同步與非同步分頁

分頁程式提供同步和非同步的分頁機制。同步分頁可在 .NET Framework 4.7.2（或更新版本）專案中使用。.NET Core 專案 (.NET Core 3.1、.NET 5 等）提供非同步分頁。

由於建議使用非同步操作和 .NET Core，接下來的範例會顯示非同步分頁。有關如何使用同步分頁和 .NET Framework 4.7.2（或更新版本）執行相同任務的資訊會顯示在 中的範例之後。 分頁程式的其他考量事項

範例

下列範例說明如何使用 適用於 .NET 的 AWS SDK 來顯示日誌群組的清單。反之，此範例示範如何使用和不使用分頁程式來執行此操作。在查看稍後顯示的完整程式碼之前，請考慮下列程式碼片段。

在沒有分頁器的情況下取得 CloudWatch 日誌群組

```
    // 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));
```

使用分頁器取得 CloudWatch 日誌群組

```
    // 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);
    }
```

這兩個程式碼片段的結果完全相同，因此使用分頁器的優勢可以清楚地看到。

> **Note**
>
> 在您嘗試建置並執行完整程式碼之前，請確定您已設定環境和專案。
> 您可能還需要 Microsoft.Bcl.AsyncInterfaces NuGet 套件，因為非同步分頁程式使用 `IAsyncEnumerable` 介面。

完成程式碼

本節顯示此範例的相關參考和完整程式碼。

開發套件參考

NuGet 套件：

- AWSSDK.CloudWatch

程式設計元素：

- 命名空間 Amazon.CloudWatch

  AmazonCloudWatchLogsClient 類別
- 命名空間 Amazon.CloudWatchLogs.Model

  類別 DescribeLogGroupsRequest

  類別 DescribeLogGroupsResponse

  類別 LogGroup

完整程式碼

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

分頁程式的其他考量事項

- 分頁程式無法多次使用

  如果您需要程式碼中多個位置的特定 AWS 分頁器結果，則不得多次使用分頁器物件。反之，請在每次需要時建立新的分頁器。此概念顯示在 `DisplayLogGroupsWithPaginators`方法中的上述範例程式碼中。

- 同步分頁

  同步分頁適用於 .NET Framework 4.7.2（或更新版本）專案。

  若要查看，請建立 .NET Framework 4.7.2（或更新版本）專案，並將上述程式碼複製到其中。然後，只需從兩個`foreach`分頁器呼叫中移除`await`關鍵字，如下列範例所示。

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

建置並執行專案，以查看與非同步分頁相同的結果。

在 中支援 HTTP 2 適用於 .NET 的 AWS SDK

有些 AWS 服務和操作需要 HTTP 2。例如，Amazon Transcribe Streaming 中的雙向串流無法透過 HTTP 1.1 進行，因此需要 HTTP 2。 適用於 .NET 的 AWS SDK 新增對 HTTP 2 的 第 4 版支援，讓您可以在應用程式中使用這些操作。對於雙向 HTTP 2 操作，接收串流時 SDK 的行為類似於 HTTP 1.1 的行為。也就是說，當使用 SDK 的應用程式將事件傳送到服務時，請求會有開發人員指派的發佈者。

若要查看此行為的實際運作狀態，請考慮下列 Amazon Transcribe Streaming 的範例。它使用 Amazon.TranscribeStreaming 和 Amazon.TranscribeStreaming.Model 命名空間。

在此範例中，開發人員使用回呼函數定義 `StartStreamTranscriptionRequest.AudioStreamPublisher` 屬性，即 .NET `Func`。開發套件使用 `Func` 定義的 `AudioStreamPublisher` ，從使用者的程式碼提取事件以串流至使用者。軟體開發套件會呼叫 ，`Func`直到傳回 null。

此程式碼示範如何將檔案的音訊串流至 Amazon Transcribe Streaming 進行處理。

```
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**
>
> 有些雙向 HTTP 2 操作，例如來自 Amazon Bedrock 的 `InvokeModelWithBidirectionalStreamAsync`方法，即 Amazon.BedrockRuntime 命名空間，在發佈某些事件之前，不會傳回來自服務用戶端上初始調用的回應。此行為可能會導致您的應用程式遭到封鎖。若要避免這種情況，請區隔向發佈者提供事件的應用程式程式碼，並在不同於在服務用戶端上叫用操作的執行緒上執行。

其他考量

- 適用於 .NET 的 AWS SDK HTTP 2 的支援僅適用於以 .NET 8 及更高版本為目標的版本。它不適用於以 .NET Framework 為目標的版本。
- 如需詳細資訊，請參閱 aws-sdk-net GitHub 儲存庫中的 PR 3730。

其他工具

以下是一些額外的工具，您可以用來簡化開發、部署和維護 .NET 應用程式的工作。

AWS 部署工具

在開發機器上開發雲端原生 .NET Core 應用程式之後，您可以使用適用於 .NET CLI 的 AWS 部署工具，更輕鬆地將應用程式部署到 AWS。

如需詳細資訊，請參閱將應用程式部署到 AWS。

AWS 適用於 .NET 的訊息處理架構

如果您使用的是 Amazon SQS、Amazon SNS 或 Amazon EventBridge 等服務，您可以利用適用於 .NET AWS 的訊息處理架構。如需詳細資訊，請參閱AWS 適用於 .NET 的訊息處理架構。

與 .NET Acover 整合

您可以利用與 .NET Aspire 整合的優勢來改善內部開發迴圈。如需詳細資訊，請參閱在 中 AWS 與 .NET ACache 整合 適用於 .NET 的 AWS SDK。

使用 進行進階身分驗證和授權 適用於 .NET 的 AWS SDK

本節中的主題提供有關 適用於 .NET 的 AWS SDK 應用程式中身分驗證和授權的進階技術資訊。

主題

- 使用 的單一登入 適用於 .NET 的 AWS SDK

使用 的單一登入 適用於 .NET 的 AWS SDK

AWS IAM Identity Center 是一種雲端型單一登入 (SSO) 服務，可讓您輕鬆地集中管理對所有 AWS 帳戶 和雲端應用程式的 SSO 存取。如需完整詳細資訊，請參閱 IAM Identity Center 使用者指南。

如果您不熟悉 SDK 如何與 IAM Identity Center 互動，請參閱下列資訊。

高階互動模式

在高階，SDKs與 IAM Identity Center 的互動方式類似下列模式：

1. 通常透過 IAM Identity Center 主控台設定 IAM Identity Center，並邀請 SSO 使用者參與。
2. 使用者電腦上的共用 AWS `config`檔案會更新為 SSO 資訊。
3. 使用者透過 IAM Identity Center 登入，並取得已為其設定的 AWS Identity and Access Management (IAM) 許可的短期憑證。此登入可以透過非 SDK 工具啟動，例如 AWS CLI，或透過 .NET 應用程式以程式設計方式啟動。
4. 使用者繼續執行其工作。當他們執行使用 SSO 的其他應用程式時，不需要再次登入即可開啟應用程式。

本主題的其餘部分提供設定和使用的參考資訊 AWS IAM Identity Center。它提供補充和比 中基本 SSO 設定更進階的資訊使用 驗證 AWS。如果您是 SSO on 的新手 AWS，建議您先查看該主題以取得基本資訊，然後在下列教學課程中查看 SSO 實際運作：

- 教學課程：僅限 .NET 應用程式
- 教學課程： AWS CLI 和 .NET 應用程式

本主題包含下列章節：

- 先決條件

- 設定 SSO 設定檔
- 產生和使用 SSO 權杖
- 其他資源
- 教學課程

先決條件

使用 IAM Identity Center 之前，您必須執行特定任務，例如選擇身分來源和設定相關 AWS 帳戶 和 應用程式。如需其他資訊，請參閱以下內容：

- 如需這些任務的一般資訊，請參閱《IAM Identity Center 使用者指南》中的入門。
- 如需特定任務範例，請參閱本主題結尾的教學課程清單。不過，請務必先檢閱本主題中的資訊，再嘗試教學課程。

設定 SSO 設定檔

在相關 中設定 IAM Identity Center 之後 AWS 帳戶，必須將 SSO 的具名設定檔新增至使用者的共用 AWS `config`檔案。此設定檔用於連線至AWS 存取入口網站，這會針對已為使用者設定的 IAM 許可傳回短期憑證。

共用`config`檔案通常`%USERPROFILE%\.aws\config`在 Windows 和 Linux 和 macOS `~/.aws/config`中命名。您可以使用您偏好的文字編輯器來新增 SSO 的新設定檔。或者，您可以使用 `aws configure sso`命令。如需此命令的詳細資訊，請參閱AWS Command Line Interface 《 使用者指南》中的設定 AWS CLI 以使用 IAM Identity Center。

新設定檔類似下列內容：

```
[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
```

新設定檔的設定定義如下。前兩個設定定義 AWS 存取入口網站。其他兩個設定是一對組合，可定義已為使用者設定的許可。需要這四個設定。

sso_start_url

指向組織AWS 存取入口網站的 URL。若要尋找此值，請開啟 IAM Identity Center 主控台，選擇設定，然後尋找入口網站 URL。

sso_region

AWS 區域 包含存取入口網站主機的 。這是在您啟用 IAM Identity Center 時選取的區域。它可以不同於您用於其他任務的區域。

如需 AWS 區域 及其代碼的完整清單，請參閱《》中的區域端點Amazon Web Services 一般參考。

sso_account_id

透過 AWS Organizations 服務 AWS 帳戶 新增的 ID。若要查看可用帳戶的清單，請前往 IAM Identity Center 主控台並開啟AWS 帳戶頁面。您為此設定選擇的帳戶 ID 將對應至您計劃提供給`sso_role_name`設定的值，如下所示。

sso_role_name

IAM Identity Center 許可集的名稱。此許可集定義使用者透過 IAM Identity Center 獲得的許可。

下列程序是尋找此設定值的一種方式。

1. 前往 IAM Identity Center 主控台並開啟AWS 帳戶頁面。
2. 選擇帳戶以顯示其詳細資訊。您選擇的帳戶將包含您要為其授予 SSO 許可的 SSO 使用者或群組。
3. 查看指派給帳戶的使用者和群組清單，並尋找感興趣的使用者或群組。您在設定中指定的許可集`sso_role_name`是與此使用者或群組相關聯的其中一個集。

將此值提供給此設定時，請使用許可集的名稱，而不是 Amazon Resource Name (ARN)。

許可集已連接 IAM 政策和自訂許可政策。如需詳細資訊，請參閱《IAM Identity Center 使用者指南》中的許可集。

產生和使用 SSO 權杖

若要使用 SSO，使用者必須先產生臨時字符，然後使用該字符來存取適當的 AWS 應用程式和資源。對於 .NET 應用程式，您可以使用下列方法來產生和使用這些臨時字符：

- 視需要先建立產生字符的 .NET 應用程式，然後使用字符。
- 使用 產生字符， AWS CLI 然後在 .NET 應用程式中使用該字符。

這些方法會在下列各節中說明，並在教學課程中示範。

> **Important**
>
> 您的應用程式必須參考下列 NuGet 套件，以便 SSO 解析能夠運作：
>
> - `AWSSDK.SSO`
> - `AWSSDK.SSOOIDC`
>
> 未參考這些套件將導致執行時間例外狀況。

僅限 .NET 應用程式

本節說明如何建立 .NET 應用程式，在必要時產生臨時 SSO 字符，然後使用該字符。如需此程序的完整教學課程，請參閱 僅使用 .NET 應用程式的 SSO 教學課程。

以程式設計方式產生和使用 SSO 字符

除了使用 之外 AWS CLI，您也可以透過程式設計方式產生 SSO 字符。

若要這樣做，您的應用程式會為 SSO 設定檔建立 `AWSCredentials` 物件，如果有任何可用，則會載入臨時登入資料。然後，您的應用程式必須將`AWSCredentials`物件轉換為`SSOAWSCredentials`物件，並設定一些選項屬性，包括在必要時用來提示使用者登入資訊的回呼方法。

此方法會顯示在下列程式碼片段中。

> **Important**
>
> 您的應用程式必須參考下列 NuGet 套件，以便 SSO 解析能夠運作：
>
> - `AWSSDK.SSO`
> - `AWSSDK.SSOOIDC`
>
> 未參考這些套件將導致執行時間例外狀況。

```
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;
}
```

如果沒有適當的 SSO 字符，則會啟動預設瀏覽器視窗，並開啟適當的登入頁面。例如，如果您使用 IAM Identity Center 做為身分來源，使用者會看到類似以下的登入頁面：

> **Note**
>
> 您提供的文字字串`SSOAWSCredentials.Options.ClientName`不能有空格。如果字串有空格，您會收到執行時間例外狀況。

僅使用 .NET 應用程式的 SSO 教學課程

AWS CLI 和 .NET 應用程式

本節說明如何使用 產生臨時 SSO 字符 AWS CLI，以及如何在應用程式中使用該字符。如需此程序的完整教學課程，請參閱 使用 AWS CLI 和 .NET 應用程式的 SSO 教學課程。

使用 產生 SSO 權杖 AWS CLI

除了以程式設計方式產生臨時 SSO 字符之外，您還可以使用 AWS CLI 來產生字符。下列資訊說明如何進行。

使用者建立已啟用 SSO 的設定檔後，如上一節所示，他們會從 執行 `aws sso login`命令 AWS CLI。它們必須確實包含具有啟用 SSO 設定檔名稱的 `--profile` 參數。下列範例顯示這種情況：

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

如果使用者想要在目前的臨時權杖過期後產生新的臨時權杖，則可以再次執行相同的命令。

在 .NET 應用程式中使用產生的 SSO 字符

以下資訊說明如何使用已產生的臨時字符。

> **Important**
>
> 您的應用程式必須參考下列 NuGet 套件，以便 SSO 解析能夠運作：
>
> - `AWSSDK.SSO`
> - `AWSSDK.SSOOIDC`
>
> 未參考這些套件將導致執行時間例外狀況。

您的應用程式會為 SSO 設定檔建立 `AWSCredentials` 物件，這會載入 先前產生的臨時登入資料 AWS CLI。這類似於 中顯示的方法在應用程式中存取登入資料和設定檔，且格式如下：

```
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;
}
```

`AWSCredentials` 物件接著會傳遞至服務用戶端的建構函式。例如：

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

> **Note**
>
> 如果您的應用程式已建置AWSCredentials為使用 SSO 的[default]設定檔，則不需要使用載入暫時登入資料。在這種情況下，應用程式可以建立沒有參數 AWS 的服務用戶端，類似於「var client = new AmazonS3Client();」。

使用 AWS CLI 和 .NET 應用程式的 SSO 教學課程

其他資源

如需其他說明，請參閱下列資源。

- 什麼是 IAM Identity Center？
- 設定 AWS CLI 以使用 IAM Identity Center
- 在 中使用 IAM Identity Center 登入資料 AWS Toolkit for Visual Studio

教學課程

主題

- 僅使用 .NET 應用程式的 SSO 教學課程
- 使用 AWS CLI 和 .NET 應用程式的 SSO 教學課程

僅使用 .NET 應用程式的 SSO 教學課程

本教學課程說明如何為基本應用程式和測試 SSO 使用者啟用 SSO。它會設定應用程式以程式設計方式產生臨時 SSO 字符，而不是使用 AWS CLI。

本教學課程顯示 中一小部分的 SSO 功能 適用於 .NET 的 AWS SDK。如需搭配 使用 IAM Identity Center 的完整詳細資訊 適用於 .NET 的 AWS SDK，請參閱具有背景資訊的主題。在該主題中，請參閱名為 的子區段中此案例的高階描述僅限 .NET 應用程式。

> **Note**
>
> 本教學中的數個步驟可協助您設定 AWS Organizations 和 IAM Identity Center 等服務。如果您已經執行該組態，或者只對程式碼感興趣，您可以使用範例程式碼跳到 區段。

先決條件

- 如果您尚未設定開發環境，請進行設定。這會在 安裝和設定 的工具鏈 適用於 .NET 的 AWS SDK和等章節中說明開始使用。
- 識別或建立至少一個可用於測試 SSO AWS 帳戶 的 。基於本教學課程的目的，這稱為測試 AWS 帳戶或僅測試帳戶。
- 識別可為您測試 SSO 的 SSO 使用者。這是將使用 SSO 和您建立的基本應用程式的人員。在本教學課程中，該人員可能是您（開發人員）或其他人。我們也建議 SSO 使用者在不在開發環境中的電腦上運作的設定。不過，這並非絕對必要。
- SSO 使用者的電腦必須安裝與您用來設定開發環境的架構相容的 .NET 架構。

設定 AWS

本節說明如何為本教學課程設定各種 AWS 服務。

若要執行此設定，請先以管理員 AWS 帳戶 身分登入測試。然後，執行下列動作：

Amazon S3

前往 Amazon S3 主控台並新增一些無害的儲存貯體。在本教學課程稍後，SSO 使用者將擷取這些儲存貯體的清單。

AWS IAM

前往 IAM 主控台並新增一些 IAM 使用者。如果您授予 IAM 使用者許可，請將許可限制為幾個無害的唯讀許可。在本教學稍後，SSO 使用者將擷取這些 IAM 使用者的清單。

AWS Organizations

前往 AWS Organizations 主控台並啟用 Organizations。如需詳細資訊，請參閱《AWS Organizations 使用者指南》https://docs.aws.amazon.com/organizations/latest/userguide/中的建立組織。

此動作會將測試新增至 AWS 帳戶 組織做為管理帳戶。如果您有其他測試帳戶，您可以邀請他們加入組織，但此教學課程不需要這麼做。

IAM Identity Center

前往 IAM Identity Center 主控台並啟用 SSO。視需要執行電子郵件驗證。如需詳細資訊，請參閱《IAM Identity Center 使用者指南》中的啟用 IAM Identity Center。

然後，執行下列組態。

設定 IAM Identity Center

1. 前往設定頁面。尋找「存取入口網站 URL」，並記錄 值以供稍後在 `sso_start_url`設定中使用。
2. 在 橫幅中 AWS Management Console，尋找在您啟用 SSO AWS 區域 時設定的 。這是 AWS 帳戶 ID 左側的下拉式選單。記錄區域碼以供稍後在 `sso_region`設定中使用。此程式碼將類似於 `us-east-1`。
3. 建立 SSO 使用者，如下所示：
   a. 前往使用者頁面。
   b. 選擇新增使用者，然後輸入使用者的使用者名稱、電子郵件地址、名字和姓氏。然後選擇下一步。
   c. 在 頁面上選擇群組的下一步，然後檢閱資訊，然後選擇新增使用者。
4. 建立群組，如下所示：
   a. 前往群組頁面。
   b. 選擇建立群組，然後輸入群組的群組名稱和描述。
   c. 在新增使用者至群組區段中，選取您先前建立的測試 SSO 使用者。然後，選取建立群組。
5. 建立許可集，如下所示：
   a. 前往許可集頁面，然後選擇建立許可集。
   b. 在許可集類型下，選取自訂許可集，然後選擇下一步。
   c. 開啟內嵌政策並輸入下列政策：

      JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
```

```
            "Effect": "Allow",
            "Action": [
                "s3:ListAllMyBuckets",
                "iam:ListUsers"
            ],
            "Resource": "*"
        }
    ]
}
```

d. 在本教學課程中，輸入 SSOReadOnlyRole做為許可集名稱。如果需要，請新增描述，然後選擇下一步。

e. 檢閱資訊，然後選擇建立。

f. 記錄許可集的名稱，以供稍後在 sso_role_name設定中使用。

6. 前往AWS 帳戶頁面，然後選擇您先前新增至組織的 AWS 帳戶。
7. 在該頁面的概觀區段中，尋找帳戶 ID，並記錄在 sso_account_id設定中以供稍後使用。
8. 選擇使用者和群組索引標籤，然後選擇指派使用者或群組。
9. 在指派使用者和群組頁面上，選擇群組索引標籤，選取您先前建立的群組，然後選擇下一步。
10. 選取您先前建立的許可集，然後選擇下一步，然後選擇提交。組態需要一些時間。

建立範例應用程式

建立下列應用程式。它們將在 SSO 使用者的電腦上執行。

列出 Amazon S3 儲存貯體

除了 AWSSDK.SSO和 之外，AWSSDK.SSOOIDC還包含 NuGet 套件 AWSSDK.S3和 AWSSDK.SecurityToken。

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


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

```
using Amazon.SecurityToken.Model;

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

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

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

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

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

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

            var ssoCredentials = credentials as SSOAWSCredentials;
```

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

除了 AWSSDK.SSO和 之外，AWSSDK.SSOOIDC還包含 NuGet 套件 AWSSDK.IdentityManagement和 AWSSDK.SecurityToken。

```
using System;
```

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


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

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

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

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

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

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

```
        }

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

            var ssoCredentials = credentials as SSOAWSCredentials;

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

```
        }
    }
}
```

除了顯示 Amazon S3 儲存貯體和 IAM 使用者的清單之外，這些應用程式還會顯示已啟用 SSO 之設定檔的使用者身分 ARN，這在本教學`my-sso-profile`課程中。

這些應用程式透過在 SSOAWSCredentials 物件的選項屬性中提供回呼方法來執行 SSO 登入任務。SSOAWSCredentials

指示 SSO 使用者

要求 SSO 使用者檢查其電子郵件並接受 SSO 邀請。系統會提示他們設定密碼。訊息可能需要幾分鐘的時間才能送達 SSO 使用者的收件匣。

將先前建立的應用程式提供給 SSO 使用者。

然後，讓 SSO 使用者執行下列動作：

1. 如果包含共用 AWS `config`檔案的資料夾不存在，請建立它。如果資料夾確實存在且具有名為 的子資料夾`.sso`，請刪除該子資料夾。

   此資料夾的位置通常`%USERPROFILE%\.aws`位於 Windows 和 Linux 和 macOS `~/.aws`中。
2. 視需要在該資料夾中建立共用 AWS `config`檔案，並將設定檔新增至其中，如下所示：

   ```
   [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. 執行 Amazon S3 應用程式。
4. 在產生的 Web 登入頁面中，登入。使用邀請訊息中的使用者名稱，以及為回應訊息而建立的密碼。
5. 登入完成時，應用程式會顯示 S3 儲存貯體清單。
6. 執行 IAM 應用程式。應用程式會顯示 IAM 使用者的清單。即使未執行第二次登入，也是如此。IAM 應用程式使用先前建立的臨時字符。

清除

如果您不想保留在本教學課程中建立的資源，請清除它們。這些可能是開發環境中 AWS 的資源，例如檔案和資料夾。

使用 AWS CLI 和 .NET 應用程式的 SSO 教學課程

本教學課程說明如何為基本 .NET 應用程式和測試 SSO 使用者啟用 SSO。它使用 AWS CLI 來產生臨時 SSO 權杖，而不是以程式設計方式產生權杖。

本教學課程顯示 中一小部分的 SSO 功能 適用於 .NET 的 AWS SDK。如需搭配 使用 IAM Identity Center 的完整詳細資訊 適用於 .NET 的 AWS SDK，請參閱具有背景資訊的主題。在該主題中，請參閱名為 的子區段中此案例的高階描述AWS CLI 和 .NET 應用程式。

> **Note**
>
> 本教學中的數個步驟可協助您設定 AWS Organizations 和 IAM Identity Center 等服務。如果您已經執行這些組態，或者只對程式碼感興趣，您可以使用範例程式碼跳到 區段。

先決條件

- 如果您尚未設定開發環境，請進行設定。這會在 安裝和設定 的工具鏈 適用於 .NET 的 AWS SDK和等章節中說明開始使用。
- 識別或建立至少一個可用於測試 SSO AWS 帳戶 的 。基於本教學課程的目的，這稱為測試 AWS 帳戶或僅測試帳戶。
- 識別可為您測試 SSO 的 SSO 使用者。這是將使用 SSO 和您建立的基本應用程式的人員。在本教學課程中，該人員可能是您（開發人員）或其他人。我們也建議 SSO 使用者在不在開發環境中的電腦上運作的設定。不過，這並非絕對必要。
- SSO 使用者的電腦必須安裝與您用來設定開發環境的架構相容的 .NET 架構。

- 請確定 AWS CLI 版本 2 已安裝在 SSO 使用者的電腦上。您可以在命令提示字元或終端機`aws --version`中執行 來檢查此問題。

設定 AWS

本節說明如何設定本教學課程的各種 AWS 服務。

若要執行此設定，請先以管理員 AWS 帳戶 身分登入測試。然後，執行下列動作：

Amazon S3

前往 Amazon S3 主控台並新增一些無害的儲存貯體。在本教學稍後，SSO 使用者將擷取這些儲存貯體的清單。

AWS IAM

前往 IAM 主控台並新增幾個 IAM 使用者。如果您授予 IAM 使用者許可，請將許可限制為幾個無害的唯讀許可。在本教學課程稍後，SSO 使用者將擷取這些 IAM 使用者的清單。

AWS Organizations

前往 AWS Organizations 主控台並啟用 Organizations。如需詳細資訊，請參閱《AWS Organizations 使用者指南》https://docs.aws.amazon.com/organizations/latest/userguide/中的建立組織。

此動作會將測試新增至 AWS 帳戶 組織做為管理帳戶。如果您有其他測試帳戶，您可以邀請他們加入組織，但此教學課程不需要這麼做。

IAM Identity Center

前往 IAM Identity Center 主控台並啟用 SSO。視需要執行電子郵件驗證。如需詳細資訊，請參閱《IAM Identity Center 使用者指南》中的啟用 IAM Identity Center。

然後，執行下列組態。

設定 IAM Identity Center

1. 前往設定頁面。尋找「存取入口網站 URL」，並記錄 值以供稍後在 `sso_start_url`設定中使用。
2. 在 橫幅中 AWS Management Console，尋找在您啟用 SSO AWS 區域 時設定的 。這是 AWS 帳戶 ID 左側的下拉式選單。記錄區域碼以供稍後在 `sso_region`設定中使用。此程式碼將類似於 `us-east-1`。
3. 建立 SSO 使用者，如下所示：

    a. 前往使用者頁面。

    b. 選擇新增使用者，然後輸入使用者的使用者名稱、電子郵件地址、名字和姓氏。然後選擇下一步。

    c. 在 頁面上選擇群組的下一步，然後檢閱資訊，然後選擇新增使用者。

4. 建立群組，如下所示：

   a. 前往群組頁面。

   b. 選擇建立群組，然後輸入群組的群組名稱和描述。

   c. 在新增使用者至群組區段中，選取您先前建立的測試 SSO 使用者。然後，選取建立群組。

5. 建立許可集，如下所示：

   a. 前往許可集頁面，然後選擇建立許可集。

   b. 在許可集類型下，選取自訂許可集，然後選擇下一步。

   c. 開啟內嵌政策並輸入下列政策：

      JSON

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

   d. 在本教學課程中，輸入 SSOReadOnlyRole做為許可集名稱。如果需要，請新增描述，然後選擇下一步。

   e. 檢閱資訊，然後選擇建立。

   f. 記錄許可集的名稱，以供稍後在 sso_role_name設定中使用。

6. 前往AWS 帳戶頁面，然後選擇您先前新增至組織的 AWS 帳戶。

7. 在該頁面的概觀區段中，尋找帳戶 ID，並記錄在 sso_account_id設定中以供日後使用。

8. 選擇使用者和群組索引標籤，然後選擇指派使用者或群組。

9. 在指派使用者和群組頁面上，選擇群組索引標籤，選取您先前建立的群組，然後選擇下一步。

10. 選取您先前建立的許可集，然後選擇下一步，然後選擇提交。組態需要一些時間。

建立範例應用程式

建立下列應用程式。它們將在 SSO 使用者的電腦上執行。

列出 Amazon S3 儲存貯體

除了 AWSSDK.SSO和 之外，AWSSDK.SSOOIDC還包含 NuGet 套件 AWSSDK.S3和 AWSSDK.SecurityToken。

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

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

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

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

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

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

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

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

列出 IAM 使用者

除了 AWSSDK.SSO和 之外，AWSSDK.SSOOIDC還包含 NuGet 套件 AWSSDK.IdentityManagement和 AWSSDK.SecurityToken。

```
using System;
```

```
using System.Threading.Tasks;

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

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

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

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

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

```
            }
            Console.WriteLine();
        }

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

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

除了顯示 Amazon S3 儲存貯體和 IAM 使用者的清單之外，這些應用程式還會顯示啟用 SSO 之設定檔的使用者身分 ARN，這在本教學`my-sso-profile`課程中。

指示 SSO 使用者

要求 SSO 使用者檢查其電子郵件並接受 SSO 邀請。系統會提示他們設定密碼。訊息可能需要幾分鐘的時間才能送達 SSO 使用者的收件匣。

將先前建立的應用程式提供給 SSO 使用者。

然後，讓 SSO 使用者執行下列動作：

1. 如果包含共用 AWS `config`檔案的資料夾不存在，請建立它。如果資料夾確實存在且具有名為 的子資料夾`.sso`，請刪除該子資料夾。

此資料夾的位置通常%USERPROFILE%\.aws位於 Windows 和 Linux 和 macOS ~/.aws中。

2. 視需要在該資料夾中建立共用 AWS config檔案，並將設定檔新增至其中，如下所示：

```
[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. 執行 Amazon S3 應用程式。出現執行時間例外狀況。
4. 執行下列 AWS CLI 命令：

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

5. 在產生的 Web 登入頁面中，登入。使用邀請訊息中的使用者名稱，以及為回應訊息而建立的密碼。
6. 再次執行 Amazon S3 應用程式。應用程式現在會顯示 S3 儲存貯體的清單。
7. 執行 IAM 應用程式。應用程式會顯示 IAM 使用者的清單。即使未執行第二次登入，也是如此。IAM 應用程式使用先前建立的臨時字符。

清除

如果您不想保留在本教學課程中建立的資源，請清除它們。這些可能是開發環境中 AWS 的資源，例如檔案和資料夾。

將應用程式部署到 AWS

在開發機器上開發雲端原生 .NET Core 應用程式或服務之後，您會想要將其部署到 AWS。您可以使用 AWS Management Console 或 等特定服務來執行此操作 AWS CloudFormation AWS Cloud Development Kit (AWS CDK)。您也可以使用為部署目的而建立 AWS 的工具。透過使用這些工具，您可以執行下列動作。

從 .NET CLI 部署

您可以使用下列適用於 .NET CLI AWS 的工具，將您的應用程式部署到 AWS：

- AWS 部署適用於 .NET CLI 的工具 - 支援部署至 AWS App Runner、Amazon Elastic Container Service (Amazon ECS) 和 AWS Elastic Beanstalk。
- AWS Lambda 適用於 .NET CLI 的工具 - 支援部署 AWS Lambda 專案。

從 IDE 工具組部署

您可以使用 AWS 工具組直接從您選擇的 IDE 部署應用程式：

- AWS Toolkit for Visual Studio

  > **Note**
  >
  > 工具組中的「發佈至 AWS」功能會公開與適用於 .NET CLI 的 AWS 部署工具相同的功能。若要進一步了解，請前往AWS Toolkit for Visual Studio 《 使用者指南》中的發佈至 AWS 。

- AWS Toolkit for JetBrains

  請參閱使用無 AWS 伺服器應用程式和使用 AWS App Runner 。

- AWS 適用於 VS 程式碼的工具組

  請參閱使用無伺服器應用程式和使用 AWS App Runner。

- AWS Toolkit for Azure DevOps

使用案例

下列各節包含特定應用程式類型的使用案例，包括有關如何使用 .NET CLI 部署這些應用程式的資訊。

- ASP.NET Core 應用程式
- .NET 主控台應用程式
- Blazor WebAssembly 應用程式
- AWS Lambda 專案

ASP.NET Core 應用程式

.NET CLI 的AWS 部署工具可協助您部署 ASP.NET 應用程式，並引導您完成部署程序。這是 .NET CLI 的互動式工具，可協助部署最低 AWS 知識的 .NET 應用程式。

部署工具具有下列功能：

- 應用程式的運算建議 - 取得運算建議，並了解哪個 AWS 運算最適合您的應用程式。
- 產生 Dockerfile - 工具會視需要產生 Dockerfile，或使用現有的 Dockerfile。
- 自動封裝和部署 – 此工具會建置部署成品、使用產生的 AWS CDK 部署專案佈建基礎設施，並將您的應用程式部署到所選的 AWS 運算。
- 可重複且可共用的部署 – 您可以產生和修改 AWS CDK 部署專案，以符合您的特定使用案例。您也可以控制專案的版本，並與您的團隊共用，以進行可重複的部署。
- 協助學習 AWS CDK .NET - 此工具可協助您逐步學習其建置的基礎 AWS 工具，例如 AWS CDK。

AWS 部署工具支援將 ASP.NET Core 應用程式部署至下列 AWS 服務：

- 使用 的 Amazon ECS Service AWS Fargate - 支援使用無 AWS Fargate 伺服器運算引擎管理的運算能力，將 Web 應用程式部署至 Amazon Elastic Container Service (Amazon ECS)。
- AWS App Runner - 支援部署至全受管服務，可讓開發人員輕鬆大規模部署容器化 Web 應用程式和 APIs。不需要先前的基礎設施體驗。
- AWS Elastic Beanstalk - 支援部署至 服務，可讓開發人員輕鬆大規模將 Web 應用程式和 APIs 部署至全受管環境。不需要先前的基礎設施體驗。

若要進一步了解，請參閱工具概觀。若要從那裡開始，請導覽至文件、入門，然後選擇如何安裝 以取得安裝指示。

.NET 主控台應用程式

.NET CLI 的AWS 部署工具可協助您將 .NET 主控台應用程式部署為服務或排程任務部署為 Linux 上的容器映像，並引導您完成部署程序。如果您的應用程式沒有 Dockerfile，工具會自動產生它。否則，會使用現有的 Dockerfile。

部署工具具有下列功能：

- 應用程式的運算建議 - 取得運算建議，並了解哪個 AWS 運算最適合您的應用程式。
- 產生 Dockerfile - 工具會視需要產生 Dockerfile，或使用現有的 Dockerfile。
- 自動封裝和部署 – 此工具會建置部署成品、使用產生的 AWS CDK 部署專案佈建基礎設施，並將您的應用程式部署到所選的 AWS 運算。
- 可重複且可共用的部署 – 您可以產生和修改 AWS CDK 部署專案，以符合您的特定使用案例。您也可以控制專案的版本，並與您的團隊共用，以進行可重複的部署。
- 協助學習 AWS CDK .NET - 此工具可協助您逐步學習其建置的基礎 AWS 工具，例如 AWS CDK。

AWS 部署工具支援將 .NET 主控台應用程式部署至下列 AWS 服務：

- 使用 的 Amazon ECS Service AWS Fargate - 支援使用無 AWS Fargate 伺服器運算引擎管理的運算能力，將 .NET 應用程式部署為服務（例如背景處理器）到 Amazon Elastic Container Service (Amazon ECS)。
- 使用 的 Amazon ECS 排程任務 AWS Fargate - 支援使用無 AWS Fargate 伺服器運算引擎管理的運算能力，將 .NET 應用程式部署為排程任務（例如end-of-day程序）到 Amazon ECS。

若要進一步了解，請參閱工具概觀。若要從那裡開始，請導覽至文件、入門，然後選擇如何安裝 以取得安裝指示。

Blazor WebAssembly 應用程式

.NET CLI 的AWS 部署工具可協助您在 Amazon S3 中託管 Blazor WebAssembly 應用程式，並使用 Amazon CloudFront 進行內容網路交付。您的應用程式會部署到 S3 儲存貯體以進行 Web 託管。此工具會建立和設定 S3 儲存貯體，然後將您的 Blazor 應用程式上傳至儲存貯體。

部署工具具有下列功能：

- 自動封裝和部署 – 此工具會建置部署成品、使用產生的 AWS CDK 部署專案佈建基礎設施，並將您的應用程式部署到所選的 AWS 運算。

- 可重複且可共用的部署 – 您可以產生和修改 AWS CDK 部署專案，以符合您的特定使用案例。您也可以控制專案的版本，並與您的團隊共用，以進行可重複的部署。
- 協助學習 AWS CDK .NET - 此工具可協助您逐步學習其建置的基礎 AWS 工具，例如 AWS CDK。

若要進一步了解，請參閱工具概觀。若要從那裡開始，請導覽至文件、入門，然後選擇如何安裝 以取得安裝指示。

AWS Lambda 專案

AWS Lambda 是一種運算服務，可讓您執行程式碼，而無需佈建或管理伺服器。它會在高可用性運算基礎設施上執行程式碼，並執行運算資源的所有管理。如需 Lambda 的詳細資訊，請參閱《 AWS Lambda 開發人員指南》中的什麼是 AWS Lambda？。

您可以使用 .NET 命令列界面 (CLI) 部署 Lambda 函數。

主題

- 先決條件
- 可用的 Lambda 命令
- 部署步驟

先決條件

開始使用 .NET CLI 部署 Lambda 函數之前，您必須先滿足下列先決條件：

- 確認您已安裝 .NET CLI。例如：`dotnet --version`。如有需要，請前往 https://dotnet.microsoft.com/download 進行安裝。
- 設定 .NET CLI 以使用 Lambda。如需如何執行此操作的描述，請參閱《 AWS Lambda 開發人員指南》中的 .NET Core CLI。在該程序中，以下是部署命令：

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

  如果您不確定如何為此練習建立 IAM 角色，請不要包含 `--function-role role`部分。此工具將協助您建立新的角色。

可用的 Lambda 命令

若要列出可透過 .NET CLI 使用的 Lambda 命令，請開啟命令提示字元或終端機，然後輸入 `dotnet lambda --help`。命令輸出將類似如下：

```
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>
```

輸出會列出目前可用的所有命令。

部署步驟

下列指示假設您已建立 AWS Lambda .NET 專案。基於此程序的目的，專案名為 `DotNetCoreLambdaTest`。

1. 開啟命令提示字元或終端機，然後導覽至包含 .NET Lambda 專案檔案的資料夾。
2. 輸入 `dotnet lambda deploy-function`。
3. 如果出現提示，請輸入 AWS 區域 (Lambda 函數將部署到哪個區域）。
4. 出現提示時，輸入要部署的函數名稱，例如 `DotNetCoreLambdaTest`。它可以是已存在於 中的函數名稱， AWS 帳戶 或尚未部署到該函數的名稱。
5. 出現提示時，選取或建立 Lambda 在執行函數時將擔任的 IAM 角色。

成功完成後，即會顯示所建立的 New Lambda 函數訊息。

```
Executing publish command
...
(etc.)
New Lambda function created
```

如果您部署的函數已存在於您的帳戶中，則部署函數只會要求 AWS 區域 （如有必要）。在此情況下，命令輸出會以 結尾`Updating code for existing function`。

部署 Lambda 函數之後，即可開始使用。如需詳細資訊，請參閱如何使用 AWS Lambda 的範例。

Lambda 會自動為您監控 Lambda 函數，並透過 Amazon CloudWatch 報告指標。若要監控 Lambda 函數並進行疑難排解，請參閱監控 Lambda 應用程式並進行疑難排解。

從 呼叫 AWS 服務 適用於 .NET 的 AWS SDK

下列各節包含範例、教學課程、任務和指南，說明如何使用 適用於 .NET 的 AWS SDK 來使用 AWS 服務。這些範例和教學課程依賴 適用於 .NET 的 AWS SDK 提供的 API。若要查看 API 中有哪些類別和方法可用，請參閱 適用於 .NET 的 AWS SDK API 參考。

如果您是初次使用 適用於 .NET 的 AWS SDK，建議您先查看建立簡單的應用程式主題。它為您提供 SDK 的簡介。

您可以在 GitHub 的程式碼範例AWS 儲存庫和 awslabs 儲存庫中找到更多程式碼範例。 https://github.com/awslabs/aws-sdk-net-samples

開始之前，請確定您已設定環境並設定專案。也請檢閱 中的資訊使用開發套件。

主題

- 的引導式程式碼範例 適用於 .NET 的 AWS SDK
- 使用 AWS Lambda 進行運算服務
- 的高階程式庫和架構 適用於 .NET 的 AWS SDK
- 程式設計 AWS OpsWorks 以使用堆疊和應用程式
- 支援其他 AWS 服務和組態

的引導式程式碼範例 適用於 .NET 的 AWS SDK

下列各節包含程式碼範例，並提供範例的指引。他們可以協助您了解如何使用 適用於 .NET 的 AWS SDK 來使用 AWS 服務。如需其他程式碼範例，請參閱 程式碼範例。

> **Note**
>
> 適用於 .NET 的 AWS SDK 也提供 的 V3 專屬程式碼範例。若要尋找它們，請參閱《 適用於 .NET 的 AWS SDK (V3) 開發人員指南》中的具有指引的程式碼範例和程式碼範例。不過，請注意，如果您搭配 SDK 的 V4 （最新版本） 使用 V3-specific程式碼範例，您可能需要根據中的資訊進行調整遷移至第 4 版。

如果您是初次使用 適用於 .NET 的 AWS SDK，建議您先查看建立簡單的應用程式主題。它為您提供 SDK 的簡介。

開始之前，請確定您已設定環境並設定專案。也請檢閱 中的資訊使用開發套件。

主題

- AWS CloudFormation 使用 存取 適用於 .NET 的 AWS SDK
- 使用 Amazon Cognito 驗證使用者
- 使用 Amazon DynamoDB NoSQL 資料庫
- 使用 Amazon EC2
- 使用 存取 AWS Identity and Access Management (IAM) 適用於 .NET 的 AWS SDK
- 使用 Amazon Simple Storage Service 網際網路儲存
- 使用 Amazon Simple Notification Service 從雲端傳送通知
- 使用 Amazon SQS 傳送訊息

AWS CloudFormation 使用 存取 適用於 .NET 的 AWS SDK

適用於 .NET 的 AWS SDK 支援 AWS CloudFormation，可預測並重複建立和佈建 AWS 基礎設施部署。

API

為 AWS CloudFormation 用戶端 適用於 .NET 的 AWS SDK 提供 APIs。APIs 可讓您使用 範本和堆疊等 AWS CloudFormation 功能。本節包含一些範例，顯示您在使用這些 APIs 時可遵循的模式。若要檢視完整的 APIs 集，請參閱 適用於 .NET 的 AWS SDK API 參考 （並捲動至「Amazon.CloudFormation」）。

AWS CloudFormation APIs 由 AWSSDK.CloudFormation 套件提供。

先決條件

開始之前，請確定您已設定環境並設定專案。也請檢閱 中的資訊使用開發套件。

主題

主題

- 使用 列出 AWS 資源 AWS CloudFormation

使用 列出 AWS 資源 AWS CloudFormation

此範例說明如何使用 適用於 .NET 的 AWS SDK 列出 AWS CloudFormation 堆疊中的資源。此範例使用低階 API。應用程式不會使用引數，但只會收集使用者登入資料可存取之所有堆疊的資訊，然後顯示這些堆疊的相關資訊。

開發套件參考

NuGet 套件：

- AWSSDK.CloudFormation

程式設計元素：

- 命名空間 Amazon.CloudFormation

  AmazonCloudFormationClient 類別
- 命名空間 Amazon.CloudFormation.Model

  類別 ICloudFormationPaginatorFactory.DescribeStacks

  類別 DescribeStackResourcesRequest

  類別 DescribeStackResourcesResponse

  類別堆疊

  類別 StackResource

  類別標籤

```
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 驗證使用者

> **Note**
>
> 本主題中的資訊專屬於以 .NET Framework 和 3.3 適用於 .NET 的 AWS SDK 版及更早版本為基礎的專案。

使用 Amazon Cognito Identity，您可以為使用者建立唯一身分，並對其進行驗證，以安全地存取您的 AWS 資源，例如 Amazon S3 或 Amazon DynamoDB。Amazon Cognito Identity 支援公有身分提供者，例如 Amazon、Facebook、Twitter/Digits、Google 或任何 OpenID Connect 相容提供者，以及未驗證的身分。Amazon Cognito 也支援開發人員驗證的身分，可讓您使用自己的後端身分驗證程序註冊和驗證使用者，同時仍使用 Amazon Cognito Sync 同步使用者資料和存取 AWS 資源。

如需 Amazon Cognito 的詳細資訊，請參閱《Amazon Cognito 開發人員指南》。

下列程式碼範例示範如何輕鬆使用 Amazon Cognito Identity。此憑證提供者範例示範如何建立和驗證使用者身分。此CognitoAuthentication 延伸程式庫範例示範如何使用 CognitoAuthentication 延伸程式庫來驗證 Amazon Cognito 使用者集區。

主題

- Amazon Cognito 登入資料提供者
- Amazon CognitoAuthentication 延伸程式庫範例

Amazon Cognito 登入資料提供者

> **Note**
>
> 本主題中的資訊專屬於以 .NET Framework 和 3.3 適用於 .NET 的 AWS SDK 版及更早版本為基礎的專案。

`Amazon.CognitoIdentity.CognitoAWSCredentials`位於 AWSSDK.CognitoIdentity NuGet 套件中的 是登入資料物件，使用 Amazon Cognito 和 AWS Security Token Service (AWS STS) 來擷取登入資料以進行 AWS 呼叫。

設定 `CognitoAWSCredentials` 的第一步是建立「身分集區」。身分集區是您的帳戶專屬的使用者身分資訊存放區。此資訊可跨用戶端平台、裝置和作業系統擷取，因此，如果使用者一開始是在手機上使用您的應用程式，之後再切換到平板電腦，所保存的應用程式資訊仍然可供該使用者使用。您可以從 Amazon Cognito 主控台建立新的身分集區。如果您使用主控台，它還會提供您需要的其他資訊：

- 您的帳號是 12 位數字的號碼，例如 123456789012，是您帳戶專有。
- 未驗證角色 ARN - 未驗證的使用者將擔任的角色。例如，此角色可提供您資料的唯讀許可。
- 已驗證的角色 ARN - 已驗證的使用者將擔任的角色。此角色可提供您資料更多許可。

設定 CognitoAWSCredentials

下列程式碼範例示範如何設定 `CognitoAWSCredentials`，然後您可以使用它以未經驗證的使用者身分呼叫 Amazon S3。這可讓您只使用所需的最少量資料來驗證使用者身分。使用者權限由角色控制，因此您可以依需要設定存取權。

```
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();
}
```

使用 AWS 做為未驗證的使用者

下列程式碼範例示範如何以未經驗證的使用者 AWS 身分開始使用 ，然後透過 Facebook 進行身分驗證，並更新登入資料以使用 Facebook 登入資料。使用此方法，您便可已驗證的角色授與不同功能給已驗證身分的使用者。例如您的手機應用程式允許使用者匿名檢視內容，如果使用一或多個設定的提供者身分登入的話，還可讓他們張貼文章。

```
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();
}
```

如果您使用 AmazonCognitoSyncClient (屬 適用於 .NET 的 AWS SDK的一部分)，CognitoAWSCredentials 物件甚至能提供更多功能。如果您同時使用 AmazonCognitoSyncClient和 CognitoAWSCredentials，則使用 呼叫時不需要指定 IdentityPoolId和 IdentityId 屬性AmazonCognitoSyncClient。這些屬性會自動從 CognitoAWSCredentials 填入。下一個程式碼範例將對此做說明，並且提供一個事件，只要適用於 CognitoAWSCredentials 的 IdentityId 有變動時就會通知您。在某些情況下，例如從未驗證使用者變更為已驗證使用者時，IdentityId 即可變更。

```
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 延伸程式庫範例

> **Note**
>
> 本主題中的資訊專屬於以 .NET Framework 和 3.3 適用於 .NET 的 AWS SDK 版及更早版本為基礎的專案。

CognitoAuthentication 延伸程式庫位於 Amazon.Extensions.CognitoAuthentication NuGet 套件中，可簡化 .NET Core 和 Xamarin 開發人員的 Amazon Cognito 使用者集區的身分驗證程序。程式庫建置在 Amazon Cognito Identity Provider API 之上，以建立和傳送使用者身分驗證 API 呼叫。

使用 CognitoAuthentication 延伸程式庫

Amazon Cognito 有一些標準身分驗證流程的內建 `AuthFlow` 和 `ChallengeName`值，可透過安全遠端密碼 (SRP) 驗證使用者名稱和密碼。如需有關驗證流量的詳細資訊，請參閱 Amazon Cognito 使用者集區身分驗證流程 。

以下範例需要這些 `using` 陳述式：

```
// 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;
```

使用基本身分驗證

使用 AnonymousAWSCredentials 建立 AmazonCognitoIdentityProviderClient，這不需要簽署的請求。您不需要提供一個區域。如果未提供區域，基本程式碼會呼叫 FallbackRegionFactory.GetRegionEndpoint()。建立 CognitoUserPool 和 CognitoUser 物件。以 InitiateSrpAuthRequest 呼叫 StartWithSrpAuthAsync 的方法，其中包含使用者密碼。

```
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;

}
```

利用挑戰進行驗證

繼續挑戰驗證流程，例如使用 NewPasswordRequired 和 Multi-Factor Authentication (MFA)，也更簡易。唯一的需求是 CognitoAuthentication 物件、SRP 使用者的密碼和下次挑戰需要的資訊，需要在使用者輸入後取得提示。以下程式碼會在驗證流程中顯示一個方式來檢查挑戰類型、取得 MFA 的適當回應和 NewPasswordRequired 挑戰。

如同之前提出基本驗證請求，並且 await AuthFlowResponse。當透過傳回的 AuthenticationResult 物件收到循環回應時。如果 ChallengeName 類型為 NEW_PASSWORD_REQUIRED，呼叫 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.");
    }

}
```

身分驗證後使用 AWS 資源

使用 CognitoAuthentication 程式庫驗證使用者後，下一步是允許使用者存取適當的 AWS 資源。若要這樣做，您必須透過 Amazon Cognito 聯合身分主控台建立身分集區。透過使用其 poolID 和 clientID 指定您建立為提供者的 Amazon Cognito 使用者集區，您可以允許 Amazon Cognito 使用者集區使用者存取 AWS 連接到您帳戶的資源。您也可以指定不同的角色，以啟用這兩種未經驗證及經過驗證的使用者存取不同的資源。您可以在 IAM 主控台變更這些規則，其中您可以新增或移除附加政策之角色的 Action (動作) 欄位的許可。然後，使用適當的身分集區、使用者集區和 Amazon Cognito 使用者資訊，您可以呼叫不同的 AWS 資源。下列範例顯示透過 SRP 驗證的使用者，存取關聯身分集區角色允許的不同 Amazon S3 儲存貯體

```
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);
        }
    }
}
```

更多身分驗證選項

除了 SRP、NewPasswordRequired 和 MFA，CognitoAuthentication 擴展程式庫提供更輕鬆的驗證流程：

- 自訂 - 以呼叫 `StartWithCustomAuthAsync(InitiateCustomAuthRequest customRequest)` 起始
- RefreshToken - 以呼叫 `StartWithRefreshTokenAuthAsync(InitiateRefreshTokenAuthRequest refreshTokenRequest)` 起始
- RefreshTokenSRP - 以呼叫 `StartWithRefreshTokenAuthAsync(InitiateRefreshTokenAuthRequest refreshTokenRequest)` 起始
- AdminNoSRP - 以呼叫 `StartWithAdminNoSrpAuthAsync(InitiateAdminNoSrpAuthRequest adminAuthRequest)` 起始

根據您要的流程呼叫適當的方法。然後，當每個方法呼叫的 `AuthFlowResponse` 物件中顯示挑戰時，繼續提示使用者挑戰的存在。同時呼叫適當的回應方法，例如 `RespondToSmsMfaAuthAsync` 用於 MFA 挑戰，`RespondToCustomAuthAsync` 用於自訂挑戰。

使用 Amazon DynamoDB NoSQL 資料庫

> **Note**
>
> 這些主題中的程式設計模型同時存在於 .NET Framework 和 .NET (Core) 中，但呼叫慣例會有所不同，無論是同步或非同步。

適用於 .NET 的 AWS SDK 支援 Amazon DynamoDB，這是 提供的快速 NoSQL 資料庫服務 AWS。開發套件提供三種與 DynamoDB 通訊的程式設計模型：低階模型、文件模型和物件持久性模型。

下列資訊介紹這些模型及其 APIs、提供如何使用和何時使用這些模型的範例，並為您提供 中其他 DynamoDB 程式設計資源的連結 適用於 .NET 的 AWS SDK。

低階模型

低階程式設計模型會包裝對 DynamoDB 服務的直接呼叫。您透過 Amazon.DynamoDBv2 命名空間存取此模型。

三個模型中，低階模型要求您撰寫最多的程式碼。例如，您必須將 .NET 資料類型轉換為 DynamoDB 中的對等資料類型。不過，這個模型可讓您存取最多的功能。

下列範例示範如何使用低階模型來建立資料表、修改資料表，以及將項目插入 DynamoDB 中的資料表。

建立資料表

在下列範例中，您可以使用 `AmazonDynamoDBClient` 類別的 `CreateTable` 方法建立資料表。此 `CreateTable` 方法使用`CreateTableRequest` 類別的執行個體，其中包含特性，例如必要項目屬性名稱、主索引鍵定義和傳輸量。`CreateTable` 方法傳回 `CreateTableResponse` 類別的執行個體。

```
// 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);
}
```

確認資料表已準備修改

在您可以變更或修改資料表之前，資料表必須做好修改的準備。下列範例示範如何使用低階模型來驗證 DynamoDB 中的資料表是否已就緒。在這個範例中，透過 AmazonDynamoDBClient 類別的 DescribeTable方法來做為檢查目標資料表的參考。程式碼每 5 秒檢查一次資料表的 TableStatus 屬性值。當狀態是設定為 ACTIVE，資料表已準備好做修改。

```
// 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);
```

將項目插入到資料表

在下列範例中，您使用低階模型將兩個項目插入 DynamoDB 中的資料表。每個項目使用 PutItemRequest 類別的執行個體，透過 AmazonDynamoDBClient 類別的 PutItem 方法插

入。PutItemRequest 類別的兩個執行個體的每一個，採用具有一系列的項目屬性值，且其項目將插入的資料表名稱。

```
// 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);
```

文件模型

文件程式設計模型提供更簡單的方法來使用 DynamoDB 中的資料。此模型特別適用於存取資料表和資料表中的項目。您透過 Amazon.DynamoDBv2.DocumentModel 命名空間存取此模型。

相較於低階程式設計模型，文件模型較容易針對 DynamoDB 資料編寫程式碼。例如，您不需要在 DynamoDB 中將任意數量的 .NET 資料類型轉換為同等類型。不過，這個模型不提供如同低階程式設計模型數量一樣多功能的存取權。例如，您可以使用此模型來建立、擷取、更新和刪除資料表中的項

目。不過，若要建立資料表，您必須使用低階模型。相較於物件持續性模型，這個模型要求您編寫更多的程式碼來存放、載入和查詢 .NET 物件。

如需 DynamoDB 文件程式設計模型的詳細資訊，請參閱《Amazon DynamoDB 開發人員指南》中的 .NET：文件模型。

下列各節提供如何建立所需 DynamoDB 資料表之表示法的相關資訊，以及如何使用文件模型將項目插入資料表並從資料表取得項目的範例。

建立資料表的表示法

若要使用文件模型執行資料操作，您必須先建立代表特定資料表的 Table類別執行個體。有兩種主要方式可以執行此操作。

LoadTable 方法

第一個機制是使用 Table類別的其中一個靜態LoadTable方法，類似下列範例：

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

> **Note**
>
> 雖然此機制可以運作，但在某些情況下，它有時可能會因為冷啟動和執行緒集區行為而導致額外的延遲或死結。如需這些行為的詳細資訊，請參閱部落格文章改善 的 DynamoDB 初始化模式 適用於 .NET 的 AWS SDK。

TableBuilder

AWSSDK.DynamoDBv2 NuGet 套件的 3.7.203 版中推出了替代機制 TableBuilder類別。此機制可以透過移除特定隱含方法呼叫來解決上述行為；特別是 DescribeTable方法。此機制的使用方式類似下列範例：

```
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();
```

如需此替代機制的詳細資訊，請再次參閱部落格文章改善 的 DynamoDB 初始化模式 適用於 .NET 的 AWS SDK。

將項目插入資料表

在下列範例中，回覆會透過 Table類別的 PutItemAsync方法插入回覆資料表。此 PutItemAsync 方法採用 Document 類別的執行個體； Document 類別只是一組初始化屬性。

```
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);
```

從資料表取得項目

在下列範例中，會透過 Table類別的 GetItemAsync方法擷取回覆。為了判斷要取得的回覆，GetItemAsync方法會使用目標回覆的hash-and-range主索引鍵。

```
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"]);
```

上述範例會隱含地將資料表值轉換為 WriteLine 方法的字串。您可以使用 DynamoDBEntry類別的各種「As【type】」方法進行明確轉換。例如，您可以透過 AsGuid()方法，將 的值Id從Primitive資料類型明確轉換為 GUID：

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

物件持續性模型

物件持久性程式設計模型專為在 DynamoDB 中存放、載入和查詢 .NET 物件而設計。您透過 Amazon.DynamoDBv2.DataModel 命名空間存取此模型。

在三個模型中，每當您儲存、載入或查詢 DynamoDB 資料時，物件持久性模型最容易針對 編寫程式碼。例如，您可以直接使用 DynamoDB 資料類型。不過，此模型只能存取在 DynamoDB 中存放、載入和查詢 .NET 物件的操作。例如，您可以使用此模型來建立、擷取、更新和刪除資料表中的項目。不過，您必須先使用低階模型建立資料表，然後使用此模型將 .NET 類別對應至資料表。

如需 DynamoDB 物件持久性程式設計模型的詳細資訊，請參閱《Amazon DynamoDB 開發人員指南》中的 .NET：物件持久性模型。

下列範例示範如何定義代表 DynamoDB 項目的 .NET 類別、使用 .NET 類別的執行個體將項目插入 DynamoDB 資料表，以及使用 .NET 類別的執行個體從資料表取得項目。

定義代表資料表中項目的 .NET 類別

在下列類別定義範例中， DynamoDBTable 屬性會指定資料表名稱，而 DynamoDBHashKey和 DynamoDBRangeKey 屬性則會建立資料表hash-and-range主索引鍵的模型。DynamoDBGlobalSecondaryIndexHashKey 屬性已定義，因此可以建構特定作者的回覆查詢。

```
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; }
}
```

建立物件持久性模型的內容

若要使用 DynamoDB 的物件持久性程式設計模型，您必須建立內容，以提供與 DynamoDB 的連線，並可讓您存取資料表、執行各種操作和執行查詢。

基本內容

下列範例示範如何建立最基本的內容。

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

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

DisableFetchingTableMetadata 屬性的內容

下列範例示範如何額外設定 DynamoDBContextConfig類別的 DisableFetchingTableMetadata 屬性，以防止對 DescribeTable方法的隱含呼叫。

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

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

如果 DisableFetchingTableMetadata 屬性設為 false（預設值），如第一個範例所示，您可以省略描述Reply類別中資料表項目索引鍵和索引結構的屬性。這些屬性將改為透過對 DescribeTable方法的隱含呼叫來推斷。如果 DisableFetchingTableMetadata 設定為 true，

如第二個範例所示，物件持久性模型的方法，例如 SaveAsync和 完全QueryAsync依賴 Reply類別中定義的屬性。在此情況下，不會呼叫 DescribeTable方法。

> **Note**
>
> 在某些情況下，對 DescribeTable方法的呼叫有時可能會因為冷啟動和執行緒集區行為而導致額外的延遲或死結。因此，有時最好避免呼叫該方法。
> 如需這些行為的詳細資訊，請參閱部落格文章改善 的 DynamoDB 初始化模式 適用於 .NET 的 AWS SDK。

使用 .NET 類別的執行個體將項目插入資料表

在此範例中，會透過 DynamoDBContext類別的 SaveAsync方法插入項目，這會取得代表項目之 .NET 類別的初始化執行個體。

```
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 類別的執行個體從資料表取得項目

在此範例中，會建立查詢，使用 DynamoDBContext類別的 QueryAsync方法尋找「Author1」的所有記錄。然後，會透過查詢的 GetNextSetAsync方法擷取項目。

```
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);
}
```

物件持久性模型的其他資訊

上述範例和說明有時會包含名為 之DynamoDBContext類別的屬性DisableFetchingTableMetadata。此屬性在 AWSSDK.DynamoDBv2 NuGet 套件的 3.7.203 版中推出，可讓您避免因冷啟動和執行緒集區行為而可能導致額外延遲或死結的特定情況。如需詳細資訊，請參閱部落格文章改善 的 DynamoDB 初始化模式 適用於 .NET 的 AWS SDK。

以下是有關此屬性的一些其他資訊。

- 如果您使用 .NET Framework，則可以在 app.config或 web.config 檔案中全域設定此屬性。
- 您可以使用 AWSConfigsDynamoDB類別全域設定此屬性，如下列範例所示。

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

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

- 在某些情況下，您無法將 DynamoDB 屬性新增至 .NET 類別；例如，如果類別是在相依性中定義。在這種情況下，仍然可以利用 DisableFetchingTableMetadata 屬性。若要這樣做，請在 DisableFetchingTableMetadata 屬性之外使用 TableBuilder類別。此TableBuilder類別也在 AWSSDK.DynamoDBv2 NuGet 套件的 3.7.203 版中推出。

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

其他資訊

使用 適用於 .NET 的 AWS SDK 編寫 DynamoDB 程式、資訊和範例

- DynamoDB API
- DynamoDB 系列開始
- DynamoDB 系列 - 文件模型
- DynamoDB 系列 - 轉換結構描述
- DynamoDB 系列 - 物件持久性模型

- DynamoDB 系列 - 運算式
- 搭配 Amazon DynamoDB 和 使用運算式 適用於 .NET 的 AWS SDK
- Amazon DynamoDB 中的 JSON 支援

低階模型、資訊和範例

- 使用 適用於 .NET 的 AWS SDK 低階 API 處理資料表
- 使用 適用於 .NET 的 AWS SDK 低階 API 處理項目
- 使用 適用於 .NET 的 AWS SDK 低階 API 查詢資料表
- 使用 適用於 .NET 的 AWS SDK 低階 API 掃描資料表
- 使用 適用於 .NET 的 AWS SDK 低階 API 處理本機次要索引
- 使用 適用於 .NET 的 AWS SDK 低階 API 處理全域次要索引

文件模型、資訊和範例

- DynamoDB 資料類型
- DynamoDBEntry
- .NET：文件模型

物件持久性模型、資訊和範例

- .NET：物件持久性模型

其他實用資訊

- AWS 與 .NET Acover 整合 如需使用 Amazon DynamoDB 進行本機開發的相關資訊，請參閱 。

主題

- 搭配 Amazon DynamoDB 和 使用運算式 適用於 .NET 的 AWS SDK
- Amazon DynamoDB 中的 JSON 支援

搭配 Amazon DynamoDB 和 使用運算式 適用於 .NET 的 AWS SDK

> **Note**
>
> 本主題中的資訊專屬於以 .NET Framework 和 3.3 適用於 .NET 的 AWS SDK 版及更早版本為基礎的專案。

下列程式碼範例示範如何使用 適用於 .NET 的 AWS SDK 搭配表達式來程式設計 DynamoDB。運算式表示您要從 DynamoDB 資料表中的項目讀取的屬性。您也可以在寫入項目時使用表達式來表示必須符合的任何條件 (也稱為條件更新) 及屬性的更新方式。有些更新範例是以新值取代屬性，或新增資料到清單或對應表。如需詳細資訊，請參閱使用運算式讀取和寫入項目。

主題

- 範例資料
- 使用表達式和項目的主索引鍵取得單一項目
- 使用表達式和資料表的主索引鍵取得多重項目
- 使用運算式和其他項目屬性來取得多重項目
- 列印項目
- 使用運算式建立或取代項目
- 使用運算式更新項目
- 使用運算式刪除項目
- 詳細資訊

範例資料

本主題中的程式碼範例依賴名為 的 DynamoDB 資料表中的下列兩個範例項目`ProductCatalog`。這些項目描述有關產品項目存放在虛構自行車目錄的資訊。這些項目中提供的範例是根據 案例研究：ProductCatalog 項目。資料類型描述項如 `BOOL`、`L`、`M`、`N`、`NS`、`S` 和 `SS`，對應到 JSON 資料格式中的描述項。

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

使用表達式和項目的主索引鍵取得單一項目

以下範例功能 Amazon.DynamoDBv2.AmazonDynamoDBClient.GetItem 方法和一組可取得並列印具有 Id 的 205 項目的運算式。只有下列項目的屬性會傳回：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);
```

在上述範例中，ProjectionExpression 屬性指定要傳回的屬性。此 ExpressionAttributeNames 屬性指定預留位置 #pr 代表 ProductReviews 屬性，而預留位置 #ri 代表 RelatedItems 屬性。呼叫 PrintItem 是指在列印項目中描述的自訂功能。

使用表達式和資料表的主索引鍵取得多重項目

以下範例具備 Amazon.DynamoDBv2.AmazonDynamoDBClient.Query 方法和一組可取得的運算式，然後列印具有 Id 301 的項目，但只限於 Price 值大於 150 時。只有下列項目的屬性會傳回：Id、Title 和所有在 ProductReviews 的 ThreeStar 屬性。

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

在上述範例中，ProjectionExpression 屬性指定要傳回的屬性。此 ExpressionAttributeNames 屬性指定預留位置 #pr 代表 ProductReviews 屬性，而預留位置 #p 代表 Price 屬性。#pr.ThreeStar 指定只傳回 ThreeStar 屬性。ExpressionAttributeValues 屬性指定預留位置 :val 代表值 150。FilterExpression 屬性指定 #p (Price) 必須大於 :val (150)。呼叫 PrintItem 是指在列印項目中描述的自訂功能。

使用運算式和其他項目屬性來取得多重項目

以下範例具備 Amazon.DynamoDBv2.AmazonDynamoDBClient.Scan 方法和一組可取得的運算式，然後列印所有具有 Bike 的 ProductCategory 項目。只有下列項目的屬性會傳回：Id、Title 和所有在 ProductReviews 的屬性。

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

在上述範例中，ProjectionExpression 屬性指定要傳回的屬性。此 ExpressionAttributeNames 屬性指定預留位置 #pr 代表 ProductReviews 屬性，而預留位置 #pc 代表 ProductCategory 屬性。ExpressionAttributeValues 屬性指定預留位置 :catg 代表值 Bike。FilterExpression 屬性指定 #pc (ProductCategory) 必須等於 :catg (Bike)。呼叫 PrintItem 是指在列印項目中描述的自訂功能。

列印項目

以下範例說明如何列印某個項目的屬性和值。此範例用於上述範例，說明如何使用表達式和項目的主索引鍵取得單一項目、使用表達式和資料表的主索引鍵取得多重項目，以及使用表達式和其他項目屬性取得多重項目。

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

在上述範例中，每個屬性值有數種資料類型特定屬性，可進行評估以判斷列印屬性所用的正確格式。這些屬性包括 B、BOOL、BS、L、M、N、NS、NULL、S 和 SS，這些分別對應於那些JSON 資料格式。對於像 B、N、NULL 和 S 的屬性，如果對應的屬性不是 null，則屬性是對應的非 null 資料類型。對於，例如BS、L、M、NS 和 SS 屬性，如果 Count 大於零，則屬性是對應的非零值資料類型。如果所有屬性的資料類型特定屬性是 null 或 Count 等於零，則屬性對應到 BOOL 資料類型。

使用運算式建立或取代項目

以下範例具備 Amazon.DynamoDBv2.AmazonDynamoDBClient.PutItem 方法和一組可取得的運算式，可更新具有 18-Bicycle 301 的 Title 的項目。如果項目尚未存在，則會新增新的項目。

```
// 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);
```

在上述範例中，ExpressionAttributeNames 屬性指定預留位置 #title 代表 Title 屬性。ExpressionAttributeValues 屬性指定預留位置 :product 代表值 18-Bicycle 301。ConditionExpression 屬性指定 #title (Title) 必須等於 :product (18-Bicycle 301)。此呼叫 CreateItemData 是指以下自訂函數：

```
// 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;
}
```

在上述範例中，具範例資料的範例項目會傳回給呼叫者。建構一系列屬性和對應的值，使用資料類型例如 BOOL、L、M、N、NS、S、和SS、分別對應於那些 JSON 資料格式。

使用運算式更新項目

以下範例具備 Amazon.DynamoDBv2.AmazonDynamoDBClient.UpdateItem 方法和一組運算式，為具有 Id 301 的項目變更 Title 到 18" Girl's Bike。

```
// 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);
```

在上述範例中，ExpressionAttributeNames 屬性指定預留位置 #title 代表 Title 屬性。ExpressionAttributeValues 屬性指定預留位置 :newproduct 代表值 18" Girl's Bike。UpdateExpression 屬性指定變更 #title (Title) 為 :newproduct (18" Girl's Bike)。

使用運算式刪除項目

以下範例具備 Amazon.DynamoDBv2.AmazonDynamoDBClient.DeleteItem 方法和一組表達式，可刪除具有 Id 301 的項目，但只有項目的 Title 為 18-Bicycle 301。

```
// using Amazon.DynamoDBv2;
// using Amazon.DynamoDBv2.Model;

var client = new AmazonDynamoDBClient();
var request = new DeleteItemRequest
{
  TableName = "ProductCatalog",
  Key = new Dictionary<string,AttributeValue>
  {
     { "Id", new AttributeValue { N = "301" } }
  },
  ExpressionAttributeNames = new Dictionary<string, string>
  {
    { "#title", "Title" }
  },
  ExpressionAttributeValues = new Dictionary<string, AttributeValue>
  {
    { ":product", new AttributeValue { S = "18-Bicycle 301" } }
  },
  ConditionExpression = "#title = :product"
};
```

```
client.DeleteItem(request);
```

在上述範例中，ExpressionAttributeNames 屬性指定預留位置 #title 代表 Title 屬性。ExpressionAttributeValues 屬性指定預留位置 :product 代表值 18-Bicycle 301。ConditionExpression 屬性指定 #title (Title) 必須等於 :product (18-Bicycle 301)。

詳細資訊

如需詳細資訊和編碼範例，請參閱:

- DynamoDB 系列 - 運算式
- 存取項目的投影運算式屬性
- 使用預留位置的屬性名稱和值
- 使用條件運算式指定條件
- 以更新運算式修改項目和屬性
- 使用 適用於 .NET 的 AWS SDK 低階 API 處理項目
- 使用 適用於 .NET 的 AWS SDK 低階 API 查詢資料表
- 使用 適用於 .NET 的 AWS SDK 低階 API 掃描資料表
- 使用 適用於 .NET 的 AWS SDK 低階 API 處理本機次要索引
- 使用 適用於 .NET 的 AWS SDK 低階 API 處理全域次要索引

Amazon DynamoDB 中的 JSON 支援

> **Note**
>
> 本主題中的資訊專屬於以 .NET Framework 和 3.3 適用於 .NET 的 AWS SDK 版及更早版本為基礎的專案。

使用 Amazon DynamoDB 時， 適用於 .NET 的 AWS SDK 支援 JSON 資料。這可讓您更輕鬆地從 DynamoDB 資料表取得 JSON 格式的資料，並將 JSON 文件插入其中。

主題

- 以 JSON 格式從 DynamoDB 資料表取得資料

- 將 JSON 格式資料插入 DynamoDB 資料表
- DynamoDB 資料類型轉換為 JSON
- 詳細資訊

以 JSON 格式從 DynamoDB 資料表取得資料

下列範例顯示如何從 DynamoDB 資料表取得 JSON 格式的資料：

```
// 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
//   }
```

在上述範例中，Document 類別的 ToJson 方法將某個項目從資料表轉換到 JSON 格式字串。該項目透過 Table 類別的 GetItem 方法擷取。在此範例中，為判斷要取得的項目，GetItem 方法使用目標項目的雜湊和範圍主索引鍵。為了決定要從中取得項目的資料表， Table 類別的 LoadTable方法會使用 DynamoDB 中 AmazonDynamoDBClient類別的執行個體和目標資料表的名稱。

將 JSON 格式資料插入 DynamoDB 資料表

下列範例示範如何使用 JSON 格式將項目插入 DynamoDB 資料表：

```
// 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);
```

在上述範例中，Document 類別的 FromJson 方法將 JSON 格式字串轉換到項目。項目是透過 Table 類別的 PutItem 方法插入到資料表中，其使用包含項目的 Document 類別的執行個體。若要判斷要插入項目的資料表，會呼叫 Table類別的 LoadTable 方法，在 DynamoDB 中指定 AmazonDynamoDBClient類別的執行個體和目標資料表的名稱。

DynamoDB 資料類型轉換為 JSON

每當您呼叫 Document類別的 ToJson方法，然後在產生的 JSON 資料上呼叫 FromJson方法，將 JSON 資料轉換回Document類別的執行個體時，某些 DynamoDB 資料類型將不會如預期轉換。具體而言：

- DynamoDB 集合 (SS、 NS和 BS類型 ） 將轉換為 JSON 陣列。
- DynamoDB 二進位純量和集合 (B和 BS類型 ） 將轉換為 base64 編碼的 JSON 字串或字串清單。

  在此案例中，您必須呼叫的 Document 類別的 DecodeBase64Attributes 方法，使用正確的二進位代碼取代以 base64 編碼的 JSON 資料。在以下範例中，在名為 Picture 之 Document 類別的執行個體中，使用正確的二進位代碼取代以 base64 編碼的二進位純量項目屬性。此範例也在名為 RelatedPictures 之 Document 類別的相同執行個體中，針對以 base64 編碼的二進位集項目屬性進行目同操作：

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

詳細資訊

如需使用 DynamoDB 搭配 程式設計 JSON 的詳細資訊和範例 適用於 .NET 的 AWS SDK，請參閱：

- DynamoDB JSON 支援
- Amazon DynamoDB 更新 - JSON、擴充免費方案、靈活擴展、較大項目

使用 Amazon EC2

適用於 .NET 的 AWS SDK 支援 Amazon EC2，這是一種 Web 服務，可提供可調整大小的運算容量。您可以使用此運算容量來建置和託管軟體系統。

API

為 Amazon EC2 用戶端 適用於 .NET 的 AWS SDK 提供 APIs。APIs 可讓您使用安全群組和金鑰對等 EC2 功能。APIs 也可讓您控制 Amazon EC2 執行個體。本節包含一些範例，顯示您在使用這些 APIs 時可遵循的模式。若要檢視完整的 APIs 集，請參閱 適用於 .NET 的 AWS SDK API 參考 （並捲動至「Amazon.EC2」）。

Amazon EC2 APIs 由 AWSSDK.EC2 NuGet 套件提供。

先決條件

開始之前，請確定您已設定環境並設定專案。也請檢閱 中的資訊使用開發套件。

關於範例

本節中的範例說明如何使用 Amazon EC2 用戶端和管理 Amazon EC2 執行個體。

EC2 Spot 執行個體教學課程說明如何請求 Amazon EC2 Spot 執行個體。Spot 執行個體可讓您以低於隨需價格的價格存取未使用的 EC2 容量。

主題

- 在 Amazon EC2 中使用安全群組
- 使用 Amazon EC2 金鑰對
- 查看您的 Amazon EC2 區域和可用區域
- 使用 Amazon EC2 執行個體
- Amazon EC2 Spot 執行個體教學課程

在 Amazon EC2 中使用安全群組

在 Amazon EC2 中，安全群組可做為虛擬防火牆，控制一或多個 EC2 執行個體的網路流量。根據預設，EC2 會將您的執行個體與不允許傳入流量的安全群組建立關聯。您可以建立允許您的 EC2 執行個體接受特定連接的安全群組。例如，如果您需要連接到 EC2 Windows 執行個體，則必須設定安全群組以允許 RDP 流量。

若要進一步了解安全群組，請參閱《Amazon EC2 使用者指南》中的 Amazon EC2 安全群組。

Amazon EC2

> **Warning**
>
> EC2-Classic 在 2022 年 8 月 15 日淘汰。建議您從 EC2-Classic 遷移至 VPC。如需詳細資訊，請參閱部落格文章 EC2-Classic Networking 正在淘汰 – 以下是如何準備。

如需 APIs和先決條件的相關資訊，請參閱父區段 (使用 Amazon EC2)。

主題

- 列舉安全群組
- 建立安全群組
- 更新安全群組

列舉安全群組

此範例說明如何使用 適用於 .NET 的 AWS SDK 列舉安全群組。如果您提供 Amazon Virtual Private Cloud ID，應用程式會列舉該特定 VPC 的安全群組。否則，應用程式只會顯示所有可用安全群組的清單。

下列各節提供此範例的程式碼片段。之後會顯示範例的完整程式碼，並可依原樣建置和執行。

主題

- 列舉安全群組
- 完成程式碼
- 其他考量

列舉安全群組

下列程式碼片段會列舉您的安全群組。如果指定一個群組，則會列舉特定 VPC 的所有群組或群組。

本主題結尾的範例顯示此程式碼片段正在使用中。

```
    //
    // 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();
      }
    }
```

完成程式碼

本節顯示此範例的相關參考和完整程式碼。

開發套件參考

NuGet 套件：

- AWSSDK.EC2

程式設計元素：

- 命名空間 Amazon.EC2

  AmazonEC2Client 類別
- 命名空間 Amazon.EC2.Model

  類別 DescribeSecurityGroupsRequest

  類別 DescribeSecurityGroupsResponse

  類別篩選條件

  類別 SecurityGroup

程式碼

```
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();
        }
    }
  }
}
```

其他考量

- 請注意，VPC 案例的篩選條件建構時，名稱/值對Name的部分設定為 "vpc-id"。此名稱來自 DescribeSecurityGroupsRequest 類別Filters屬性的描述。
- 若要取得安全群組的完整清單，您也可以在沒有參數的情況下使用 DescribeSecurityGroupsAsync。
- 您可以在 Amazon EC2 主控台中檢查安全群組清單來驗證結果。

建立安全群組

此範例說明如何使用 適用於 .NET 的 AWS SDK 來建立安全群組。您可以提供現有 VPC 的 ID，以在 VPC 中建立 EC2 的安全群組。如果您未提供此類 ID，則如果 AWS 您的帳戶支援此 ID，新的安全群組將適用於 EC2-Classic。

如果您未提供 VPC ID，且 AWS 您的帳戶不支援 EC2-Classic，則新的安全群組將屬於您帳戶的預設 VPC。

> **Warning**
>
> EC2-Classic 在 2022 年 8 月 15 日淘汰。建議您從 EC2-Classic 遷移至 VPC。如需詳細資訊，請參閱部落格文章 EC2-Classic Networking 正在淘汰 – 以下是如何準備。

下列各節提供此範例的程式碼片段。之後會顯示範例的完整程式碼，並可依原樣建置和執行。

主題

- 尋找現有的安全群組
- 建立安全群組

- 完成程式碼

尋找現有的安全群組

下列程式碼片段會搜尋指定 VPC 中具有指定名稱的現有安全群組。

本主題結尾的範例顯示此程式碼片段的使用中。

```
//
// 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;
}
```

建立安全群組

如果指定 VPC 中不存在具有該名稱的群組，以下程式碼片段會建立新的安全群組。如果未指定 VPC，且具有該名稱的一或多個群組存在，則程式碼片段只會傳回群組清單。

本主題結尾的範例顯示此程式碼片段正在使用中。

```
//
// 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;
  }
```

完成程式碼

本節顯示此範例的相關參考和完整程式碼。

開發套件參考

NuGet 套件：

- AWSSDK.EC2

程式設計元素：

- 命名空間 Amazon.EC2

  AmazonEC2Client 類別
- 命名空間 Amazon.EC2.Model

  類別 CreateSecurityGroupRequest

  類別 CreateSecurityGroupResponse

  類別 DescribeSecurityGroupsRequest

  類別 DescribeSecurityGroupsResponse

  類別篩選條件

  類別 SecurityGroup

程式碼

```
using System;
using System.Threading.Tasks;
using System.Collections.Generic;
using Amazon.EC2;
using Amazon.EC2.Model;

namespace EC2CreateSecGroup
{
  // = = =
```

```
        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);
    }
  }

}
```

更新安全群組

此範例說明如何使用 適用於 .NET 的 AWS SDK 將規則新增至安全群組。特別是，此範例會新增規則，以允許特定 TCP 連接埠上的傳入流量，例如用於 EC2 執行個體的遠端連線。應用程式會取得現有安全群組的 ID、CIDR 格式的 IP 地址（或地址範圍），以及選用的 TCP 連接埠號碼。然後，它會將傳入規則新增至指定的安全群組。

> **Note**
>
> 若要使用此範例，您需要 CIDR 格式的 IP 地址（或地址範圍）。如需取得本機電腦 IP 地址的方法，請參閱本主題結尾的其他考量事項。

下列各節提供此範例的程式碼片段。之後會顯示範例的完整程式碼，並可依原樣建置和執行。

主題

- 新增傳入規則
- 完成程式碼
- 其他考量

新增傳入規則

下列程式碼片段會將傳入規則新增至特定 IP 地址（或範圍）和 TCP 連接埠的安全群組。

本主題結尾的範例顯示此程式碼片段正在使用中。

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

完成程式碼

本節顯示此範例的相關參考和完整程式碼。

開發套件參考

NuGet 套件：

- AWSSDK.EC2

程式設計元素：

- 命名空間 Amazon.EC2

  AmazonEC2Client 類別
- 命名空間 Amazon.EC2.Model

類別 AuthorizeSecurityGroupIngressRequest

Class AuthorizeSecurityGroupIngressResponse

類別 IpPermission

類別 IpRange

程式碼

```
using System;
using System.Threading.Tasks;
using System.Collections.Generic;
using Amazon.EC2;
using Amazon.EC2.Model;

namespace EC2AddRuleForRDP
{
  // = = =
```

```
      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.");
    }
  }


  // = = =
```

```
        // 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);
    }
  }

}
```

其他考量

- 如果您未提供連接埠號碼，應用程式會預設為連接埠 3389。這是 Windows RDP 的連接埠，可讓您連線至執行 Windows 的 EC2 執行個體。如果您要啟動執行 Linux 的 EC2 執行個體，您可以改用 TCP 連接埠 22 (SSH)。

- 請注意，範例`IpProtocol`設定為 "tcp"。您可以在 IpPermission 類別`IpProtocol`屬性的描述`IpProtocol`中找到 的值。

- 當您使用此範例時，您可能想要本機電腦的 IP 地址。以下是您可以取得地址的一些方式。
  - 如果您的本機電腦（您將從中連線到 EC2 執行個體）具有靜態公有 IP 地址，您可以使用 服務來取得該地址。這類服務之一是 https：//http://checkip.amazonaws.com/。若要進一步了解授權傳入流量，請參閱《Amazon EC2 使用者指南》中的將規則新增至安全群組和不同使用案例的安全群組規則。
  - 取得本機電腦 IP 地址的另一種方法是使用 Amazon EC2 主控台。

    選取其中一個安全群組，選取傳入規則索引標籤，然後選擇編輯傳入規則。在傳入規則中，開啟來源欄中的下拉式功能表，然後選擇我的 IP，以 CIDR 格式查看本機電腦的 IP 地址。請務必取消操作。

- 您可以在 Amazon EC2 主控台中檢查安全群組清單，以驗證此範例的結果。

使用 Amazon EC2 金鑰對

Amazon EC2 使用公有金鑰加密法將登入資訊進行加密及解密。公有金鑰加密會使用公有金鑰來加密資料，然後收件人會使用私有金鑰來解密資料。公有金鑰和私有金鑰稱為金鑰對。如果您想要登入 EC2 執行個體，您必須在啟動時指定金鑰對，然後在連線時提供該對的私有金鑰。

當您啟動 EC2 執行個體時，您可以為其建立金鑰對，或使用您在啟動其他執行個體時已使用的金鑰對。若要進一步了解 Amazon EC2 金鑰對，請參閱《Amazon EC2 使用者指南》中的使用 Amazon EC2 金鑰對。 Amazon EC2

如需 APIs和先決條件的相關資訊，請參閱父區段 (使用 Amazon EC2)。

主題

- 建立和顯示金鑰對
- 刪除金鑰對

建立和顯示金鑰對

此範例說明如何使用 適用於 .NET 的 AWS SDK 來建立金鑰對。應用程式會取得新金鑰對的名稱和 PEM 檔案的名稱 （ 副檔名為 ".pem")。它會建立金鑰對，將私有金鑰寫入 PEM 檔案，然後顯示所有可用的金鑰對。如果您不提供命令列引數，應用程式只會顯示所有可用的金鑰對。

下列各節提供此範例的程式碼片段。之後會顯示範例的完整程式碼，並可依原樣建置和執行。

主題

- 建立金鑰對
- 顯示可用的金鑰對
- 完成程式碼
- 其他考量

建立金鑰對

下列程式碼片段會建立金鑰對，然後將私有金鑰儲存到指定的 PEM 檔案。

本主題結尾的範例顯示此程式碼片段正在使用中。

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

顯示可用的金鑰對

下列程式碼片段會顯示可用金鑰對的清單。

本主題結尾的範例顯示此程式碼片段的使用中。

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

完成程式碼

本節顯示此範例的相關參考和完整程式碼。

開發套件參考

NuGet 套件：

- AWSSDK.EC2

程式設計元素：

- 命名空間 Amazon.EC2

  AmazonEC2Client 類別
- 命名空間 Amazon.EC2.Model

  類別 CreateKeyPairRequest

  類別 CreateKeyPairResponse

  類別 DescribeKeyPairsResponse

  類別 KeyPairInfo

程式碼

```
using System;
using System.Threading.Tasks;
using System.IO;
using Amazon.EC2;
using Amazon.EC2.Model;
using System.Collections.Generic;

namespace EC2CreateKeyPair
{
  // = = =
```

```
        "\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.");
    }
 }


 // = = =
```

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

```
}
```

其他考量

- 執行範例後，您可以在 Amazon EC2 主控台中看到新的金鑰對。
- 建立金鑰對時，您必須儲存傳回的私有金鑰，因為您稍後無法擷取私有金鑰。

刪除金鑰對

此範例說明如何使用 適用於 .NET 的 AWS SDK 刪除金鑰對。應用程式會取得金鑰對的名稱。它會刪除金鑰對，然後顯示所有可用的金鑰對。如果您不提供命令列引數，應用程式只會顯示所有可用的金鑰對。

下列各節提供此範例的程式碼片段。之後會顯示範例的完整程式碼，並可依原樣建置和執行。

主題

- 刪除金鑰對
- 顯示可用的金鑰對
- 完成程式碼

刪除金鑰對

下列程式碼片段會刪除金鑰對。

本主題結尾的範例顯示此程式碼片段正在使用中。

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

顯示可用的金鑰對

下列程式碼片段會顯示可用金鑰對的清單。

本主題結尾的範例顯示此程式碼片段正在使用中。

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

完成程式碼

本節顯示此範例的相關參考和完整程式碼。

開發套件參考

NuGet 套件：

- AWSSDK.EC2

程式設計元素：

- 命名空間 Amazon.EC2

  AmazonEC2Client 類別
- 命名空間 Amazon.EC2.Model

  類別 DeleteKeyPairRequest

  類別 DescribeKeyPairsResponse

  類別 KeyPairInfo

程式碼

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

查看您的 Amazon EC2 區域和可用區域

Amazon EC2 託管於全球多個位置。這些地點是由 區域及可用區域組成。每個區域都是獨立的地理區域，具有多個隔離的位置，稱為可用區域。

若要進一步了解區域和可用區域，請參閱《Amazon EC2 使用者指南》中的區域和區域。

此範例說明如何使用 適用於 .NET 的 AWS SDK 來取得與 EC2 用戶端相關的區域和可用區域的詳細資訊。應用程式會顯示 EC2 用戶端可用的區域和可用區域的清單。

開發套件參考

NuGet 套件：

- AWSSDK.EC2

程式設計元素：

- 命名空間 Amazon.EC2

  AmazonEC2Client 類別
- 命名空間 Amazon.EC2.Model

  類別 DescribeAvailabilityZonesResponse

  類別 DescribeRegionsResponse

  類別AvailabilityZone

  類別區域

```
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 執行個體

您可以使用 透過建立、啟動和終止等操作 適用於 .NET 的 AWS SDK 來控制 Amazon EC2 執行個體。本節中的主題提供一些如何執行此操作的範例。若要進一步了解 EC2 執行個體，請參閱《Amazon EC2 使用者指南》中的 Amazon EC2 執行個體。 Amazon EC2

如需 APIs和先決條件的相關資訊，請參閱父區段 (使用 Amazon EC2)。

主題

- 啟動 Amazon EC2 執行個體
- 終止 Amazon EC2 執行個體

啟動 Amazon EC2 執行個體

此範例說明如何使用 適用於 .NET 的 AWS SDK ，從相同的 Amazon Machine Image (AMI) 啟動一或多個設定相同的 Amazon EC2 執行個體。使用您提供的數個輸入，應用程式會啟動 EC2 執行個體，然後監控執行個體，直到其處於「待定」狀態為止。

當 EC2 執行個體執行時，您可以遠端連線到它，如中所述（選用）連線至執行個體。

> **Warning**
>
> EC2-Classic 在 2022 年 8 月 15 日淘汰。建議您從 EC2-Classic 遷移至 VPC。如需詳細資訊，請參閱部落格文章 EC2-Classic Networking 正在淘汰 – 以下是如何準備。

下列各節提供此範例的程式碼片段和其他資訊。範例的完整程式碼會顯示在程式碼片段之後，並可依原樣建置和執行。

主題

- 收集您需要的內容
- 啟動執行個體
- 監控執行個體

- 完成程式碼
- 其他考量
- （選用）連線至執行個體
- 清除

收集您需要的內容

若要啟動 EC2 執行個體，您需要幾件事。

- 將啟動執行個體的 VPC。如果它是 Windows 執行個體，而且您將透過 RDP 連接到它，VPC 很可能需要連接網際網路閘道，以及路由表中網際網路閘道的項目。如需詳細資訊，請參閱《Amazon VPC 使用者指南》中的網際網路閘道。

- 要啟動執行個體之 VPC 中現有子網路的 ID。尋找或建立此項目的簡單方法是登入 Amazon VPC 主控台，但您也可以使用 CreateSubnetAsync 和 DescribeSubnetsAsync 方法以程式設計方式取得。

  > **Note**
  >
  > 如果您未提供此參數，新的執行個體會在您帳戶的預設 VPC 中啟動。

- 屬於要啟動執行個體之 VPC 的現有安全群組 ID。如需詳細資訊，請參閱在 Amazon EC2 中使用安全群組。

- 如果您想要連線到新的執行個體，先前提及的安全群組必須具有適當的傳入規則，允許連接埠 22 (Linux 執行個體）上的 SSH 流量或連接埠 3389 (Windows 執行個體）上的 RDP 流量。如需如何執行此操作的詳細資訊，請參閱 更新安全群組，包括其他考量接近該主題結尾的 。

- 將用於建立執行個體的 Amazon Machine Image (AMI)。如需 AMIs的相關資訊，請參閱《Amazon EC2 使用者指南》中的 Amazon Machine Image (AMIs)。 Amazon EC2 特別是，請參閱尋找 AMI 和共用 AMIs。

- 現有 EC2 金鑰對的名稱，用於連線至新執行個體。如需詳細資訊，請參閱使用 Amazon EC2 金鑰對。

- 包含先前提及之 EC2 金鑰對私有金鑰的 PEM 檔案名稱。當您從遠端連線至執行個體時，會使用 PEM 檔案。

啟動執行個體

下列程式碼片段會啟動 EC2 執行個體。

本主題結尾附近的範例顯示此程式碼片段正在使用中。

```
    //
    // 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;
    }
```

監控執行個體

下列程式碼片段會監控執行個體，直到其處於「待定」狀態為止。

本主題結尾附近的範例顯示此程式碼片段正在使用中。

如需 `Instance.State.Code` 屬性的有效值，請參閱 InstanceState 類別。

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

```
    }
```

完成程式碼

本節顯示此範例的相關參考和完整程式碼。

開發套件參考

NuGet 套件：

- AWSSDK.EC2

程式設計元素：

- 命名空間 Amazon.EC2

  AmazonEC2Client 類別

  類別 InstanceType
- 命名空間 Amazon.EC2.Model

  類別 DescribeInstancesRequest

  類別 DescribeInstancesResponse

  類別執行個體

  類別 InstanceNetworkInterfaceSpecification

  類別 RunInstancesRequest

  類別 RunInstancesResponse

程式碼

```
using System;
using System.Threading;
using System.Threading.Tasks;
using System.Collections.Generic;
using Amazon.EC2;
using Amazon.EC2.Model;
```

```
namespace EC2LaunchInstance
{
  // = = =
```

```
    }


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

```
 // = = =
```

```
        }
      }

      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);
    }
  }

}
```

其他考量

- 檢查 EC2 執行個體的狀態時，您可以將篩選條件新增至 DescribeInstancesRequest 物件的 `Filter` 屬性。使用此技術，您可以將請求限制為特定執行個體，例如具有特定使用者指定標籤的執行個體。

- 為了簡潔起見，某些屬性被給予典型值。任何或所有這些屬性都可以透過程式設計方式或使用者輸入來決定。

- 您可以用於 RunInstancesRequest 物件的 `MinCount`和 `MaxCount` 屬性的值取決於目標可用區域，以及執行個體類型允許的執行個體數量上限。如需詳細資訊，請參閱 Amazon EC2 一般常見問答集中的我可以在 Amazon EC2 中執行多少執行個體。 Amazon EC2

- 如果您想要使用與此範例不同的執行個體類型，有數種執行個體類型可供選擇。如需詳細資訊，請參閱《Amazon EC2 使用者指南》中的 Amazon EC2 執行個體類型。 Amazon EC2 另請參閱執行個體類型詳細資訊和執行個體類型總管。

- 您也可以在啟動執行個體時，將 IAM 角色連接至執行個體。若要這樣做，請建立`Name`屬性設定為 IAM 角色名稱的 IamInstanceProfileSpecification 物件。然後將該物件新增至 RunInstancesRequest 物件的 `IamInstanceProfile` 屬性。

  > **Note**
  >
  > 若要啟動已連接 IAM 角色的 EC2 執行個體，IAM 使用者的組態必須包含特定許可。如需所需許可的詳細資訊，請參閱《Amazon EC2 使用者指南》中的授予使用者將 IAM 角色傳遞至執行個體的許可。

（選用）連線至執行個體

執行個體執行後，您可以使用適當的遠端用戶端從遠端連線至執行個體。對於 Linux 和 Windows 執行個體，您需要執行個體的公有 IP 地址或公有 DNS 名稱。您也需要下列項目。

針對 Linux 執行個體

您可以使用 SSH 用戶端連線到 Linux 執行個體。請確定您在啟動執行個體時使用的安全群組允許連接埠 22 上的 SSH 流量，如中所述更新安全群組。

您也需要用來啟動執行個體之金鑰對的私有部分，也就是 PEM 檔案。

如需詳細資訊，請參閱《Amazon EC2 使用者指南》中的連線至 Linux 執行個體。

對於 Windows 執行個體

您可以使用 RDP 用戶端連線到您的執行個體。請確定您在啟動執行個體時使用的安全群組允許連接埠 3389 上的 RDP 流量，如中所述更新安全群組。

您也需要管理員密碼。您可以使用下列範例程式碼來取得此項目，該程式碼需要執行個體 ID 和用來啟動執行個體之金鑰對的私有部分，也就是 PEM 檔案。

如需詳細資訊，請參閱《Amazon EC2 使用者指南》中的連線至 Windows 執行個體。

> **Warning**
>
> 此範例程式碼會傳回執行個體的純文字管理員密碼。

開發套件參考

NuGet 套件：

- AWSSDK.EC2

程式設計元素：

- 命名空間 Amazon.EC2

  AmazonEC2Client 類別
- 命名空間 Amazon.EC2.Model

  類別 GetPasswordDataRequest

  類別 GetPasswordDataResponse

程式碼

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Threading.Tasks;
using Amazon.EC2;
using Amazon.EC2.Model;

namespace EC2GetWindowsPassword
{
```

```
 // = = =
```

```
      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);
    }
  }

}
```

清除

當您不再需要 EC2 執行個體時，請務必將其終止，如中所述終止 Amazon EC2 執行個體。

終止 Amazon EC2 執行個體

當您不再需要一或多個 Amazon EC2 執行個體時，您可以終止它們。

此範例說明如何使用 適用於 .NET 的 AWS SDK 來終止 EC2 執行個體。它需要一個執行個體 ID 做為輸入。

開發套件參考

NuGet 套件：

- AWSSDK.EC2

程式設計元素：

- 命名空間 Amazon.EC2

  AmazonEC2Client 類別
- 命名空間 Amazon.EC2.Model

  類別 TerminateInstancesRequest

類別 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);
      }
    }
  }
}
```

執行範例後，最好登入 Amazon EC2 主控台，以確認 EC2 執行個體已終止。

Amazon EC2 Spot 執行個體教學課程

本教學課程說明如何使用 適用於 .NET 的 AWS SDK 來管理 Amazon EC2 Spot 執行個體。

概觀

Spot 執行個體可讓您以低於隨需價格的價格請求未使用的 Amazon EC2 容量。這可以大幅降低可中斷之應用程式的 EC2 成本。

以下是 Spot 執行個體的請求和使用方式的高階摘要。

1. 建立 Spot 執行個體請求，指定您願意支付的最高價格。
2. 完成請求後，像執行任何其他 Amazon EC2 執行個體一樣執行執行個體。
3. 視需要執行執行個體，然後終止執行個體，除非 Spot 價格變更，讓執行個體為您終止。
4. 當您不再需要 Spot 執行個體請求時，請將其清除，以便不再建立 Spot 執行個體。

這是 Spot 執行個體的非常高階概觀。若要進一步了解 Spot 執行個體，請參閱《Amazon EC2 使用者指南》中的 Spot 執行個體。

關於本教學

當您遵循本教學課程時，您可以使用 適用於 .NET 的 AWS SDK 執行下列動作：

- 建立 Spot 執行個體請求
- 判斷 Spot 執行個體請求何時完成
- 取消 Spot 執行個體請求
- 終止關聯的執行個體

下列各節提供此範例的程式碼片段和其他資訊。範例的完整程式碼會顯示在程式碼片段之後，並可依原樣建置和執行。

主題

- 先決條件
- 收集您需要的內容
- 建立 Spot 執行個體請求
- 判斷 Spot 執行個體請求的狀態
- 清除 Spot 執行個體請求
- 清除 Spot 執行個體
- 完成程式碼
- 其他考量

先決條件

如需 APIs和先決條件的相關資訊，請參閱父區段 (使用 Amazon EC2)。

收集您需要的內容

若要建立 Spot 執行個體請求，您需要幾件事。

- 執行個體數量及其執行個體類型。有數種執行個體類型可供選擇。如需詳細資訊，請參閱《Amazon EC2 使用者指南》中的 Amazon EC2 執行個體類型。 Amazon EC2 另請參閱執行個體類型詳細資訊和執行個體類型總管。

  此教學課程的預設號碼為 1。

- 將用於建立執行個體的 Amazon Machine Image (AMI)。如需 AMIs的相關資訊，請參閱《Amazon EC2 使用者指南》中的 Amazon Machine Image (AMIs)。 Amazon EC2 特別是，請參閱尋找 AMI 和共用 AMIs。

- 您願意為每個執行個體小時支付的最高價格。您可以在 Amazon EC2 定價頁面上查看所有執行個體類型 （適用於隨需執行個體和 Spot 執行個體） 的價格。本教學課程的預設價格將於稍後說明。

- 如果您想要從遠端連線至執行個體，則為具有適當組態和資源的安全群組。這在 中說明，在 Amazon EC2 中使用安全群組以及收集您需要的內容並連接到 中執行個體的相關資訊啟動 Amazon EC2 執行個體。為求簡化，本教學課程使用所有較新 AWS 帳戶預設的安全群組。

請求 Spot 執行個體的方法有很多種。以下是常見的策略：

- 提出確定成本低於隨需定價的請求。
- 根據產生的運算值提出請求。
- 提出請求以盡快取得運算容量。

下列說明參考《Amazon EC2 使用者指南》中的 Spot 執行個體定價歷史記錄。

將成本降低到低於隨需

您可以批次處理任務，需要花費數小時或數天的時間完成。但是，在開始和結束時都有彈性。您想要查看是否能以低於隨需執行個體的成本將它完成。

您可以使用 Amazon EC2 主控台或 Amazon EC2 API 來檢查執行個體類型的 Spot 價格歷史記錄。於一個給定的可用區域內分析您所想要的執行個體類型的歷史價格後，您的出價有兩個替代方式：

- 在 Spot 價格範圍的上端指定請求，該請求仍低於隨需價格，預期您的一次性 Spot 執行個體請求最有可能滿足並執行足夠的連續運算時間來完成任務。
- 或者，您可以出價此 Spot 價格範圍的下限，並計畫如何透過一個持久性的請求來結合數個已長期啟動的執行個體。該執行個體將總共要執行一段夠長時間，才能以更低的成本完成任務。

支付不超過結果值的費用

您有一個要執行的資料正在處理任務。您充分了解任務結果的價值，以得知其在運算成本方面的價值。

分析執行個體類型的 Spot 價格歷史記錄後，您可以選擇運算時間成本不超過任務結果值的價格。您建立可以長久出價的方式，且允許它在 Spot 價格出現波動並等於或低於您的出價時，間歇性地執行。

快速取得運算容量

對於無法透過隨需執行個體取得的額外容量，您有非預期的短期需求。分析執行個體類型的 Spot 價格歷史記錄後，您可以選擇高於最高歷史價格的價格，以大幅提高快速履行請求的可能性，並繼續運算，直到完成為止。

收集所需的內容並選擇策略之後，您就可以請求 Spot 執行個體。在本教學課程中，預設最大 Spot 執行個體價格設定為與隨需價格相同 (在本教學課程中為 $0.003 美元)。以這種方法設定價格能夠最大化實現請求的機會。

建立 Spot 執行個體請求

下列程式碼片段說明如何使用先前收集的元素建立 Spot 執行個體請求。

本主題結尾的範例顯示此程式碼片段正在使用中。

```
    //
    // 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];
    }
```

從此方法傳回的重要值是 Spot 執行個體請求 ID，此 ID 包含在傳回的 SpotInstanceRequest 物件`SpotInstanceRequestId`的成員中。

> **Note**
>
> 您需要為啟動的任何 Spot 執行個體支付費用。為了避免不必要的成本，請務必取消任何請求並終止任何執行個體。

判斷 Spot 執行個體請求的狀態

下列程式碼片段說明如何取得 Spot 執行個體請求的相關資訊。您可以使用該資訊在程式碼中做出特定決策，例如是否繼續等待 Spot 執行個體請求履行。

本主題結尾的範例顯示此程式碼片段正在使用中。

```
    //
    // 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];
    }
```

方法會傳回 Spot 執行個體請求的相關資訊，例如執行個體 ID、其狀態和狀態碼。如需 Spot 執行個體請求狀態碼的詳細資訊，請參閱《Amazon EC2 使用者指南》中的 Spot 請求狀態。

清除 Spot 執行個體請求

當您不再需要請求 Spot 執行個體時，請務必取消任何未完成的請求，以防止這些請求重新履行。下列程式碼片段說明如何取消 Spot 執行個體請求。

本主題結尾的範例顯示此程式碼片段正在使用中。

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

清除 Spot 執行個體

為了避免不必要的成本，請務必終止從 Spot 執行個體請求啟動的任何執行個體；只要取消 Spot 執行個體請求就不會終止您的執行個體，這表示您將繼續支付這些執行個體的費用。下列程式碼片段示範如何在取得作用中 Spot 執行個體的執行個體識別符後終止執行個體。

本主題結尾的範例顯示此程式碼片段正在使用中。

```
    //
```

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

完成程式碼

下列程式碼範例會呼叫先前所述的方法來建立和取消 Spot 執行個體請求，並終止 Spot 執行個體。

開發套件參考

NuGet 套件：

- AWSSDK.EC2

程式設計元素：

- 命名空間 Amazon.EC2

AmazonEC2Client 類別

類別 InstanceType

- 命名空間 Amazon.EC2.Model

  類別 CancelSpotInstanceRequestsRequest

  類別 DescribeSpotInstanceRequestsRequest

  類別 DescribeSpotInstanceRequestsResponse

  類別 InstanceStateChange

  類別 LaunchSpecification

  類別 RequestSpotInstancesRequest

  Class RequestSpotInstancesResponse

  類別 SpotInstanceRequest

  類別 TerminateInstancesRequest

  類別 TerminateInstancesResponse

程式碼

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

其他考量

- 執行教學課程後，最好登入 Amazon EC2 主控台，確認 Spot 執行個體請求已取消，且 Spot 執行個體已終止。

使用 存取 AWS Identity and Access Management (IAM) 適用於 .NET 的 AWS SDK

適用於 .NET 的 AWS SDK 支援 AWS Identity and Access Management，這是一種 Web 服務，可讓 AWS 客戶管理 中的使用者和使用者許可 AWS。

AWS Identity and Access Management (IAM) 使用者是您建立的實體 AWS。實體代表與 互動的人員或應用程式 AWS。如需 IAM 使用者的詳細資訊，請參閱《IAM 使用者指南》中的 IAM 使用者和 IAM 和 STS 限制。

您可以透過建立 IAM 政策將許可授予使用者。政策包含的政策文件列出使用者可以執行的動作，以及這些動作可以影響的資源。如需 IAM 政策的詳細資訊，請參閱《IAM 使用者指南》中的政策和許可。

> **Warning**
>
> 為避免安全風險，在開發專用軟體或使用真實資料時，請勿使用 IAM 使用者進行身分驗證。相反地，搭配使用聯合功能和身分提供者，例如 AWS IAM Identity Center。

API

為 IAM 用戶端 適用於 .NET 的 AWS SDK 提供 APIs。APIs 可讓您使用 IAM 功能，例如使用者、角色和存取金鑰。

本節包含一些範例，顯示您在使用這些 APIs 時可遵循的模式。若要檢視完整的 APIs 集，請參閱 適用於 .NET 的 AWS SDK API 參考（並捲動至「Amazon.IdentityManagement」）。

本節也包含範例，說明如何將 IAM 角色連接至 Amazon EC2 執行個體，以便更輕鬆地管理登入資料。

IAM APIs 由 AWSSDK.IdentityManagement NuGet 套件提供。

先決條件

開始之前，請確定您已設定環境並設定專案。也請檢閱 中的資訊使用開發套件。

主題

主題

- 從 JSON 建立 IAM 受管政策
- 顯示 IAM 受管政策的政策文件
- 使用 IAM 角色授予存取權

從 JSON 建立 IAM 受管政策

此範例說明如何使用 從 JSON 中的指定政策文件中 適用於 .NET 的 AWS SDK 建立 IAM 受管政策。應用程式會建立 IAM 用戶端物件、從 檔案讀取政策文件，然後建立政策。

> **Note**
>
> 如需 JSON 中的範例政策文件，請參閱本主題結尾的其他考量事項。

下列各節提供此範例的程式碼片段。之後會顯示範例的完整程式碼，並可依原樣建置和執行。

主題

- 建立 政策
- 完成程式碼
- 其他考量

建立 政策

下列程式碼片段會建立具有指定名稱和政策文件的 IAM 受管政策。

本主題結尾的範例顯示此程式碼片段的使用中。

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

完成程式碼

本節顯示此範例的相關參考和完整程式碼。

開發套件參考

NuGet 套件：

- AWSSDK.IdentityManagement

程式設計元素：

- 命名空間 Amazon.IdentityManagement

  AmazonIdentityManagementServiceClient 類別
- 命名空間 Amazon.IdentityManagement.Model

  類別 CreatePolicyRequest

  類別 CreatePolicyResponse

程式碼

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Threading.Tasks;
using Amazon.IdentityManagement;
using Amazon.IdentityManagement.Model;

namespace IamCreatePolicyFromJson
{
  // = = =
```

```
      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);
    }
  }

}
```

其他考量

- 以下是您可以複製到 JSON 檔案並用作此應用程式輸入的範例政策文件：

  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": "*"
    }
  ]
}
```

- 您可以在 IAM 主控台中查看政策是否已建立。在篩選政策下拉式清單中，選取客戶受管。當您不再需要政策時，請將其刪除。

- 如需政策建立的詳細資訊，請參閱《IAM 使用者指南》中的建立 IAM 政策和 IAM JSON 政策參考

顯示 IAM 受管政策的政策文件

此範例說明如何使用 適用於 .NET 的 AWS SDK 來顯示政策文件。應用程式會建立 IAM 用戶端物件、尋找指定 IAM 受管政策的預設版本，然後在 JSON 中顯示政策文件。

下列各節提供此範例的程式碼片段。之後會顯示範例的完整程式碼，並可依原樣建置和執行。

主題

- 尋找預設版本
- 顯示政策文件
- 完成程式碼

尋找預設版本

下列程式碼片段會尋找指定 IAM 政策的預設版本。

本主題結尾的範例顯示此程式碼片段正在使用中。

```
    //
    // 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;
    }
```

顯示政策文件

下列程式碼片段會以指定 IAM 政策的 JSON 顯示政策文件。

本主題結尾的範例顯示此程式碼片段的使用中。

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

完成程式碼

本節顯示此範例的相關參考和完整程式碼。

開發套件參考

NuGet 套件：

- AWSSDK.IdentityManagement

程式設計元素：

- 命名空間 Amazon.IdentityManagement

  AmazonIdentityManagementServiceClient 類別
- 命名空間 Amazon.IdentityManagement.Model

  類別 GetPolicyVersionRequest

  類別 GetPolicyVersionResponse

  類別 ListPolicyVersionsRequest

  類別 ListPolicyVersionsResponse

  類別PolicyVersion

程式碼

```
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 角色授予存取權

本教學課程說明如何使用 適用於 .NET 的 AWS SDK 在 Amazon EC2 執行個體上啟用 IAM 角色。

概觀

對 的所有請求 AWS 都必須使用 發行的登入資料以密碼編譯方式簽署 AWS。因此，您需要策略來管理在 Amazon EC2 執行個體上執行之應用程式的登入資料。您必須安全地分發、存放和輪換這些登入資料，同時讓應用程式能夠存取這些登入資料。

透過 IAM 角色，您可以有效地管理這些登入資料。您可以建立 IAM 角色，並使用應用程式所需的許可進行設定，然後將該角色連接到 EC2 執行個體。若要進一步了解使用 IAM 角色的好處，請參閱《Amazon EC2 使用者指南》中的 Amazon EC2 的 IAM 角色。另請參閱《IAM 使用者指南》中的 IAM 角色相關資訊。

對於使用 建置的應用程式 適用於 .NET 的 AWS SDK，當應用程式建構 AWS 服務的用戶端物件時，物件會從數個潛在來源搜尋登入資料。其搜尋的順序會顯示在 中憑證和設定檔解析。

如果用戶端物件找不到任何其他來源的登入資料，則會擷取具有與已設定為 IAM 角色且位於 EC2 執行個體中繼資料中相同許可的臨時登入資料。這些登入資料用於 AWS 從用戶端物件呼叫 。

關於本教學

遵循本教學課程時，您可以使用 適用於 .NET 的 AWS SDK （和其他工具） 啟動已連接 IAM 角色的 Amazon EC2 執行個體，然後使用 IAM 角色的許可查看執行個體上的應用程式。

主題

- 建立範例 Amazon S3 應用程式
- 建立 IAM 角色
- 啟動 EC2 執行個體並連接 IAM 角色
- 連線至 EC2 執行個體
- 在 EC2 執行個體上執行範例應用程式
- 清除

建立範例 Amazon S3 應用程式

此範例應用程式會從 Amazon S3 擷取物件。若要執行應用程式，您需要下列項目：

- 包含文字檔案的 Amazon S3 儲存貯體。
- AWS 開發機器上的登入資料，可讓您存取 儲存貯體。

如需有關建立 Amazon S3 儲存貯體和上傳物件的資訊，請參閱 Amazon Simple Storage Service 使用者指南。如需 AWS 登入資料的資訊，請參閱 適用於 .NET 的 AWS SDK 使用 驗證 AWS。

使用下列程式碼建立 .NET Core 專案。然後在開發機器上測試應用程式。

> **Note**
>
> 在您的開發機器上安裝 .NET Core Runtime，可讓您執行應用程式而不發佈。當您稍後在本教學課程中建立 EC2 執行個體時，您可以選擇在執行個體上安裝 .NET Core Runtime。這為您提供了類似的體驗和較小的檔案傳輸。
> 不過，您也可以選擇不要在執行個體上安裝 .NET Core Runtime。如果您選擇此動作，則必須發佈應用程式，以便在將應用程式轉移到執行個體時包含所有相依性。

開發套件參考

NuGet 套件：

- AWSSDK.S3

程式設計元素：

- 命名空間 Amazon.S3

  AmazonS3Client 類別
- 命名空間 Amazon.S3.Model

  類別 GetObjectResponse

程式碼

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Threading.Tasks;
using Amazon.S3;
using Amazon.S3.Model;

namespace S3GetTextItem
{
```

```
 // = = =
```

```
      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.");
    }
 }


 // = = =
```

```
      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，您可以暫時移除在開發機器上使用的登入資料，以查看應用程式如何回應。（但請務必在完成時還原登入資料。）

建立 IAM 角色

建立具有適當許可的 IAM 角色來存取 Amazon S3。

1. 開啟 IAM 主控台。
2. 在導覽窗格中，選擇角色，然後選擇建立角色。
3. 選取AWS 服務，尋找並選擇 EC2，然後選擇下一步：許可。
4. 在連接許可政策下，尋找並選取 AmazonS3ReadOnlyAccess。如果您想要的話，請檢閱政策，然後選擇下一步：標籤。
5. 如果需要，請新增標籤，然後選擇下一步：檢閱。
6. 輸入該角色的名稱和說明，然後選擇 Create role (建立新角色)。請記住此名稱，因為在您啟動 EC2 執行個體時需要用到。

啟動 EC2 執行個體並連接 IAM 角色

使用您先前建立的 IAM 角色啟動 EC2 執行個體。您可以透過下列方式執行此操作。

- 使用 EC2 主控台

  若要使用 EC2 主控台啟動執行個體，請參閱《Amazon EC2 使用者指南》中的使用新的啟動執行個體精靈啟動執行個體。

  當您查看啟動頁面時，您至少應該展開進階詳細資訊窗格，以便您可以指定先前在 IAM 執行個體描述檔中建立的 IAM 角色。

- 使用 適用於 .NET 的 AWS SDK

  如需相關資訊，請參閱 啟動 Amazon EC2 執行個體，包括該主題其他考量接近結尾的 。

若要啟動已連接 IAM 角色的 EC2 執行個體，IAM 使用者的組態必須包含特定許可。如需所需許可的詳細資訊，請參閱《Amazon EC2 使用者指南》中的授予使用者將 IAM 角色傳遞至執行個體的許可。

連線至 EC2 執行個體

連線至 EC2 執行個體，以便您可以將範例應用程式轉移至該執行個體，然後執行應用程式。您需要包含用來啟動執行個體之金鑰對私有部分的 檔案，也就是 PEM 檔案。

如需連接至執行個體的資訊，請參閱《Amazon EC2 使用者指南》中的連接至 Linux 執行個體或連接至 Windows 執行個體。當您連線時，請以可讓您將檔案從開發機器傳輸到執行個體的方式執行此操作。

如果您在 Windows 上使用 Visual Studio，您也可以使用 Toolkit for Visual Studio 連線到執行個體。如需詳細資訊，請參閱 AWS Toolkit for Visual Studio 《 使用者指南》中的連線至 Amazon EC2 執行個體。

在 EC2 執行個體上執行範例應用程式

1. 將應用程式檔案從本機磁碟機複製到執行個體。

   傳輸哪些檔案取決於您建置應用程式的方式，以及執行個體是否已安裝 .NET Core Runtime。如需有關如何將檔案傳輸至執行個體的資訊，請參閱《Amazon EC2 使用者指南》中的連線至 Linux 執行個體 （請參閱適當的子區段） 或將檔案傳輸至 Windows 執行個體。
2. 啟動應用程式，並確認其執行的結果與開發機器上的結果相同。
3. 確認應用程式使用 IAM 角色提供的登入資料。

   a. 開啟 Amazon EC2 主控台。

   b. 選取執行個體，並透過動作、執行個體設定、連接/取代 IAM 角色來分離 IAM 角色。

   c. 再次執行應用程式，並查看它傳回授權錯誤。

清除

完成本教學課程後，如果您不再需要您建立的 EC2 執行個體，請務必終止執行個體，以避免不必要的成本。您可以在 Amazon EC2 主控台或以程式設計方式執行此操作，如中所述終止 Amazon EC2 執行

個體。如果您願意，也可以刪除您為此教學課程建立的其他資源。這些可能包括 IAM 角色、EC2 金鑰對和 PEM 檔案、安全群組等。

使用 Amazon Simple Storage Service 網際網路儲存

適用於 .NET 的 AWS SDK 支援 Amazon S3，這是網際網路的儲存體。此服務旨在降低開發人員進行網路規模運算的難度。

API

為 Amazon S3 用戶端 適用於 .NET 的 AWS SDK 提供 APIs。APIs 可讓您使用 Amazon S3 資源，例如儲存貯體和項目。若要檢視 Amazon S3 的完整 APIs 集，請參閱以下內容：

- 適用於 .NET 的 AWS SDK API 參考 （並捲動至 "Amazon.S3")。
- Amazon.Extensions.S3.Encryption 文件

Amazon S3 APIs 由下列 NuGet 套件提供：

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

先決條件

開始之前，請確定您已設定環境並設定專案。也請檢閱 中的資訊使用開發套件。

本文件中的範例

本文件中的下列主題說明如何使用 適用於 .NET 的 AWS SDK 來使用 Amazon S3。

- 使用 KMS 金鑰進行 S3 加密

其他文件中的範例

下列 Amazon S3 開發人員指南的連結提供如何使用 適用於 .NET 的 AWS SDK 與 Amazon S3 搭配使用的其他範例。

> **Note**
>
> 雖然這些範例和其他程式設計考量是針對 適用於 .NET 的 AWS SDK 使用 .NET Framework 的第 3 版所建立，但它們也適用於 適用於 .NET 的 AWS SDK 使用 .NET Core 的較新版本。程式碼中的小調整有時是必要的。

Amazon S3 程式設計範例

- 管理 ACL
- 建立儲存貯體
- 上傳物件
- 使用高階 API 進行分段上傳 (Amazon.S3.Transfer.TransferUtility)
- 使用低階 API 執行分段上傳
- 列出物件
- 列出金鑰
- 取得物件
- 複製物件
- 使用分段上傳 API 複製物件
- 刪除物件
- 刪除多個物件
- 還原物件
- 設定儲存貯體的通知
- 管理物件的生命週期
- 產生預先簽章的物件 URL
- 管理網站
- 啟用跨來源資源分享 (CORS)

其他程式設計考量事項

- 使用 適用於 .NET 的 AWS SDK 進行 Amazon S3 程式設計
- 使用 IAM 使用者暫時性登入資料提出請求

- 使用聯合身分使用者暫時登入資料提出請求
- 指定伺服器端加密
- 使用客戶提供的加密金鑰指定伺服器端加密

在 中使用 AWS KMS 金鑰進行 Amazon S3 加密 適用於 .NET 的 AWS SDK

此範例說明如何使用 AWS Key Management Service 金鑰來加密 Amazon S3 物件。應用程式會建立客戶主金鑰 (CMK)，並使用它來建立 AmazonS3EncryptionClientV2 物件以進行用戶端加密。應用程式使用該用戶端，從現有 Amazon S3 儲存貯體中的指定文字檔案建立加密物件。然後，它會解密物件並顯示其內容。

> **Warning**
>
> 稱為 的類似類別`AmazonS3EncryptionClient`已棄用，且比`AmazonS3EncryptionClientV2`類別更不安全。若要遷移使用 的現有程式碼`AmazonS3EncryptionClient`，請參閱 S3 加密用戶端遷移。

主題

- 建立加密資料
- 建立和加密 Amazon S3 物件
- 完成程式碼
- 其他考量

建立加密資料

下列程式碼片段會建立包含 KMS 金鑰 ID 的`EncryptionMaterials`物件。

本主題結尾的範例顯示此程式碼片段正在使用中。

```
      // 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);
```

建立和加密 Amazon S3 物件

下列程式碼片段會建立使用先前建立之加密資料的AmazonS3EncryptionClientV2物件。然後，它會使用用戶端來建立和加密新的 Amazon S3 物件。

本主題結尾的範例顯示此程式碼片段正在使用中。

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

完成程式碼

本節顯示此範例的相關參考和完整程式碼。

開發套件參考

NuGet 套件：

- Amazon.Extensions.S3.Encryption

程式設計元素：

- 命名空間 Amazon.Extensions.S3.Encryption

  AmazonS3EncryptionClientV2 類別

  AmazonS3CryptoConfigurationV2 類別

  類別 CryptoStorageMode

  類別 EncryptionMaterialsV2
- 命名空間 Amazon.Extensions.S3.Encryption.Primitives

  類別 KmsType
- 命名空間 Amazon.S3.Model

  類別 GetObjectRequest

  類別 GetObjectResponse

  類別 PutObjectRequest
- 命名空間 Amazon.KeyManagementService

  AmazonKeyManagementServiceClient 類別
- 命名空間 Amazon.KeyManagementService.Model

  類別 CreateKeyRequest

  類別 CreateKeyResponse

程式碼

```
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
{
  // = = =
```

```
      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.");
    }

  }

  // = = =
```

```
          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);
```

```
    }
  }

}
```

其他考量

- 您可以檢查此範例的結果。若要這樣做，請前往 Amazon S3 主控台並開啟您提供給應用程式的儲存貯體。然後尋找新物件、下載它，然後在文字編輯器中開啟它。

- AmazonS3EncryptionClientV2 類別實作與標準AmazonS3Client類別相同的界面。這可讓您更輕鬆地將程式碼移植到 AmazonS3EncryptionClientV2類別，以便在用戶端中自動且透明地進行加密和解密。

- 使用 AWS KMS 金鑰做為主金鑰的一個優點是您不需要儲存和管理自己的主金鑰；這由 完成 AWS。第二個優點是 的 AmazonS3EncryptionClientV2類別可與 的 AmazonS3EncryptionClientV2類別 適用於 .NET 的 AWS SDK 互通 適用於 Java 的 AWS SDK。這表示您可以使用 加密 適用於 Java 的 AWS SDK ，並使用 解密 適用於 .NET 的 AWS SDK，反之亦然。

  > **Note**
  >
  > 的 AmazonS3EncryptionClientV2類別僅在中繼資料模式下執行時 適用於 .NET 的 AWS SDK 支援 KMS 主金鑰。AmazonS3EncryptionClientV2 類別 的指示檔案模式與 AmazonS3EncryptionClientV2類別 適用於 .NET 的 AWS SDK 不相容 適用於 Java 的 AWS SDK。

- 如需使用 AmazonS3EncryptionClientV2類別進行用戶端加密，以及信封加密如何運作的詳細資訊，請參閱使用 適用於 .NET 的 AWS SDK 和 Amazon S3 進行用戶端資料加密。

使用 Amazon Simple Notification Service 從雲端傳送通知

> **Note**
>
> 本主題中的資訊專屬於以 .NET Framework 和 3.3 適用於 .NET 的 AWS SDK 版及更早版本為基礎的專案。

適用於 .NET 的 AWS SDK 支援 Amazon Simple Notification Service (Amazon SNS)，這是一種 Web 服務，可讓應用程式、最終使用者和裝置立即從雲端傳送通知。如需詳細資訊，請參閱 Amazon SNS。

列出您的 Amazon SNS 主題

下列範例說明如何列出 Amazon SNS 主題、每個主題的訂閱，以及每個主題的屬性。此範例使用預設的 AmazonSimpleNotificationServiceClient。

```
// using Amazon.SimpleNotificationService;
// using Amazon.SimpleNotificationService.Model;

var client = new AmazonSimpleNotificationServiceClient();
var request = new ListTopicsRequest();
var response = new ListTopicsResponse();

do
{
  response = client.ListTopics(request);

  foreach (var topic in response.Topics)
  {
    Console.WriteLine("Topic: {0}", topic.TopicArn);

    var subs = client.ListSubscriptionsByTopic(
      new ListSubscriptionsByTopicRequest
      {
        TopicArn = topic.TopicArn
      });

    var ss = subs.Subscriptions;

    if (ss.Any())
    {
```

```
      Console.WriteLine("  Subscriptions:");

      foreach (var sub in ss)
      {
        Console.WriteLine("    {0}", sub.SubscriptionArn);
      }
    }

    var attrs = client.GetTopicAttributes(
      new GetTopicAttributesRequest
      {
        TopicArn = topic.TopicArn
      }).Attributes;

    if (attrs.Any())
    {
      Console.WriteLine("  Attributes:");

      foreach (var attr in attrs)
      {
        Console.WriteLine("    {0} = {1}", attr.Key, attr.Value);
      }
    }

    Console.WriteLine();
  }

  request.NextToken = response.NextToken;

} while (!string.IsNullOrEmpty(response.NextToken));
```

傳送訊息至 Amazon SNS 主題

下列範例示範如何傳送訊息至 Amazon SNS 主題。此範例採用一個引數，即 Amazon SNS 主題的 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);
            }
        }
    }
}
```

請參閱 GitHub 上的完整範例，包括如何從命令列建立和執行範例的資訊。

傳送簡訊至一個電話號碼

以下範例說明如何傳送簡訊到電話號碼。此範例採用一個引數，即電話號碼，其必須採用註解中所述兩種格式之一。

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

請參閱 GitHub 上的完整範例，包括如何從命令列建立和執行範例的資訊。

使用 Amazon SQS 傳送訊息

適用於 .NET 的 AWS SDK 支援 Amazon Simple Queue Service (Amazon SQS)，這是一種訊息佇列服務，可處理系統中元件之間的訊息或工作流程。

Amazon SQS 佇列提供一種機制，可讓您在微型服務、分散式系統和無伺服器應用程式等軟體元件之間傳送、存放和接收訊息。這可讓您解耦這類元件，讓您無需設計和操作自己的簡訊系統。如需有關佇列和訊息如何在 Amazon SQS 中運作的資訊，請參閱《Amazon Simple Queue Service 開發人員指南》中的 Amazon SQS 教學課程和基本 Amazon SQS 架構。

> **Important**
>
> 由於佇列的分散式性質，Amazon SQS 無法保證您會收到精確的訊息。如果您需要保留訊息順序，請使用 Amazon SQS FIFO 佇列。

API

為 Amazon SQS 用戶端 適用於 .NET 的 AWS SDK 提供 APIs。APIs 可讓您使用 Amazon SQS 功能，例如佇列和訊息。本節包含一些範例，顯示您在使用這些 APIs 時可遵循的模式。若要檢視完整的 APIs 集，請參閱 適用於 .NET 的 AWS SDK API 參考 （並捲動至「Amazon.SQS」）。

Amazon SQS APIs 由 AWSSDK.SQS NuGet 套件提供。

先決條件

開始之前，請確定您已設定環境並設定專案。也請檢閱 中的資訊使用開發套件。

主題

主題

- 建立 Amazon SQS 佇列
- 更新 Amazon SQS 佇列

- 刪除 Amazon SQS 佇列
- 傳送 Amazon SQS 訊息
- 接收 Amazon SQS 訊息

建立 Amazon SQS 佇列

此範例說明如何使用 適用於 .NET 的 AWS SDK 來建立 Amazon SQS 佇列。如果您未提供無效字母佇列的 ARN，應用程式會建立無效字母佇列。然後，它會建立標準訊息佇列，其中包含無效字母佇列（您提供的佇列或建立的佇列）。

如果您未提供任何命令列引數，應用程式只會顯示所有現有佇列的相關資訊。

下列各節提供此範例的程式碼片段。之後會顯示範例的完整程式碼，並可依原樣建置和執行。

主題

- 顯示現有的佇列
- 建立佇列
- 取得佇列的 ARN
- 完成程式碼
- 其他考量

顯示現有的佇列

下列程式碼片段顯示 SQS 用戶端區域中現有佇列的清單，以及每個佇列的屬性。

本主題結尾的範例顯示此程式碼片段正在使用中。

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

建立佇列

下列程式碼片段會建立佇列。程式碼片段包含使用無效字母佇列，但您的佇列不一定需要無效字母佇列。

本主題結尾的範例顯示此程式碼片段正在使用中。

```
    //
    // 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;
    }
```

取得佇列的 ARN

下列程式碼片段會取得指定佇列 URL 所識別之佇列的 ARN。

本主題結尾的範例顯示此程式碼片段正在使用中。

```
    //
    // 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;
    }
```

完成程式碼

本節顯示此範例的相關參考和完整程式碼。

開發套件參考

NuGet 套件：

- AWSSDK.SQS

程式設計元素：

- 命名空間 Amazon.SQS

  AmazonSQSClient 類別

  類別 QueueAttributeName

- 命名空間 Amazon.SQS.Model

  類別 CreateQueueRequest

  類別 CreateQueueResponse

  類別 GetQueueAttributesResponse

  類別 ListQueuesResponse

程式碼

```
using System;
using System.Threading.Tasks;
using System.Collections.Generic;
using Amazon.SQS;
using Amazon.SQS.Model;

namespace SQSCreateQueue
{
  // = = =
```

```
      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}.");
    }
  }


  // = = =
```

```
        // 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);
    }
  }

}
```

其他考量

- 您的佇列名稱必須由英數字元、連字號和底線組成。
- 佇列名稱和佇列 URLs 區分大小寫
- 如果您需要佇列 URL，但只有佇列名稱，請使用其中一種`AmazonSQSClient.GetQueueUrlAsync`方法。
- 如需有關您可以設定之各種佇列屬性的資訊，請參閱《適用於 .NET 的 AWS SDK API 參考》中的 CreateQueueRequest，或《Amazon Simple Queue Service API 參考》中的 SetQueueAttributes。
- 此範例會為您建立的佇列上的所有訊息指定長輪詢。這是使用 `ReceiveMessageWaitTimeSeconds` 屬性來完成的。

  您也可以在呼叫 AmazonSQSClient 類別`ReceiveMessageAsync`的方法期間指定長輪詢。如需詳細資訊，請參閱接收 Amazon SQS 訊息。

  如需短輪詢與長輪詢的相關資訊，請參閱《Amazon Simple Queue Service 開發人員指南》中的短輪詢和長輪詢。
- 無效字母佇列是其他（來源）佇列可以為未成功處理的訊息設定目標的佇列。如需詳細資訊，請參閱《Amazon Simple Queue Service 開發人員指南》中的 Amazon SQS 無效字母佇列。
- 您也可以在 Amazon SQS 主控台中查看佇列清單和此範例的結果。

更新 Amazon SQS 佇列

此範例說明如何使用 適用於 .NET 的 AWS SDK 來更新 Amazon SQS 佇列。經過一些檢查後，應用程式會使用指定的值更新指定的屬性，然後顯示佇列的所有屬性。

如果命令列引數中只包含佇列 URL，則應用程式只會顯示佇列的所有屬性。

下列各節提供此範例的程式碼片段。之後會顯示範例的完整程式碼，並可依原樣建置和執行。

主題

- 顯示佇列屬性
- 驗證屬性名稱
- 更新佇列屬性
- 完成程式碼
- 其他考量

顯示佇列屬性

下列程式碼片段顯示指定佇列 URL 所識別之佇列的屬性。

本主題結尾的範例顯示此程式碼片段正在使用中。

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

驗證屬性名稱

下列程式碼片段會驗證要更新的屬性名稱。

本主題結尾的範例顯示此程式碼片段正在使用中。

```
//
// 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;
  }
```

更新佇列屬性

下列程式碼片段會更新指定佇列 URL 所識別之佇列的屬性。

本主題結尾的範例顯示此程式碼片段正在使用中。

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

完成程式碼

本節顯示此範例的相關參考和完整程式碼。

開發套件參考

NuGet 套件：

- AWSSDK.SQS

程式設計元素：

- 命名空間 Amazon.SQS

  AmazonSQSClient 類別

  類別 QueueAttributeName
- 命名空間 Amazon.SQS.Model

  類別 GetQueueAttributesResponse

程式碼

```
using System;
```

```
using System.Collections.Generic;
using System.Threading.Tasks;
using Amazon.SQS;
using Amazon.SQS.Model;

namespace SQSUpdateQueue
{
  // = = =
```

```
    // 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.");
    }
  }


  // = = =
```

```
    // 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);
    }
  }

}
```

其他考量

- 若要更新 RedrivePolicy 屬性，您必須根據您的作業系統，引用整個值並逸出鍵/值對的引號。

  例如，在 Windows 上， 值的建構方式類似如下：

  ```
  "{\"deadLetterTargetArn\":\"DEAD_LETTER-QUEUE-ARN\",\"maxReceiveCount\":\"10\"}"
  ```

刪除 Amazon SQS 佇列

此範例說明如何使用 適用於 .NET 的 AWS SDK 刪除 Amazon SQS 佇列。應用程式會刪除佇列、等待一段指定的時間讓佇列消失，然後顯示剩餘的佇列清單。

如果您未提供任何命令列引數，應用程式只會顯示現有佇列的清單。

下列各節提供此範例的程式碼片段。之後會顯示範例的完整程式碼，並可依原樣建置和執行。

主題

- 刪除佇列
- 等待佇列消失
- 顯示現有佇列的清單
- 完成程式碼
- 其他考量

刪除佇列

下列程式碼片段會刪除指定佇列 URL 所識別的佇列。

本主題結尾的範例顯示此程式碼片段正在使用中。

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

等待佇列消失

下列程式碼片段會等待刪除程序完成，這可能需要 60 秒。

本主題結尾的範例顯示此程式碼片段正在使用中。

```
    //
    // 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;
      }
    }
```

顯示現有佇列的清單

下列程式碼片段顯示 SQS 用戶端區域中現有佇列的清單。

本主題結尾的範例顯示此程式碼片段正在使用中。

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

完成程式碼

本節顯示此範例的相關參考和完整程式碼。

開發套件參考

NuGet 套件：

- AWSSDK.SQS

程式設計元素：

- 命名空間 Amazon.SQS

  AmazonSQSClient 類別
- 命名空間 Amazon.SQS.Model

  類別 ListQueuesResponse

程式碼

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

namespace SQSDeleteQueue
{
  // = = =
```

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

其他考量

- DeleteQueueAsync API 呼叫不會檢查您要刪除的佇列是否用作無效字母佇列。更複雜的程序可以檢查此項目。

- 您也可以在 Amazon SQS 主控台中查看佇列清單和此範例的結果。

傳送 Amazon SQS 訊息

此範例說明如何使用 適用於 .NET 的 AWS SDK 傳送訊息至 Amazon SQS 佇列，您可以透過程式設計方式或使用 Amazon SQS 主控台建立該佇列。應用程式會將單一訊息傳送至佇列，然後傳送一批訊息。然後，應用程式會等待使用者輸入，這可以是要傳送至佇列的其他訊息或結束應用程式的請求。

此範例和接收訊息的下一個範例可以一起使用，以查看 Amazon SQS 中的訊息流程。

下列各節提供此範例的程式碼片段。之後會顯示範例的完整程式碼，並可依原樣建置和執行。

主題

- 傳送訊息
- 傳送一批訊息
- 從佇列刪除所有訊息

- 完成程式碼
- 其他考量

傳送訊息

下列程式碼片段會傳送訊息至指定佇列 URL 所識別的佇列。

本主題結尾的範例顯示此程式碼片段正在使用中。

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

傳送一批訊息

下列程式碼片段會將一批訊息傳送至指定佇列 URL 所識別的佇列。

本主題結尾的範例顯示此程式碼片段正在使用中。

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

從佇列刪除所有訊息

下列程式碼片段會從指定的佇列 URL 識別的佇列中刪除所有訊息。這也稱為清除佇列。

本主題結尾的範例顯示此程式碼片段正在使用中。

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

完成程式碼

本節顯示此範例的相關參考和完整程式碼。

開發套件參考

NuGet 套件：

- AWSSDK.SQS

程式設計元素：

- 命名空間 Amazon.SQS

  AmazonSQSClient 類別
- 命名空間 Amazon.SQS.Model

  類別 PurgeQueueResponse

  類別 SendMessageBatchResponse

  類別 SendMessageResponse

  類別 SendMessageBatchRequestEntry

  類別 SendMessageBatchResultEntry

程式碼

```
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Amazon.SQS;
using Amazon.SQS.Model;

namespace SQSSendMessages
{
  // = = =
```

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

其他考量

- 如需訊息各種限制的相關資訊，包括允許的字元，請參閱《Amazon Simple Queue Service 開發人員指南》中的訊息相關配額。

- 訊息會保留在佇列中，直到刪除或清除佇列為止。當應用程式收到訊息時，即使佇列中仍有訊息，它也不會出現在佇列中。如需可見性逾時的詳細資訊，請參閱 Amazon SQS 可見性逾時。

- 除了訊息內文之外，您也可以將屬性新增至訊息。如需詳細資訊，請參閱訊息中繼資料。

接收 Amazon SQS 訊息

此範例說明如何使用 從 Amazon SQS 佇列 適用於 .NET 的 AWS SDK 接收訊息，您可以透過程式設計方式或使用 Amazon SQS 主控台建立。應用程式會從佇列讀取單一訊息、處理訊息（在此情況下，會在主控台上顯示訊息內文），然後從佇列刪除訊息。應用程式會重複這些步驟，直到使用者在鍵盤上輸入金鑰為止。

此範例和先前有關傳送訊息的範例可以一起使用，以查看 Amazon SQS 中的訊息流程。

下列各節提供此範例的程式碼片段。之後會顯示範例的完整程式碼，並可依原樣建置和執行。

主題

- 接收訊息
- 刪除訊息
- 完成程式碼
- 其他考量

接收訊息

下列程式碼片段會從指定佇列 URL 識別的佇列接收訊息。

本主題結尾的範例顯示此程式碼片段正在使用中。

```
//
// 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.)
  });
}
```

刪除訊息

下列程式碼片段會從指定佇列 URL 識別的佇列中刪除訊息。

本主題結尾的範例顯示此程式碼片段正在使用中。

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

完成程式碼

本節顯示此範例的相關參考和完整程式碼。

開發套件參考

NuGet 套件：

- AWSSDK.SQS

程式設計元素：

- 命名空間 Amazon.SQS

  AmazonSQSClient 類別
- 命名空間 Amazon.SQS.Model

  類別 ReceiveMessageRequest

  類別 ReceiveMessageResponse

程式碼

```
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);
    }
  }
}
```

其他考量

- 若要指定長輪詢，此範例會針對 ReceiveMessageAsync方法的每個呼叫使用 WaitTimeSeconds 屬性。

  您也可以在建立或更新佇列時使用 ReceiveMessageWaitTimeSeconds 屬性，為佇列上的所有訊息指定長輪詢。

如需短輪詢與長輪詢的相關資訊，請參閱《Amazon Simple Queue Service 開發人員指南》中的短輪詢和長輪詢。

- 在訊息處理期間，您可以使用接收控點來變更訊息可見性逾時。如需如何執行此操作的詳細資訊，請參閱 AmazonSQSClient 類別`ChangeMessageVisibilityAsync`的方法。

- 無論可見性逾時設定為何，無條件呼叫 `DeleteMessageAsync`方法都會從佇列中移除訊息。

使用 AWS Lambda 進行運算服務

適用於 .NET 的 AWS SDK 支援 AWS Lambda，可讓您在不佈建或管理伺服器的情況下執行程式碼。如需詳細資訊，請參閱 AWS Lambda 產品頁面和 AWS Lambda 開發人員指南，特別是使用 C# 一節。

API

適用於 .NET 的 AWS SDK 提供 APIs AWS Lambda。APIs 可讓您使用函數、觸發條件和事件等 Lambda 功能。若要檢視完整的 APIs 集，請參閱 適用於 .NET 的 AWS SDK API 參考中的 Lambda。

Lambda APIs 由 NuGet 套件提供。

先決條件

開始之前，請確定您已設定環境並設定專案。也請檢閱 中的資訊使用開發套件。

其他資訊

如需 AWS Lambda 透過 AWS 與 .NET Acover 整合 .NET Aspire 使用 進行開發的資訊，請參閱 。

主題

主題

- 使用註釋撰寫 AWS Lambda 函數

使用註釋撰寫 AWS Lambda 函數

撰寫 Lambda 函數時，有時需要撰寫大量處理常式程式碼和更新 AWS CloudFormation 範本，以及其他任務。Lambda Annotations 是一種架構，可協助減輕 .NET 6 Lambda 函數的這些負擔，從而使編寫 Lambda 的體驗在 C# 中更自然。

使用 Lambda 註釋架構的好處範例是，請考慮下列程式碼片段，這些程式碼片段會新增兩個數字。

沒有 Lambda 註釋

```
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 註釋

```
public class Functions
{
    [LambdaFunction]
    [RestApi("/plus/{x}/{y}")]
    public int Plus(int x, int y)
    {
        return x + y;
    }
}
```

如範例所示，Lambda Annotations 可以消除特定沸騰板程式碼的需求。

如需如何使用架構的詳細資訊以及其他資訊，請參閱下列資源：

- 有關 Lambda 註釋APIs 和屬性的文件的 GitHub README。
- Lambda 註釋的 部落格文章。
- `Amazon.Lambda.Annotations` NuGet 套件。
- GitHub 上的相片資產管理專案。具體而言，請參閱專案 README 中的 PamApiAnnotations 資料夾和 Lambda Annotations 參考。

> **Note**
>
> 上述範例專屬於 的 V3 適用於 .NET 的 AWS SDK。如果您使用範例搭配開發套件的 V4（最新版本），您可能需要根據 中的資訊進行調整遷移至第 4 版。

的高階程式庫和架構 適用於 .NET 的 AWS SDK

下列各節包含不屬於核心 SDK 功能之高階程式庫和架構的相關資訊。這些程式庫和架構使用核心 SDK 功能來建立可簡化特定任務的功能。

如果您是初次使用 適用於 .NET 的 AWS SDK，建議您先查看建立簡單的應用程式主題。它為您提供 SDK 的簡介。

開始之前，請確定您已設定環境並設定專案。也請檢閱 中的資訊使用開發套件。

主題

- AWS 適用於 .NET 的訊息處理架構

- 在 中 AWS 與 .NET ACache 整合 適用於 .NET 的 AWS SDK

AWS 適用於 .NET 的訊息處理架構

適用於 .NET AWS 的訊息處理架構是一種 AWS原生架構，可簡化開發使用 Amazon Simple Queue Service (SQS)、Amazon Simple Notification Service (SNS) 和 Amazon EventBridge 等 AWS 服務的 .NET 訊息處理應用程式。此架構可減少開發人員需要撰寫的樣板程式碼數量，讓您在發佈和使用訊息時專注於商業邏輯。如需架構如何簡化開發的詳細資訊，請參閱部落格文章介紹適用於 .NET AWS 的訊息處理架構 （預覽）。其中第一部分提供示範，顯示使用低階 API 呼叫和使用架構之間的差異。

訊息處理架構支援下列活動和功能：

- 傳送訊息至 SQS 並將事件發佈至 SNS 和 EventBridge。
- 使用長時間執行的輪詢器接收和處理來自 SQS 的訊息，這通常用於背景服務。這包括在處理訊息時管理可見性逾時，以防止其他用戶端處理它。
- 在 AWS Lambda 函數中處理訊息。
- FIFO first-in-first-out) SQS 佇列和 SNS 主題。
- 用於記錄的 OpenTelemetry。

如需這些活動和功能的詳細資訊，請參閱部落格文章的功能一節和下列主題。

開始之前，請確定您已設定環境並設定專案。也請檢閱 中的資訊使用開發套件。

其他資源

- NuGet.org 上的AWS.Messaging套件。
- API 參考。
- GitHub 儲存庫中的 README 檔案位於 https://github.com/aws/aws-dotnet-messaging/
- Microsoft 的 .NET 相依性插入。
- Microsoft 的 .NET Generic Host。

主題

- 開始使用適用於 .NET AWS 的訊息處理架構
- 使用適用於 .NET 的訊息處理架構發佈 AWS 訊息
- 使用適用於 .NET 的訊息處理架構來使用 AWS 訊息

- 將 FIFO 與適用於 .NET AWS 的訊息處理架構搭配使用
- 適用於 .NET AWS 的訊息處理架構的記錄和開啟遙測
- 自訂適用於 .NET AWS 的訊息處理架構
- 適用於 .NET AWS 的訊息處理架構安全性

開始使用適用於 .NET AWS 的訊息處理架構

開始之前，請確定您已設定環境並設定專案。也請檢閱 中的資訊使用開發套件。

本主題提供的資訊可協助您開始使用訊息處理架構。除了先決條件和組態資訊之外，還提供教學課程，說明如何實作常見案例。

先決條件和組態

- 您為應用程式提供的登入資料必須具有其使用的簡訊服務和操作的適當許可。如需詳細資訊，請參閱其各自開發人員指南中的 SQS、SNS 和 EventBridge 的安全主題。另請參閱 GitHub 上討論特定許可的 README 檔案部分。
- 若要使用適用於 .NET AWS 的訊息處理架構，您必須將 `AWS.Messaging` NuGet 套件新增至您的專案。例如：

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

- 框架與 .NET 的相依性注入 (DI) 服務容器整合。您可以在應用程式啟動期間呼叫 AddAWSMessageBus以將其新增至 DI 容器，藉此設定架構。

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

教學課程

本教學課程示範如何使用適用於 .NET AWS 的訊息處理架構。它會建立兩個應用程式：在 API 端點收到請求時將訊息傳送至 Amazon SQS 佇列的 ASP.NET Core Minimal API，以及輪詢這些訊息並加以處理的長時間執行主控台應用程式。

- 本教學中的指示偏好 .NET CLI，但您可以使用 .NET CLI 或 Microsoft Visual Studio 等跨平台工具來執行本教學課程。如需工具的資訊，請參閱 安裝和設定 的工具鏈 適用於 .NET 的 AWS SDK。
- 本教學假設您正在將[default]設定檔用於登入資料。它還假設短期憑證可用，並具有傳送和接收 Amazon SQS 訊息的適當許可。如需詳細資訊，請參閱 適用於 .NET 的 AWS SDK 使用 驗證 AWS和 SQS 的安全主題。

> **Note**
>
> 透過執行本教學課程，您可能需要支付 SQS 訊息費用。

步驟

- 建立 SQS 佇列
- 建立並執行發佈應用程式
- 建立並執行處理應用程式
- 清除

建立 SQS 佇列

本教學課程需要 SQS 佇列來傳送訊息至 並從中接收訊息。您可以針對 AWS CLI 或 使用下列其中一個命令來建立佇列 AWS Tools for PowerShell。請記下傳回的佇列 URL，以便您可以在接下來的架構組態中指定該 URL。

AWS CLI

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

AWS Tools for PowerShell

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

建立並執行發佈應用程式

使用下列程序來建立和執行發佈應用程式。

1. 開啟命令提示字元或終端機。尋找或建立可在其下建立 .NET 專案的作業系統資料夾。
2. 在該資料夾中，執行下列命令以建立 .NET 專案。

```
dotnet new webapi --name Publisher
```

3. 導覽至新專案的資料夾。在適用於 .NET AWS 的訊息處理架構上新增相依性。

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

> **Note**
>
> 如果您使用 AWS IAM Identity Center 進行身分驗證，請務必同時新增 `AWSSDK.SSO`和 `AWSSDK.SSOOIDC`。

4. 使用`Program.cs`下列程式碼取代 中的程式碼。

```
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. 執行下列命令。這應該會使用 Swagger UI 開啟瀏覽器視窗，讓您探索和測試 API。

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

6. 開啟/greeting端點，然後選擇試用。
7. 指定訊息的 senderName和 greeting值，然後選擇執行。這會叫用您的 API，以傳送 SQS 訊息。

建立並執行處理應用程式

使用下列程序來建立和執行處理應用程式。

1. 開啟命令提示字元或終端機。尋找或建立可在其下建立 .NET 專案的作業系統資料夾。
2. 在該資料夾中，執行下列命令以建立 .NET 專案。

```
dotnet new console --name Handler
```

3. 導覽至新專案的資料夾。在適用於 .NET AWS 的訊息處理架構上新增相依性。同時新增`Microsoft.Extensions.Hosting`套件，可讓您透過 .NET Generic Host 設定架構。

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

> **Note**
>
> 如果您使用 AWS IAM Identity Center 進行身分驗證，請務必同時新增 `AWSSDK.SSO`和`AWSSDK.SSOOIDC`。

4. 使用`Program.cs`下列程式碼取代 中的程式碼。

```
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. 執行下列命令。這會啟動長時間執行的輪詢器。

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

啟動後，應用程式很快就會收到本教學課程第一部分所傳送的訊息，並記錄下列訊息：

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

6. 按 Ctrl+C 停止輪詢器。

清除

使用下列其中一個命令，讓 AWS CLI 或 AWS Tools for PowerShell 刪除佇列。

AWS CLI

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

AWS Tools for PowerShell

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

使用適用於 .NET 的訊息處理架構發佈 AWS 訊息

.NET AWS 的訊息處理架構支援發佈一或多個訊息類型、處理一或多個訊息類型，或在相同的應用程式中同時執行兩者。

下列程式碼顯示將不同訊息類型發佈至不同 AWS 服務之應用程式的組態。

```
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");
```

```
});
```

在啟動期間註冊架構後，請將一般 插入IMessagePublisher您的程式碼。呼叫其 PublishAsync 方法來發佈上述設定的任何訊息類型。一般發佈者會根據訊息的類型，決定要將訊息路由到 的目的地。

在下列範例中，ASP.NET MVC 控制器會收到來自使用者ChatMessage的訊息和OrderInfo事件，然後將其分別發佈到 Amazon SQS 和 Amazon SNS。這兩種訊息類型都可以使用上面設定的一般發佈者來發佈。

```
[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();
    }
}
```

為了將訊息路由到適當的處理邏輯，架構會使用稱為訊息類型識別符的中繼資料。根據預設，這是訊息 .NET 類型的全名，包括其組件名稱。如果您同時傳送和處理訊息，如果您跨專案共用訊息物件的定義，則此機制會正常運作。不過，如果訊息是在不同的命名空間中重新定義，或者如果您要與其他架構或程式設計語言交換訊息，則您可能需要覆寫訊息類型識別符。

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

服務特定的發佈者

上述範例使用一般 IMessagePublisher，可根據設定的訊息類型發佈到任何支援的 AWS 服務。此架構也為 Amazon SQS、Amazon SNS 和 Amazon EventBridge 提供服務特定的發佈者。這些特定發佈者公開僅適用於該服務的選項，並且可以使用類型 ISQSPublisher、 ISNSPublisher和 注入IEventBridgePublisher。

例如，傳送訊息至 SQS FIFO 佇列時，您必須設定適當的訊息群組 ID。下列程式碼再次顯示ChatMessage範例，但現在使用 ISQSPublisher來設定 SQS 特定的選項。

```
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();
    }
}
```

您可以IEventBridgePublisher分別使用 和 對 SNS ISNSPublisher和 EventBridge 進行相同的操作。

```
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>
});
```

根據預設，指定類型的訊息會傳送至預先設定的目的地。不過，您可以使用訊息特定的發佈者覆寫單一訊息的目的地。您也可以覆寫用於發佈訊息的基礎 適用於 .NET 的 AWS SDK 用戶端，這在您需要變更角色或憑證的多租用戶應用程式中非常有用，視目的地而定。

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

使用適用於 .NET 的訊息處理架構來使用 AWS 訊息

.NET AWS 的訊息處理架構可讓您使用架構或其中一個訊息服務來取用已發佈的訊息。這些訊息可以透過各種方式使用，其中一些如下所述。

訊息處理常式

若要取用訊息，請使用您要處理的每個訊息類型的`IMessageHandler`界面來實作訊息處理常式。訊息類型和訊息處理常式之間的映射是在專案啟動中設定。

```
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();
```

下列程式碼顯示訊息的範例ChatMessage訊息處理常式。

```
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包含架構使用的中繼資料。其message屬性是訊息類型 （在此案例中為ChatMessage)。

您可以返回 MessageProcessStatus.Success() ，指出訊息已成功處理，架構將從 Amazon SQS 佇列刪除訊息。傳回 時MessageProcessStatus.Failed()，如果已設定，訊息將保留在佇列中，以供再次處理或移至無效字母佇列。

在長時間執行的程序中處理訊息

您可以使用 AddSQSPoller SQS 佇列 URL 呼叫 ，以啟動長時間執行BackgroundService，以持續輪詢佇列並處理訊息。

```
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();
```

設定 SQS 訊息輪詢器

呼叫 SQSMessagePollerOptions時， 可以設定 SQS 訊息輪詢器AddSQSPoller。

- `MaxNumberOfConcurrentMessages` - 佇列中要同時處理的訊息數量上限。預設值為 10。
- `WaitTimeSeconds` - `ReceiveMessage` SQS 呼叫在傳回前等待訊息抵達佇列的持續時間（以秒為單位）。如果訊息可用，呼叫會比 更快傳回`WaitTimeSeconds`。預設值為 20。

訊息可見性逾時處理

SQS 訊息具有可見性逾時期間。當一個取用者開始處理指定的訊息時，它會保留在佇列中，但其他取用者會將其隱藏以避免處理超過一次。如果未在再次顯示訊息之前處理和刪除訊息，另一個消費者可能會嘗試處理相同的訊息。

框架會追蹤並嘗試延長目前正在處理之訊息的可見性逾時。您可以在呼叫`SQSMessagePollerOptions`時，在 上設定此行為`AddSQSPoller`。

- `VisibilityTimeout` - 從後續擷取請求中隱藏接收訊息的持續時間，以秒為單位。預設值為 30。
- `VisibilityTimeoutExtensionThreshold` - 當訊息的可見性逾時在即將到期的這幾秒鐘內時，架構會將可見性逾時（再延長一`VisibilityTimeout`秒鐘）延長。預設值為 5。
- `VisibilityTimeoutExtensionHeartbeatInterval` - 架構會在幾秒內檢查`VisibilityTimeoutExtensionThreshold`過期訊息，然後延長可見性逾時的頻率。預設值為 1。

在下列範例中，架構將每 1 秒檢查一次仍在處理的訊息。對於再次顯示訊息的 5 秒內，框架會自動將每則訊息的可見性逾時延長 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;
});
```

在 AWS Lambda 函數中處理訊息

您可以使用適用於 .NET AWS 的訊息處理架構搭配 SQS 與 Lambda 的整合。這是由`AWS.Messaging.Lambda`套件提供。請參閱其 README 以開始使用。

將 FIFO 與適用於 .NET AWS 的訊息處理架構搭配使用

對於訊息排序和訊息重複資料刪除至關重要的使用案例，.NET AWS 的訊息處理架構支援first-in-first-out(FIFO) Amazon SQS 佇列和 Amazon SNS 主題。

發布

將訊息發佈至 FIFO 佇列或主題時，您必須設定訊息群組 ID，以指定訊息所屬的群組。群組內的訊息會依序處理。您可以在 SQS 特定和 SNS 特定訊息發佈者上設定此項目。

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

訂閱

處理來自 FIFO 佇列的訊息時，框架會依照每次ReceiveMessages呼叫收到訊息的順序，來處理指定訊息群組內的訊息。框架在設定佇列以 結尾時，會自動進入此操作模式.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();
```

適用於 .NET AWS 的訊息處理架構的記錄和開啟遙測

.NET AWS 的訊息處理架構是針對 OpenTelemetry 進行檢測，以記錄架構發佈或處理的每個訊息的追蹤。這是由 AWS.Messaging.Telemetry.OpenTelemetry套件提供。請參閱其 README 以開始使用。

> **Note**
>
> 如需記錄的相關安全性資訊，請參閱 適用於 .NET AWS 的訊息處理架構安全性。

自訂適用於 .NET AWS 的訊息處理架構

.NET AWS 的訊息處理架構會在三個不同的「層」中建置、傳送和處理訊息：

1. 在最外層，架構會建置服務特有的 AWS原生請求或回應。例如，使用 Amazon SQS，它會建置SendMessage請求，並使用 服務定義的Message物件。
2. 在 SQS 請求和回應中，架構會將 MessageBody元素 （或 Message Amazon SNS 或 Detail Amazon EventBridge) 設定為 JSON 格式的 CloudEvent。這包含架構設定的中繼資料，可在處理訊息時存取MessageEnvelope物件。
3. 在最內層，CloudEvent JSON 物件內的 data 屬性包含作為訊息傳送或接收的 .NET 物件的 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>"
}
```

您可以自訂訊息信封的設定和讀取方式：

- "id" 可唯一識別訊息。根據預設，它會設定為新的 GUID，但可以透過實作您自己的 IMessageIdGenerator 並將其插入 DI 容器來覆寫。
- "type" 控制訊息路由到處理常式的方式。根據預設，這會使用與訊息對應的 .NET 類型完整名稱。透過 AddSQSPublisher、 或 將訊息類型映射至目的地時AddSNSPublisher，您可以透過 messageTypeIdentifier 參數覆寫此項目AddEventBridgePublisher。
- "source" 指出哪個系統或伺服器傳送訊息。
  - 如果從 發佈，這會是函數名稱 AWS Lambda，如果在 Amazon ECS 上是叢集名稱和任務 ARN，如果在 Amazon EC2 上是執行個體 ID，則將是備用值 /aws/messaging。

  - 您可以透過 AddMessageSource或在 AddMessageSourceSuffix上覆寫此項目MessageBusBuilder。
- "time" 設定為目前的 UTC DateTime。這可以透過實作您自己的 IDateTimeHandler並將其注入 DI 容器來覆寫。
- "data" 包含作為訊息傳送或接收的 .NET 物件的 JSON 表示法：
  - ConfigureSerializationOptions 上的 MessageBusBuilder可讓您設定序列化和還原序列化訊息時System.Text.Json.JsonSerializerOptions將使用的 。
  - 若要在架構建置後插入其他屬性或轉換訊息信封，您可以在 AddSerializationCallback上透過 實作ISerializationCallback並註冊該屬性MessageBusBuilder。

適用於 .NET AWS 的訊息處理架構安全性

.NET AWS 的訊息處理架構倚賴 適用於 .NET 的 AWS SDK 與 通訊 AWS。如需 中安全性的詳細資訊 適用於 .NET 的 AWS SDK，請參閱 此 AWS 產品或服務的安全性。

基於安全考量，架構不會記錄使用者傳送的資料訊息。如果您想要啟用此功能進行偵錯，則需要在訊息匯流排EnableDataMessageLogging()中呼叫 ，如下所示：

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

如果您發現潛在的安全問題，請參閱安全性政策以取得報告資訊。

在 中 AWS 與 .NET ACache 整合 適用於 .NET 的 AWS SDK

.NET Aspire 是建置雲端應用程式的新方法。特別是，它為要在其中執行、連線和偵錯分散式應用程式元件的本機環境提供協調。為了改善雲端就緒應用程式的內部開發迴圈，已建立與 .NET Aaze 的整合，將 .NET 應用程式連線至 AWS 資源。這些整合可透過 Aspire.Hosting.AWS NuGet 套件取得。

下列 .NET Acsv 整合可供使用：

- 透過 佈建 AWS 資源的能力AWS CloudFormation。此整合用於 .NET AShield AppHost 專案。

  如需詳細資訊，請參閱 部落格文章AWS 與 .NET Aspire 整合。
- 安裝、設定並將 適用於 .NET 的 AWS SDK 連線至 Amazon DynamoDB 本機。此整合用於 .NET AShield AppHost 專案。

如需詳細資訊，請參閱 部落格文章AWS 與 .NET Aspire 整合。

- 為 AWS Lambda函數啟用本機開發環境。此整合用於 .NET AShield AppHost 專案。

  如需詳細資訊，請參閱部落格文章建置和偵錯 .NET Lambda 應用程式與 .NET Acsv （第 1 部分），以及建置和偵錯 .NET Lambda 應用程式與 .NET Acsv （第 2 部分）。

  > **Note**
  >
  > 這是預覽版本之服務的發行前版本文件。內容可能變動。

  由於此功能處於預覽狀態，您將需要選擇加入預覽功能。如需此預覽功能以及如何選擇加入的詳細資訊，請參閱 GitHub 上的開發追蹤器問題。

其他資訊

如需如何使用 Aspire.Hosting.AWS 中可用整合的其他資訊和詳細資訊，請參閱下列資源。

- 部落格文章AWS 與 .NET Aspire 整合。
- 部落格發佈使用 .NET Aspire （第 1 部分）建置和偵錯 .NET Lambda 應用程式，以及使用 .NET ACache （第 2 部分）建置和偵錯 .NET Lambda 應用程式。
- GitHub 上的 integrations-on-dotnet-aspire-for-aws 儲存庫。
- Aspire.Hosting.AWS NuGet 套件的詳細 README。

程式設計 AWS OpsWorks 以使用堆疊和應用程式

> **Warning**
>
> OpsWorks 正在生命週期結束，而且 不接受新客戶。現有客戶在 2024 年 3 月或 5 月之前都不會受到影響，視他們使用的服務而定，屆時該服務將無法使用。為了準備進行此轉換，我們建議現有客戶盡快遷移至其他解決方案。如需詳細資訊，請參閱 OpsWorks 產品頁面。

適用於 .NET 的 AWS SDK 支援 AWS OpsWorks，提供簡單且彈性的方式來建立和管理堆疊和應用程式。透過 OpsWorks，您可以佈建 AWS 資源、管理其組態、將應用程式部署到這些資源，以及監控其運作狀態。如需詳細資訊，請參閱 OpsWorks 產品頁面和 AWS OpsWorks 使用者指南。

API

適用於 .NET 的 AWS SDK 提供 APIs OpsWorks。APIs 可讓您使用https://docs.aws.amazon.com/opsworks/latest/userguide/workingstacks.html堆疊及其層、執行個體和應用程式等 OpsWorks 功能。若要檢視完整的 APIs 集，請參閱 適用於 .NET 的 AWS SDK API 參考 （並捲動至「Amazon.OpsWorks」）。

OpsWorks APIs 由 AWSSDK.OpsWorks NuGet 套件提供。

先決條件

開始之前，請確定您已設定環境並設定專案。也請檢閱 中的資訊使用開發套件。

支援其他 AWS 服務和組態

除了上述各節所述 AWS 的服務之外， 適用於 .NET 的 AWS SDK 還支援服務。如需所有支援服務 APIs 的相關資訊，請參閱 適用於 .NET 的 AWS SDK API 參考。

除了個別 AWS 服務的命名空間之外， 適用於 .NET 的 AWS SDK 還提供下列 APIs：

區域圖	描述	資源
AWS 支援	以程式設計方式存取 AWS 支援案例和 Trusted Advisor 功能。	請參閱 Amazon.AWSSupport 和 Amazon.AWSSupport.Model。
一般	協助程式和列舉。	請參閱 Amazon 和 Amazon.Util。

適用於 .NET 的 SDK (v4) 程式碼範例

本主題中的程式碼範例說明如何搭配 適用於 .NET 的 AWS SDK (v4) 使用 AWS。

基本概念是程式碼範例，這些範例說明如何在服務內執行基本操作。

Actions 是大型程式的程式碼摘錄，必須在內容中執行。雖然動作會告訴您如何呼叫個別服務函數，但您可以在其相關情境中查看內容中的動作。

案例是向您展示如何呼叫服務中的多個函數或與其他 AWS 服務組合來完成特定任務的程式碼範例。

有些服務包含其他範例類別，示範如何利用服務特定的程式庫或函數。

服務

- 使用 適用於 .NET 的 SDK (v4) 的 Aurora 範例
- 使用 適用於 .NET 的 SDK (v4) 的 Auto Scaling 範例
- 使用 適用於 .NET 的 SDK (v4) 的 Amazon Bedrock 範例
- 使用 適用於 .NET 的 SDK (v4) 的 Amazon Bedrock 執行期範例
- AWS CloudFormation 使用 適用於 .NET 的 SDK (v4) 的範例
- CloudWatch 範例使用 適用於 .NET 的 SDK (v4)
- 使用 適用於 .NET 的 SDK (v4) 的 Amazon Cognito 身分提供者範例
- AWS Control Tower 使用 適用於 .NET 的 SDK (v4) 的範例
- 使用 適用於 .NET 的 SDK (v4) 的 DynamoDB 範例
- 使用 適用於 .NET 的 SDK (v4) 的 Amazon EC2 範例
- 使用 適用於 .NET 的 SDK (v4) 的 Amazon ECS 範例
- 使用 適用於 .NET 的 SDK (v4) 的 Amazon S3 範例

使用 適用於 .NET 的 SDK (v4) 的 Aurora 範例

下列程式碼範例示範如何使用 適用於 .NET 的 AWS SDK (v4) 搭配 Aurora 來執行動作和實作常見案例。

基本概念是程式碼範例，這些範例說明如何在服務內執行基本操作。

Actions 是大型程式的程式碼摘錄，必須在內容中執行。雖然動作會告訴您如何呼叫個別服務函數，但您可以在其相關情境中查看內容中的動作。

每個範例都包含完整原始程式碼的連結，您可以在其中找到如何在內容中設定和執行程式碼的指示。

開始使用

Hello Aurora

下列程式碼範例示範如何開始使用 Aurora。

適用於 .NET 的 SDK (v4)

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

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

- 如需 API 詳細資訊，請參閱《適用於 .NET 的 AWS SDK API 參考》中的 DescribeDBClusters。

主題

- 基本概念
- 動作

基本概念

了解基本概念

以下程式碼範例顯示做法：

- 建立自訂 Aurora 資料庫叢集參數群組並設定參數值。
- 建立使用該參數群組的資料庫叢集。
- 建立包含該資料庫的資料庫執行個體。
- 拍攝該資料庫叢集的快照，並清理資源。

適用於 .NET 的 SDK (v4)

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

在命令提示中執行互動式案例。

```
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;
    }
```

案例呼叫用以管理 Aurora 動作的包裝函式方式。

```
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;
    }
}
```

- 如需 API 詳細資訊，請參閱《適用於 .NET 的 AWS SDK API 參考》中的下列主題。
  - CreateDBCluster
  - CreateDBClusterParameterGroup

- CreateDBClusterSnapshot
- CreateDBInstance
- DeleteDBCluster
- DeleteDBClusterParameterGroup
- DeleteDBInstance
- DescribeDBClusterParameterGroups
- DescribeDBClusterParameters
- DescribeDBClusterSnapshots
- DescribeDBClusters
- DescribeDBEngineVersions
- DescribeDBInstances
- DescribeOrderableDBInstanceOptions
- ModifyDBClusterParameterGroup

動作

`CreateDBCluster`

以下程式碼範例顯示如何使用 `CreateDBCluster`。

適用於 .NET 的 SDK (v4)

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
/// <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;
    }
```

- 如需 API 詳細資訊，請參閱《適用於 .NET 的 AWS SDK API 參考》中的 CreateDBCluster。

`CreateDBClusterParameterGroup`

以下程式碼範例顯示如何使用 `CreateDBClusterParameterGroup`。

適用於 .NET 的 SDK (v4)

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
    /// <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;
    }
```

- 如需 API 詳細資訊，請參閱《適用於 .NET 的 AWS SDK API 參考》中的 CreateDBClusterParameterGroup。

`CreateDBClusterSnapshot`

以下程式碼範例顯示如何使用 `CreateDBClusterSnapshot`。

適用於 .NET 的 SDK (v4)

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
    /// <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;
    }
```

- 如需 API 詳細資訊，請參閱《適用於 .NET 的 AWS SDK API 參考》中的 CreateDBClusterSnapshot。

`CreateDBInstance`

以下程式碼範例顯示如何使用 `CreateDBInstance`。

適用於 .NET 的 SDK (v4)

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
    /// <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;
    }
```

- 如需 API 詳細資訊，請參閱《適用於 .NET 的 AWS SDK API 參考》中的 CreateDBInstance。

`DeleteDBCluster`

以下程式碼範例顯示如何使用 `DeleteDBCluster`。

適用於 .NET 的 SDK (v4)

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
    /// <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;
    }
```

- 如需 API 詳細資訊，請參閱《適用於 .NET 的 AWS SDK API 參考》中的 DeleteDBCluster。

`DeleteDBClusterParameterGroup`

以下程式碼範例顯示如何使用 `DeleteDBClusterParameterGroup`。

適用於 .NET 的 SDK (v4)

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
    /// <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;
    }
```

- 如需 API 詳細資訊，請參閱《適用於 .NET 的 AWS SDK API 參考》中的 DeleteDBClusterParameterGroup。

`DeleteDBInstance`

以下程式碼範例顯示如何使用 `DeleteDBInstance`。

適用於 .NET 的 SDK (v4)

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
    /// <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;
```

```
    }
```

- 如需 API 詳細資訊，請參閱《適用於 .NET 的 AWS SDK API 參考》中的 DeleteDBInstance。

`DescribeDBClusterParameterGroups`

以下程式碼範例顯示如何使用 `DescribeDBClusterParameterGroups`。

適用於 .NET 的 SDK (v4)

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
    /// <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();
    }
```

- 如需 API 詳細資訊，請參閱《適用於 .NET 的 AWS SDK API 參考》中的 DescribeDBClusterParameterGroups。

`DescribeDBClusterParameters`

以下程式碼範例顯示如何使用 `DescribeDBClusterParameters`。

適用於 .NET 的 SDK (v4)

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
    /// <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;
    }
```

- 如需 API 詳細資訊，請參閱《適用於 .NET 的 AWS SDK API 參考》中的 DescribeDBClusterParameters。

`DescribeDBClusterSnapshots`

以下程式碼範例顯示如何使用 `DescribeDBClusterSnapshots`。

適用於 .NET 的 SDK (v4)

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
   /// <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;
   }
```

- 如需 API 詳細資訊，請參閱《適用於 .NET 的 AWS SDK API 參考》中的 DescribeDBClusterSnapshots。

DescribeDBClusters

以下程式碼範例顯示如何使用 DescribeDBClusters。

適用於 .NET 的 SDK (v4)

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
    /// <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;
    }
```

- 如需 API 詳細資訊，請參閱《適用於 .NET 的 AWS SDK API 參考》中的 DescribeDBClusters。

DescribeDBEngineVersions

以下程式碼範例顯示如何使用 DescribeDBEngineVersions。

適用於 .NET 的 SDK (v4)

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
    /// <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;
    }
```

- 如需 API 詳細資訊，請參閱《適用於 .NET 的 AWS SDK API 參考》中的 DescribeDBEngineVersions。

DescribeDBInstances

以下程式碼範例顯示如何使用 DescribeDBInstances。

適用於 .NET 的 SDK (v4)

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
    /// <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;
    }
```

- 如需 API 詳細資訊，請參閱《適用於 .NET 的 AWS SDK API 參考》中的 DescribeDBInstances。

`DescribeOrderableDBInstanceOptions`

以下程式碼範例顯示如何使用 `DescribeOrderableDBInstanceOptions`。

適用於 .NET 的 SDK (v4)

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
    /// <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;
    }
```

- 如需 API 詳細資訊，請參閱《適用於 .NET 的 AWS SDK API 參考》中的 DescribeOrderableDBInstanceOptions。

ModifyDBClusterParameterGroup

以下程式碼範例顯示如何使用 ModifyDBClusterParameterGroup。

適用於 .NET 的 SDK (v4)

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
    /// <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;
    }
```

- 如需 API 詳細資訊，請參閱《適用於 .NET 的 AWS SDK API 參考》中的 ModifyDBClusterParameterGroup。

使用 適用於 .NET 的 SDK (v4) 的 Auto Scaling 範例

下列程式碼範例示範如何使用 適用於 .NET 的 AWS SDK (v4) 搭配 Auto Scaling 來執行動作和實作常見案例。

基本概念是程式碼範例，這些範例說明如何在服務內執行基本操作。

Actions 是大型程式的程式碼摘錄，必須在內容中執行。雖然動作會告訴您如何呼叫個別服務函數，但您可以在其相關情境中查看內容中的動作。

每個範例都包含完整原始程式碼的連結，您可以在其中找到如何在內容中設定和執行程式碼的指示。

開始使用

Hello Auto Scaling

下列程式碼範例示範如何開始使用 Auto Scaling。

適用於 .NET 的 SDK (v4)

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
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
        });

    }
}
```

- 如需 API 詳細資訊，請參閱適用於 .NET 的 AWS SDK 《 API 參考》中的 DescribeAutoScalingGroups。

主題

- 基本概念
- 動作

基本概念

了解基本概念

以下程式碼範例顯示做法：

- 使用啟動範本和可用區域建立 Amazon EC2 Auto Scaling 群組，並取得執行中執行個體的相關資訊。
- 啟用 Amazon CloudWatch 指標集合。
- 更新群組所需的容量，並等待執行個體啟動。
- 終止 群組中的執行個體。
- 列出為了回應使用者請求和容量變更而發生的擴展活動。
- 取得 CloudWatch 指標的統計資料，然後清除資源。

適用於 .NET 的 SDK (v4)

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
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();
    }
```

```
}
```

定義案例呼叫的函數，以管理啟動範本和指標。這些函數會包裝 Auto Scaling、Amazon EC2 和 CloudWatch 動作。

```
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;
    }

}
```

- 如需 API 詳細資訊，請參閱《適用於 .NET 的 AWS SDK API 參考》中的下列主題。
  - CreateAutoScalingGroup
  - DeleteAutoScalingGroup
  - DescribeAutoScalingGroups
  - DescribeAutoScalingInstances
  - DescribeScalingActivities
  - DisableMetricsCollection
  - EnableMetricsCollection
  - SetDesiredCapacity
  - TerminateInstanceInAutoScalingGroup
  - UpdateAutoScalingGroup

動作

`CreateAutoScalingGroup`

以下程式碼範例顯示如何使用 `CreateAutoScalingGroup`。

適用於 .NET 的 SDK (v4)

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
    /// <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;
        }
    }
```

- 如需 API 詳細資訊，請參閱《適用於 .NET 的 AWS SDK API 參考》中的 CreateAutoScalingGroup。

`DescribeAutoScalingGroups`

以下程式碼範例顯示如何使用 `DescribeAutoScalingGroups`。

適用於 .NET 的 SDK (v4)

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
    /// <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;
}
```

- 如需 API 詳細資訊，請參閱適用於 .NET 的 AWS SDK 《API 參考》中的 DescribeAutoScalingGroups。

`DescribeAutoScalingInstances`

以下程式碼範例顯示如何使用 `DescribeAutoScalingInstances`。

適用於 .NET 的 SDK (v4)

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
/// <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;
    }
```

- 如需 API 詳細資訊，請參閱適用於 .NET 的 AWS SDK 《 API 參考》中的 DescribeAutoScalingInstances。

`DescribeScalingActivities`

以下程式碼範例顯示如何使用 `DescribeScalingActivities`。

適用於 .NET 的 SDK (v4)

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
    /// <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;
    }
```

- 如需 API 詳細資訊，請參閱適用於 .NET 的 AWS SDK 《API 參考》中的 DescribeScalingActivities。

`DisableMetricsCollection`

以下程式碼範例顯示如何使用 `DisableMetricsCollection`。

適用於 .NET 的 SDK (v4)

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
    /// <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;
    }
```

- 如需 API 詳細資訊，請參閱適用於 .NET 的 AWS SDK 《 API 參考》中的 DisableMetricsCollection。

`EnableMetricsCollection`

以下程式碼範例顯示如何使用 `EnableMetricsCollection`。

適用於 .NET 的 SDK (v4)

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
    /// <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;
    }
```

- 如需 API 詳細資訊，請參閱適用於 .NET 的 AWS SDK 《 API 參考》中的 EnableMetricsCollection。

SetDesiredCapacity

以下程式碼範例顯示如何使用 SetDesiredCapacity。

適用於 .NET 的 SDK (v4)

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
    /// <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;
    }
```

- 如需 API 詳細資訊，請參閱適用於 .NET 的 AWS SDK 《 API 參考》中的 SetDesiredCapacity。

`TerminateInstanceInAutoScalingGroup`

以下程式碼範例顯示如何使用 TerminateInstanceInAutoScalingGroup。

適用於 .NET 的 SDK (v4)

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
    /// <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;
```

```
    }
```

- 如需 API 詳細資訊，請參閱適用於 .NET 的 AWS SDK 《 API 參考》中的 TerminateInstanceInAutoScalingGroup。

`UpdateAutoScalingGroup`

以下程式碼範例顯示如何使用 UpdateAutoScalingGroup。

適用於 .NET 的 SDK (v4)

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
    /// <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;
        }
    }
```

- 如需 API 詳細資訊，請參閱《適用於 .NET 的 AWS SDK API 參考》中的 UpdateAutoScalingGroup。

使用 適用於 .NET 的 SDK (v4) 的 Amazon Bedrock 範例

下列程式碼範例示範如何使用 適用於 .NET 的 AWS SDK (v4) 搭配 Amazon Bedrock 來執行動作和實作常見案例。

Actions 是大型程式的程式碼摘錄，必須在內容中執行。雖然動作會告訴您如何呼叫個別服務函數，但您可以在其相關情境中查看內容中的動作。

每個範例都包含完整原始程式碼的連結，您可以在其中找到如何在內容中設定和執行程式碼的指示。

開始使用

您好，Amazon Bedrock

下列程式碼範例示範如何開始使用 Amazon Bedrock。

適用於 .NET 的 SDK (v4)

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

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

- 如需 API 詳細資訊，請參閱適用於 .NET 的 AWS SDK 《 API 參考》中的 ListFoundationModels。

主題

- 動作

動作

`ListFoundationModels`

以下程式碼範例顯示如何使用 `ListFoundationModels`。

適用於 .NET 的 SDK (v4)

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

列出可用的 Bedrock 基礎模型。

```
   /// <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);
        }
    }
```

- 如需 API 詳細資訊，請參閱適用於 .NET 的 AWS SDK 《API 參考》中的 ListFoundationModels。

使用 適用於 .NET 的 SDK (v4) 的 Amazon Bedrock 執行期範例

下列程式碼範例示範如何使用 適用於 .NET 的 AWS SDK (v4) 搭配 Amazon Bedrock 執行期來執行動作和實作常見案例。

每個範例都包含完整原始程式碼的連結，您可以在其中找到如何在內容中設定和執行程式碼的指示。

主題

- Amazon Titan 文字
- Anthropic Claude
- Cohere Command
- Meta Llama
- 混合式 AI

Amazon Titan 文字

Converse

下列程式碼範例示範如何使用 Bedrock 的 Converse API，將文字訊息傳送至 Amazon Titan Text。

適用於 .NET 的 SDK (v4)

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

使用 Bedrock 的 Converse API，將文字訊息傳送至 Amazon Titan Text。

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

using System;
using System.Collections.Generic;
using Amazon;
using Amazon.BedrockRuntime;
using Amazon.BedrockRuntime.Model;

// Create a Bedrock Runtime client in the AWS Region you want to use.
var client = new AmazonBedrockRuntimeClient(RegionEndpoint.USEast1);

// Set the model ID, e.g., Titan Text Premier.
var modelId = "amazon.titan-text-premier-v1:0";

// Define the user message.
var userMessage = "Describe the purpose of a 'hello world' program in one line.";

// Create a request with the model ID, the user message, and an inference
 configuration.
var request = new ConverseRequest
{
    ModelId = modelId,
    Messages = new List<Message>
    {
        new Message
        {
```

```
            Role = ConversationRole.User,
            Content = new List<ContentBlock> { new ContentBlock { Text =
 userMessage } }
        }
    },
    InferenceConfig = new InferenceConfiguration()
    {
        MaxTokens = 512,
        Temperature = 0.5F,
        TopP = 0.9F
    }
};

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

    // Extract and print the response text.
    string responseText = response?.Output?.Message?.Content?[0]?.Text ?? "";
    Console.WriteLine(responseText);
}
catch (AmazonBedrockRuntimeException e)
{
    Console.WriteLine($"ERROR: Can't invoke '{modelId}'. Reason: {e.Message}");
    throw;
}
```

- 如需 API 詳細資訊，請參閱適用於 .NET 的 AWS SDK 《API 參考》中的 Converse。

ConverseStream

下列程式碼範例示範如何使用 Bedrock 的 Converse API 將文字訊息傳送至 Amazon Titan Text，並即時處理回應串流。

適用於 .NET 的 SDK (v4)

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

使用 Bedrock 的 Converse API 將文字訊息傳送至 Amazon Titan Text，並即時處理回應串流。

```
// Use the Converse API to send a text message to Amazon Titan Text
// and print the response stream.

using System;
using System.Collections.Generic;
using System.Linq;
using Amazon;
using Amazon.BedrockRuntime;
using Amazon.BedrockRuntime.Model;

// Create a Bedrock Runtime client in the AWS Region you want to use.
var client = new AmazonBedrockRuntimeClient(RegionEndpoint.USEast1);

// Set the model ID, e.g., Titan Text Premier.
var modelId = "amazon.titan-text-premier-v1:0";

// Define the user message.
var userMessage = "Describe the purpose of a 'hello world' program in one line.";

// Create a request with the model ID, the user message, and an inference
 configuration.
var request = new ConverseStreamRequest
{
    ModelId = modelId,
    Messages = new List<Message>
    {
        new Message
        {
            Role = ConversationRole.User,
            Content = new List<ContentBlock> { new ContentBlock { Text =
 userMessage } }
        }
    },
```

```
    InferenceConfig = new InferenceConfiguration()
    {
        MaxTokens = 512,
        Temperature = 0.5F,
        TopP = 0.9F
    }
};

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

    // Extract and print the streamed response text in real-time.
    foreach (var chunk in response.Stream.AsEnumerable())
    {
        if (chunk is ContentBlockDeltaEvent)
        {
            Console.Write((chunk as ContentBlockDeltaEvent).Delta.Text);
        }
    }
}
catch (AmazonBedrockRuntimeException e)
{
    Console.WriteLine($"ERROR: Can't invoke '{modelId}'. Reason: {e.Message}");
    throw;
}
```

- 如需 API 詳細資訊，請參閱適用於 .NET 的 AWS SDK 《API 參考》中的 ConverseStream。

InvokeModel

下列程式碼範例示範如何使用調用模型 API，將文字訊息傳送至 Amazon Titan Text。

適用於 .NET 的 SDK (v4)

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

使用調用模型 API 來傳送文字訊息。

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

using System;
using System.IO;
using System.Text.Json;
using System.Text.Json.Nodes;
using Amazon;
using Amazon.BedrockRuntime;
using Amazon.BedrockRuntime.Model;

// Create a Bedrock Runtime client in the AWS Region you want to use.
var client = new AmazonBedrockRuntimeClient(RegionEndpoint.USEast1);

// Set the model ID, e.g., Titan Text Premier.
var modelId = "amazon.titan-text-premier-v1:0";

// Define the user message.
var userMessage = "Describe the purpose of a 'hello world' program in one line.";

//Format the request payload using the model's native structure.
var nativeRequest = JsonSerializer.Serialize(new
{
    inputText = userMessage,
    textGenerationConfig = new
    {
        maxTokenCount = 512,
        temperature = 0.5
    }
});

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

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

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

    // Extract and print the response text.
    var responseText = modelResponse["results"]?[0]?["outputText"] ?? "";
    Console.WriteLine(responseText);
}
catch (AmazonBedrockRuntimeException e)
{
    Console.WriteLine($"ERROR: Can't invoke '{modelId}'. Reason: {e.Message}");
    throw;
}
```

- 如需 API 詳細資訊，請參閱適用於 .NET 的 AWS SDK 《API 參考》中的 InvokeModel。

Anthropic Claude

Converse

下列程式碼範例示範如何使用 Bedrock 的 Converse API，將文字訊息傳送至 Anthropic Claude。

適用於 .NET 的 SDK (v4)

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

使用 Bedrock 的 Converse API，將文字訊息傳送至 Anthropic Claude。

```
// 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;
```

```
}
```

- 如需 API 詳細資訊，請參閱適用於 .NET 的 AWS SDK 《API 參考》中的 Converse。

ConverseStream

下列程式碼範例示範如何使用 Bedrock 的 Converse API 將文字訊息傳送至 Anthropic Claude，並即時處理回應串流。

適用於 .NET 的 SDK (v4)

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

使用 Bedrock 的 Converse API 將文字訊息傳送至 Anthropic Claude，並即時處理回應串流。

```
// 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;
}
```

- 如需 API 詳細資訊，請參閱《 適用於 .NET 的 AWS SDK API 參考》中的 ConverseStream。

InvokeModel

下列程式碼範例示範如何使用調用模型 API，將文字訊息傳送至 Anthropic Claude。

適用於 .NET 的 SDK (v4)

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

使用調用模型 API 來傳送文字訊息。

```
// Use the native inference API to send a text message to Anthropic Claude.

using System;
using System.IO;
using System.Text.Json;
using System.Text.Json.Nodes;
using Amazon;
using Amazon.BedrockRuntime;
using Amazon.BedrockRuntime.Model;

// Create a Bedrock Runtime client in the AWS Region you want to use.
var client = new AmazonBedrockRuntimeClient(RegionEndpoint.USEast1);

// Set the model ID, e.g., Claude 3 Haiku.
var modelId = "anthropic.claude-3-haiku-20240307-v1:0";

// Define the user message.
var userMessage = "Describe the purpose of a 'hello world' program in one line.";

//Format the request payload using the model's native structure.
var nativeRequest = JsonSerializer.Serialize(new
{
    anthropic_version = "bedrock-2023-05-31",
    max_tokens = 512,
    temperature = 0.5,
    messages = new[]
    {
        new { role = "user", content = userMessage }
    }
```

```
});

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

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

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

    // Extract and print the response text.
    var responseText = modelResponse["content"]?[0]?["text"] ?? "";
    Console.WriteLine(responseText);
}
catch (AmazonBedrockRuntimeException e)
{
    Console.WriteLine($"ERROR: Can't invoke '{modelId}'. Reason: {e.Message}");
    throw;
}
```

- 如需 API 詳細資訊，請參閱適用於 .NET 的 AWS SDK 《API 參考》中的 InvokeModel。

Cohere Command

Converse

下列程式碼範例示範如何使用 Bedrock 的 Converse API，將文字訊息傳送至 Cohere Command。

適用於 .NET 的 SDK (v4)

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

使用 Bedrock 的 Converse API，將文字訊息傳送至 Cohere Command。

```
// 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;
}
```

- 如需 API 詳細資訊，請參閱適用於 .NET 的 AWS SDK 《 API 參考》中的 Converse。

ConverseStream

下列程式碼範例示範如何使用 Bedrock 的 Converse API 將文字訊息傳送至 Cohere Command，並即時處理回應串流。

適用於 .NET 的 SDK (v4)

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

使用 Bedrock 的 Converse API 將文字訊息傳送至 Cohere Command，並即時處理回應串流。

```
// 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;
}
```

- 如需 API 詳細資訊，請參閱《 適用於 .NET 的 AWS SDK API 參考》中的 ConverseStream。

Meta Llama

Converse

下列程式碼範例示範如何使用 Bedrock 的 Converse API，將文字訊息傳送至 Meta Llama。

適用於 .NET 的 SDK (v4)

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

使用 Bedrock 的 Converse API，將文字訊息傳送至 Meta Llama。

```
// 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;
}
```

- 如需 API 詳細資訊，請參閱《 適用於 .NET 的 AWS SDK API 參考》中的 Converse。

ConverseStream

下列程式碼範例示範如何使用 Bedrock 的 Converse API 將文字訊息傳送至 Meta Llama，並即時處理回應串流。

適用於 .NET 的 SDK (v4)

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

使用 Bedrock 的 Converse API 將文字訊息傳送至 Meta Llama，並即時處理回應串流。

```
// 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;
}
```

- 如需 API 詳細資訊，請參閱適用於 .NET 的 AWS SDK 《 API 參考》中的 ConverseStream。

混合式 AI

Converse

下列程式碼範例示範如何使用 Bedrock 的 Converse API 將文字訊息傳送至 Mistral。

適用於 .NET 的 SDK (v4)

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

使用 Bedrock 的 Converse API，將文字訊息傳送至 Mistral。

```
// 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;
}
```

- 如需 API 詳細資訊，請參閱適用於 .NET 的 AWS SDK 《API 參考》中的 Converse。

ConverseStream

下列程式碼範例示範如何使用 Bedrock 的 Converse API 將文字訊息傳送至 Mistral，並即時處理回應串流。

適用於 .NET 的 SDK (v4)

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

使用 Bedrock 的 Converse API 將文字訊息傳送至 Mistral，並即時處理回應串流。

```
// 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;
}
```

- 如需 API 詳細資訊，請參閱適用於 .NET 的 AWS SDK 《 API 參考》中的 ConverseStream。

AWS CloudFormation 使用 適用於 .NET 的 SDK (v4) 的範例

下列程式碼範例示範如何使用 適用於 .NET 的 AWS SDK (v4) 搭配 來執行動作和實作常見案例 AWS CloudFormation。

每個範例都包含完整原始程式碼的連結，您可以在其中找到如何在內容中設定和執行程式碼的指示。

開始使用

您好 AWS CloudFormation

下列程式碼範例說明如何開始使用 AWS CloudFormation。

適用於 .NET 的 SDK (v4)

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
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;
        }
    }
```

- 如需 API 詳細資訊，請參閱適用於 .NET 的 AWS SDK 《 API 參考》中的 DescribeStackResources。

CloudWatch 範例使用 適用於 .NET 的 SDK (v4)

下列程式碼範例示範如何使用 適用於 .NET 的 AWS SDK (v4) 搭配 CloudWatch 來執行動作和實作常見案例。

每個範例都包含完整原始程式碼的連結，您可以在其中找到如何在內容中設定和執行程式碼的指示。

開始使用

Hello CloudWatch

下列程式碼範例示範如何開始使用 CloudWatch。

適用於 .NET 的 SDK (v4)

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
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();
            }
        }
    }
}
```

- 如需 API 詳細資訊，請參閱 適用於 .NET 的 AWS SDK API 參考中的 ListMetrics。

使用 適用於 .NET 的 SDK (v4) 的 Amazon Cognito 身分提供者範例

下列程式碼範例示範如何使用 適用於 .NET 的 AWS SDK (v4) 搭配 Amazon Cognito 身分提供者來執行動作和實作常見案例。

Actions 是大型程式的程式碼摘錄，必須在內容中執行。雖然動作會告訴您如何呼叫個別服務函數，但您可以在其相關情境中查看內容中的動作。

每個範例都包含完整原始程式碼的連結，您可以在其中找到如何在內容中設定和執行程式碼的指示。

主題

- 動作

動作

`ListUserPools`

以下程式碼範例顯示如何使用 `ListUserPools`。

適用於 .NET 的 SDK (v4)

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
    /// <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;
    }
```

- 如需 API 詳細資訊，請參閱 適用於 .NET 的 AWS SDK API 參考中的 ListUserPools。

AWS Control Tower 使用 適用於 .NET 的 SDK (v4) 的範例

下列程式碼範例示範如何使用 適用於 .NET 的 AWS SDK (v4) 搭配 來執行動作和實作常見案例 AWS Control Tower。

基本概念是程式碼範例，這些範例說明如何在服務內執行基本操作。

Actions 是大型程式的程式碼摘錄，必須在內容中執行。雖然動作會告訴您如何呼叫個別服務函數，但您可以在其相關情境中查看內容中的動作。

每個範例都包含完整原始程式碼的連結，您可以在其中找到如何在內容中設定和執行程式碼的指示。

開始使用

您好 AWS Control Tower

下列程式碼範例示範如何開始使用 AWS Control Tower。

適用於 .NET 的 SDK (v4)

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

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

- 如需 API 詳細資訊，請參閱適用於 .NET 的 AWS SDK 《API 參考》中的 ListBaselines。

主題

- 基本概念
- 動作

基本概念

了解基本概念

以下程式碼範例顯示做法：

- 列出登陸區域。
- 列出、啟用、取得、重設和停用基準。
- 列出、啟用、取得和停用控制項。

適用於 .NET 的 SDK (v4)

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

執行示範 AWS Control Tower 功能的互動式案例。

```
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, "4.0", 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;
                                    break;
                                }
                            }

                            alreadyEnabled = true;
                            Console.WriteLine("No change, the selected baseline was
already enabled.");
                        }

                        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, "4.0",
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;
    }
}
```

案例呼叫用以管理 Aurora 動作的包裝函式方式。

```
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 if already enabled.</returns>
    public async Task<string?> EnableBaselineAsync(string targetIdentifier, string
 baselineIdentifier, string baselineVersion, string identityCenterBaseline)
    {
        try
        {
            var parameters = new List<EnabledBaselineParameter>
            {
                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) when (ex.Message.Contains("already enabled"))
        {
            Console.WriteLine("Baseline is already enabled for this target");
            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;
        }
    }

}
```

- 如需 API 詳細資訊，請參閱《適用於 .NET 的 AWS SDK API 參考》中的下列主題。

- CreateLandingZone
- DeleteLandingZone
- DisableBaseline
- DisableControl
- EnableBaseline
- EnableControl
- GetControlOperation
- GetLandingZoneOperation
- ListBaselines
- ListEnabledBaselines
- ListEnabledControls
- ListLandingZones
- ResetEnabledBaseline

動作

`DisableBaseline`

以下程式碼範例顯示如何使用 `DisableBaseline`。

適用於 .NET 的 SDK (v4)

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
    /// <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;
       }
   }
```

- 如需 API 詳細資訊，請參閱適用於 .NET 的 AWS SDK 《 API 參考》中的 DisableBaseline。

`DisableControl`

以下程式碼範例顯示如何使用 `DisableControl`。

適用於 .NET 的 SDK (v4)

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
   /// <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;
        }
    }
```

- 如需 API 詳細資訊，請參閱適用於 .NET 的 AWS SDK 《API 參考》中的 DisableControl。

`EnableBaseline`

以下程式碼範例顯示如何使用 `EnableBaseline`。

適用於 .NET 的 SDK (v4)

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
    /// <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 if already enabled.</returns>
    public async Task<string?> EnableBaselineAsync(string targetIdentifier, string
 baselineIdentifier, string baselineVersion, string identityCenterBaseline)
    {
        try
        {
            var parameters = new List<EnabledBaselineParameter>
            {
                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) when (ex.Message.Contains("already enabled"))
        {
            Console.WriteLine("Baseline is already enabled for this target");
            return null;
        }
        catch (AmazonControlTowerException ex)
        {
            Console.WriteLine($"Couldn't enable baseline. Here's why:
{ex.ErrorCode}: {ex.Message}");
            throw;
        }
    }
```

- 如需 API 詳細資訊，請參閱適用於 .NET 的 AWS SDK 《API 參考》中的 EnableBaseline。

`EnableControl`

以下程式碼範例顯示如何使用 `EnableControl`。

適用於 .NET 的 SDK (v4)

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
    /// <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;
        }
    }
```

- 如需 API 詳細資訊，請參閱適用於 .NET 的 AWS SDK 《API 參考》中的 EnableControl。

GetBaselineOperation

以下程式碼範例顯示如何使用 GetBaselineOperation。

適用於 .NET 的 SDK (v4)

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
    /// <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;
        }
    }
```

- 如需 API 詳細資訊，請參閱適用於 .NET 的 AWS SDK 《 API 參考》中的 GetBaselineOperation。

`GetControlOperation`

以下程式碼範例顯示如何使用 `GetControlOperation`。

適用於 .NET 的 SDK (v4)

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
    /// <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;
        }
    }
```

- 如需 API 詳細資訊，請參閱適用於 .NET 的 AWS SDK 《API 參考》中的 GetControlOperation。

`ListBaselines`

以下程式碼範例顯示如何使用 `ListBaselines`。

適用於 .NET 的 SDK (v4)

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
    /// <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;
        }
    }
```

- 如需 API 詳細資訊，請參閱適用於 .NET 的 AWS SDK 《API 參考》中的 ListBaselines。

`ListEnabledBaselines`

以下程式碼範例顯示如何使用 `ListEnabledBaselines`。

適用於 .NET 的 SDK (v4)

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
    /// <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;
        }
    }
```

- 如需 API 詳細資訊，請參閱適用於 .NET 的 AWS SDK 《 API 參考》中的 ListEnabledBaselines。

ListEnabledControls

以下程式碼範例顯示如何使用 ListEnabledControls。

適用於 .NET 的 SDK (v4)

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
    /// <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;
        }
    }
```

- 如需 API 詳細資訊，請參閱適用於 .NET 的 AWS SDK 《API 參考》中的 ListEnabledControls。

`ListLandingZones`

以下程式碼範例顯示如何使用 `ListLandingZones`。

適用於 .NET 的 SDK (v4)

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
    /// <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;
        }
    }
```

- 如需 API 詳細資訊，請參閱適用於 .NET 的 AWS SDK 《API 參考》中的 ListLandingZones。

`ResetEnabledBaseline`

以下程式碼範例顯示如何使用 `ResetEnabledBaseline`。

適用於 .NET 的 SDK (v4)

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
    /// <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;
        }
    }
```

- 如需 API 詳細資訊，請參閱適用於 .NET 的 AWS SDK 《 API 參考》中的 ResetEnabledBaseline。

使用 適用於 .NET 的 SDK (v4) 的 DynamoDB 範例

下列程式碼範例示範如何使用 適用於 .NET 的 AWS SDK (v4) 搭配 DynamoDB 來執行動作和實作常見案例。

基本概念是程式碼範例，這些範例說明如何在服務內執行基本操作。

Actions 是大型程式的程式碼摘錄，必須在內容中執行。雖然動作會告訴您如何呼叫個別服務函數，但您可以在其相關情境中查看內容中的動作。

每個範例都包含完整原始程式碼的連結，您可以在其中找到如何在內容中設定和執行程式碼的指示。

開始使用

Hello DynamoDB

下列程式碼範例示範如何開始使用 DynamoDB。

適用於 .NET 的 SDK (v4)

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

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

- 如需 API 的詳細資訊，請參閱 適用於 .NET 的 AWS SDK API 參考中的 ListTables。

主題

- 基本概念
- 動作

基本概念

了解基本概念

以下程式碼範例顯示做法：

- 建立可存放電影資料的資料表。
- 放入、取得和更新資料表中的單個電影。
- 將影片資料從範例 JSON 檔案寫入資料表。
- 查詢特定年份發表的電影。
- 掃描某個年份範圍內發表的電影。
- 從資料表刪除電影，然後刪除資料表。

適用於 .NET 的 SDK (v4)

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
/// <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();
        }
    }
}
```

建立包含電影資料的資料表。

```
    /// <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;
        }
    }
```

新增單一電影到資料表。

```
    /// <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;
        }
    }
```

更新資料表中的單一項目。

```
    /// <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;
        }
    }
```

從電影資料表擷取單一項目。

```
    /// <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;
        }
    }
```

將一批項目寫入電影資料表。

```
    /// <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;
        }
    }
```

從資料表刪除單一項目。

```
    /// <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;
        }
    }
```

在資料表中查詢特定年份發表的電影。

```
    /// <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;
        }
    }
```

在資料表中掃描某個年份範圍內發表的電影。

```
    /// <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;
        }
```

```
    }
```

刪除電影資料表。

```
    /// <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;
        }
    }
```

- 如需 API 詳細資訊，請參閱《適用於 .NET 的 AWS SDK API 參考》中的下列主題。
  - BatchWriteItem
  - CreateTable
  - DeleteItem
  - DeleteTable
  - DescribeTable
  - GetItem
  - PutItem
  - 查詢
  - 掃描
  - UpdateItem

動作

`BatchWriteItem`

以下程式碼範例顯示如何使用 `BatchWriteItem`。

適用於 .NET 的 SDK (v4)

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

將一批項目寫入電影資料表。

```
    /// <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;
        }
    }
```

- 如需 API 的詳細資訊，請參閱《適用於 .NET 的 AWS SDK API 參考》中的 BatchWriteItem。

`CreateTable`

以下程式碼範例顯示如何使用 `CreateTable`。

適用於 .NET 的 SDK (v4)

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
    /// <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;
        }
    }
```

- 如需 API 的詳細資訊，請參閱《適用於 .NET 的 AWS SDK API 參考》中的 CreateTable。

`DeleteItem`

以下程式碼範例顯示如何使用 `DeleteItem`。

適用於 .NET 的 SDK (v4)

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
    /// <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;
        }
    }
```

- 如需 API 的詳細資訊，請參閱《適用於 .NET 的 AWS SDK API 參考》中的 DeleteItem。

`DeleteTable`

以下程式碼範例顯示如何使用 `DeleteTable`。

適用於 .NET 的 SDK (v4)

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
    /// <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;
        }
    }
```

- 如需 API 的詳細資訊，請參閱《適用於 .NET 的 AWS SDK API 參考》中的 DeleteTable。

`GetItem`

以下程式碼範例顯示如何使用 `GetItem`。

適用於 .NET 的 SDK (v4)

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
    /// <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;
        }
    }
```

- 如需 API 的詳細資訊，請參閱《適用於 .NET 的 AWS SDK API 參考》中的 GetItem。

PutItem

以下程式碼範例顯示如何使用 PutItem。

適用於 .NET 的 SDK (v4)

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
    /// <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;
        }
    }
```

- 如需 API 的詳細資訊，請參閱《適用於 .NET 的 AWS SDK API 參考》中的 PutItem。

Query

以下程式碼範例顯示如何使用 Query。

適用於 .NET 的 SDK (v4)

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
    /// <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;
        }
    }
```

- 如需 API 的詳細資訊，請參閱《適用於 .NET 的 AWS SDK API 參考》中的 Query。

Scan

以下程式碼範例顯示如何使用 Scan。

適用於 .NET 的 SDK (v4)

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
    /// <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;
        }
    }
```

- 如需 API 的詳細資訊，請參閱《適用於 .NET 的 AWS SDK API 參考》中的 Scan。

`UpdateItem`

以下程式碼範例顯示如何使用 `UpdateItem`。

適用於 .NET 的 SDK (v4)

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
    /// <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;
        }
    }
```

- 如需 API 的詳細資訊，請參閱《適用於 .NET 的 AWS SDK API 參考》中的 UpdateItem。

使用 適用於 .NET 的 SDK (v4) 的 Amazon EC2 範例

下列程式碼範例示範如何使用 適用於 .NET 的 AWS SDK (v4) 搭配 Amazon EC2 來執行動作和實作常見案例。

每個範例都包含完整原始程式碼的連結，您可以在其中找到如何在內容中設定和執行程式碼的指示。

開始使用

您好 Amazon EC2

下列程式碼範例示範如何開始使用 Amazon EC2。

適用於 .NET 的 SDK (v4)

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

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

- 如需 API 詳細資訊，請參閱《適用於 .NET 的 AWS SDK API 參考》中的 DescribeSecurityGroups。

使用 適用於 .NET 的 SDK (v4) 的 Amazon ECS 範例

下列程式碼範例示範如何使用 適用於 .NET 的 AWS SDK (v4) 搭配 Amazon ECS 來執行動作和實作常見案例。

每個範例都包含完整原始程式碼的連結，您可以在其中找到如何在內容中設定和執行程式碼的指示。

開始使用

您好，Amazon ECS

下列程式碼範例說明如何開始使用 Amazon ECS。

適用於 .NET 的 SDK (v4)

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

```
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.");
        }

    }
}
```

- 如需 API 詳細資訊，請參閱適用於 .NET 的 AWS SDK 《API 參考》中的 ListClusters。

使用 適用於 .NET 的 SDK (v4) 的 Amazon S3 範例

下列程式碼範例示範如何使用 適用於 .NET 的 AWS SDK (v4) 搭配 Amazon S3 來執行動作和實作常見案例。

Actions 是大型程式的程式碼摘錄，必須在內容中執行。雖然動作會告訴您如何呼叫個別服務函數，但您可以在其相關情境中查看內容中的動作。

案例是向您展示如何呼叫服務中的多個函數或與其他 AWS 服務組合來完成特定任務的程式碼範例。

每個範例都包含完整原始程式碼的連結，您可以在其中找到如何在內容中設定和執行程式碼的指示。

主題

- 動作
- 案例

動作

`CreatePresignedPost`

以下程式碼範例顯示如何使用 `CreatePresignedPost`。

適用於 .NET 的 SDK (v4)

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

建立預先簽章的 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);
    }
```

- 如需 API 詳細資訊，請參閱適用於 .NET 的 AWS SDK 《 API 參考》中的 CreatePresignedPost。

案例

建立預先簽章 URL

下列程式碼範例示範如何建立 Amazon S3 的預先簽章 URL 並上傳物件。

適用於 .NET 的 SDK (v4)

> **Note**
>
> GitHub 上提供更多範例。尋找完整範例，並了解如何在 AWS 程式碼範例儲存庫中設定和執行。

建立並使用預先簽章的 POST URLs 進行直接瀏覽器上傳。

```
/// <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");
```

```
            }
        }
    }
}
```

遷移 的專案 適用於 .NET 的 AWS SDK

本節提供有關 第 4 版遷移任務的資訊 適用於 .NET 的 AWS SDK，以及如何執行這些任務的說明。如需遷移至舊版的資訊（例如，從 V2 遷移至 V3，或從一個 V3 子版本遷移至另一個版本），請參閱《開發人員指南》（第 3 版）中的遷移您的專案。 適用於 .NET 的 AWS SDK

另請參閱 最新消息和 支援平台。

主題

- 遷移至 第 4 版 適用於 .NET 的 AWS SDK
- 從 .NET Standard 1.3 遷移

遷移至 第 4 版 適用於 .NET 的 AWS SDK

適用於 .NET 的 AWS SDK 第 4 版 (V4) 有大量 SDK 第 3 版 (V3) 的重大變更。本主題說明第 4 版中的重大變更，以及從 V3 遷移環境或程式碼時可能需要執行的可能工作。如需有關 SDK 中其他值得注意變更的其他資訊，請參閱下列資源：

- GitHub 上的開發追蹤器問題：https://github.com/aws/aws-sdk-net/issues/3362。
- 部落格文章預覽 1 of 適用於 .NET 的 AWS SDK V4。
- 部落格文章預覽 4 of 適用於 .NET 的 AWS SDK V4。
- 部落格文章 V適用於 .NET 的 AWS SDK V40 的一般可用性。

.NET Framework

.NET Framework 3.5 目標已從 的 V4 中移除。 適用於 .NET 的 AWS SDK因此，開發套件不再支援 .NET Framework 3.5。此版本的 SDK 是針對 .NET Framework 4.7.2 編譯，並在 .NET 4.0 執行時間中執行。如需詳細資訊，請參閱支援的平台。

值類型

在用於提出請求和回應的類別中使用值類型的屬性已變更為使用 nullable 值類型。具有下列類型的屬性已變更：

- `bool` 已變更為 `bool?`

- double 已變更為 double?
- int 已變更為 int?
- float 已變更為 float?
- long 已變更為 long?
- Datetime 已變更為 Datetime?

如需有關此變更的其他資訊，請參閱 V4 的部落格文章預覽 1 適用於 .NET 的 AWS SDK。

集合

在用於提出請求和回應的類別中使用集合的屬性現在預設為 null。因此，您的程式碼在嘗試使用集合之前，需要驗證集合不是 null。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.
    }
}
```

初始化集合的 V3 行為可以透過將 設定為 Amazon.AWSConfigs.InitializeCollections來還原true。對於想要在升級至 V4 之前嘗試此行為變更的使用者，此屬性也存在於 V3 中。 V4

如需此變更的其他資訊，請參閱 V4 的部落格文章預覽 1 適用於 .NET 的 AWS SDK。

AWS Security Token Service (STS)

- 區域端點

  使用依賴的登入資料提供者時 AWS STS，呼叫一律使用區域端點。這與在公有分割區中執行時預設使用 us-east-1區域的 SDK V3 不同，無論設定的區域為何。

- **StsRegionalEndpointsValue**列舉

  列舉已從 StsRegionalEndpointsValue Amazon.Runtime 命名空間中移除。任何使用該列舉的程式碼都應該移除。

- **`STSAssumeRoleAWSCredentials` 類別**

  已取代的 STS 擔任角色登入資料提供者 `STSAssumeRoleAWSCredentials`已從 Amazon.SecurityToken 命名空間中移除。請改用 Amazon.Runtime 的 AssumeRoleAWSCredentials。

與 相關的變更 `ClientConfig`

Amazon.Runtime.ClientConfig 類別是服務用戶端組態類別的基本類別，例如 AmazonS3Config。已對此基本類別進行下列變更。

- 預設重試模式

  `RetryMode` 屬性預設為 ，`Standard`而不是 `Legacy`。因此，該`Legacy`值已從 Amazon.Runtime.RequestRetryMode 列舉中移除。

- 預設組態模式

  `DefaultConfigurationMode` 屬性預設為 `Standard`，而不是 `Legacy`。因此，該`Legacy`值已從 Amazon.Runtime.DefaultConfigurationMode 列舉中移除。

- **`ReadWriteTimeout` 屬性**

  從除 .NET Framework 4.7.2 以外的所有目標移除過時`ReadWriteTimeout`屬性。

AWSSDK.Extensions.NETCore.Setup NuGet 套件

AWSSDK.Extensions.NETCore.Setup NuGet 套件已更新，以減輕 SDK V3 中存在的問題，並使原生 AOT 的套件安全。這些變更摘要如下。如需詳細資訊，請參閱 GitHub 上 aws-sdk-net 儲存庫中的 PR 3353。

- **`DefaultClientConfig` 類別**

  `DefaultClientConfig` 類別不再繼承自服務用戶端組態基礎類別 Amazon.Runtime.ClientConfig。來自 的相關屬性`ClientConfig`已在 `DefaultClientConfig` 上使用 nullable 值類型複寫。此變更可讓我們在將值複製到為服務用戶端建立的組態`DefaultClientConfig`時，偵測是否已在 上設定值。

  此變更的一個特定結果是 V4 `DefaultClientConfig.HttpClientFactory` 中不再提供。請改用 `AWSConfigs.HttpClientFactory`。如需詳細資訊，請參閱 GitHub 問題 3790。

此變更的另一個結果是，已接受服務組態物件之IConfiguration.GetAWSOptions延伸方法的一般過載已移除。應該改用非一般過載，開發套件會自動處理填入服務特定的設定。如需詳細資訊，請參閱 GitHub 問題 3866。

- 原生 AOT

  建立使用 C# 11 靜態介面方法之服務用戶端的新機制已新增至套件。此變更不需要執行組裝類型載入來建立服務用戶端的執行個體，包括服務介面名稱的字串操作來運算與原生 AOT 不相容的服務用戶端類型。此變更僅適用於 .NET 8 和更新版本；較舊版本仍使用原始機制。

如需本指南中有關此套件的其他資訊，請參閱 AWSSDK.Extensions.NETCore.Setup 和 IConfiguration。此套件的原始碼位於 GitHub，網址為 https://github.com/aws/aws-sdk-net/tree/main/extensions/src/AWSSDK.Extensions.NETCore.Setup。

CookieSigner 和 UrlSigner

Amazon CloudFront 的 CookieSigner和 UrlSigner擴充功能已移至名為 AWSSDK.Extensions.CloudFront.Signers 的個別擴充功能套件。此變更是支援 OpenSSL 3，並對 BouncyCastle.Cryptography 採取相依性。

此套件的原始碼位於 GitHub，網址為 https://github.com/aws/aws-sdk-net/tree/main/extensions/src/AWSSDK.Extensions.CloudFront.Signers。

DateTime 與 UTC DateTime

有些 V3 類別具有標示為「已棄用」或「淘汰」的 DateTime 屬性，以及替代的 UTC DateTime 屬性。在這些類別中，已移除過時的 DateTime 屬性，並將 UTC DateTime 屬性的名稱變更為 DateTime 屬性的原始名稱。

以下是已實作此變更之類別的一些範例。

- DescribeSpotPriceHistoryRequest：
  - 已移除過時StartTime屬性，且StartTimeUtc屬性的名稱已變更為「StartTime」。
  - 已移除過時EndTime屬性，且EndTimeUtc屬性的名稱已變更為「EndTime」。
- CreateFleetRequest
  - 已移除過時ValidFrom屬性，且ValidFromUtc屬性的名稱已變更為「ValidFrom」。
  - 已移除過時ValidUntil屬性，且ValidUntilUtc屬性的名稱已變更為「ValidUntil」。

如果應用程式使用原始、已淘汰的 DateTime 屬性，則此變更可能會導致偏移時間。使用 UTC DateTime 屬性的程式碼會發生編譯時間錯誤。

DateTime 剖析

DateTimeUnmarshaller 類別已更新。此類別已剖析並傳回 DateTime 字串做為本機時間。在某些情況下，由於先前的更新，這些值會轉換回 UTC，但並非一律如此。現在，未封送的 DateTime 字串會假設為 UTC，並將指定和取消封送為 UTC。此更新包含下列行為變更。

某些以 DateTime 類別為基礎的時間戳記屬性已剖析為當地時間。這些包括時間戳記的回應取消封送程式，並列出格式 `TimestampFormat.ISO8601`和 的時間戳記`TimestampFormat.RFC822`。DateTime 剖析已更新，改為傳回 UTC 時間。

`ConvertFromUnixEpochSeconds` 和 `ConvertFromUnixEpochMilliseconds`

ConvertFromUnixEpochSeconds 和 ConvertFromUnixEpochMilliseconds 方法會將 Unix epoch 秒轉換為 DateTime 結構，並以本機時間而非 UTC 時間傳回 Unix Epoch 時間。這些方法現在會傳回 UTC 時間。

日誌

您在開發套件中啟用記錄的方式已更新為 V4。記錄到主控台和系統診斷的運作方式與 V3 相同；也就是說，將 AWSConfigs 類別的 `LoggingConfig.LogTo` 屬性設定為 `LoggingOptions.Console`或 `LoggingOptions.SystemDiagnostics`。的 LoggingOptions 選項`log4net`已與 SDK 的內部邏輯一起移除，以使用反射連接至 的記憶體內執行個體`log4net`。

若要將開發套件的登入納入記錄架構，會使用個別的轉接器套件來連接開發套件與記錄架構。使用適用於 的 AWSSDK.Extensions.Logging.Log4NetAdaptor 套件`log4net`和適用於 的 AWSSDK.Extensions.Logging.ILoggerAdaptor 套件`Microsoft.Extensions.Logging`。下列程式碼範例示範如何在這兩種情況下設定記錄。

log4net 的範例組態

新增 `AWSSDK.Extensions.Logging.Log4NetAdaptor` NuGet 套件，並從 呼叫靜態`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.Extensions.Logging 的範例組態

新增 AWSSDK.Extensions.Logging.ILoggerAdaptor NuGet 套件，並從 ILoggerFactory 界面呼叫ConfigureAWSSDKLogging擴充功能方法。

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

var app = builder.Build();

app.Services.GetRequiredService<ILoggerFactory>()
    .ConfigureAWSSDKLogging();
```

支援 HTTP 2

已新增對 HTTP 2 的支援，以啟用雙向串流。如需更多資訊，請參閱支援 HTTP 2。

單一登入

SSOAWSCredentialsOptions 類別的 SupportsGettingNewToken 屬性預設值已從 變更為 true false。如果您有使用 SSOAWSCredentials 類別來取得 SSO 登入資料的應用程式，您可能需要將 Options.SupportsGettingNewToken 屬性設定為 true。如需此組態的範例，請參閱 中的程式碼範例僅使用 .NET 應用程式的 SSO 教學課程。如需詳細資訊，請參閱 aws-sdk-net GitHub 儲存庫中的 PR 3737。

DynamoDB 特定的變更

下列變更專屬於 Amazon DynamoDB。其中許多都是重大變更。

如需有關 的 V4 中 DynamoDB 變更的其他資訊 適用於 .NET 的 AWS SDK，請參閱部落格文章 V4 的適用於 .NET 的 AWS SDK V4。

開啟以檢視項目

適用於 DynamoDB 的 SDK 中的 V4 變更解決了有關可測試性的一些問題，但主要以高階程式庫為中心：

- 程式碼中名為 DocumentModel 的 .NET Document Model。
- 程式碼中名為 DataModel 的 .NET Object Persistence Model。

如需這些程式設計模式的詳細資訊，請參閱本指南DynamoDB中的 。

文件模型：更新模擬**IAmazonDynamoDB**界面的例外狀況

在開發套件 V4 之前的文件模型中，如果資料表是使用模擬 IAmazonDynamoDB 介面初始化，則會傳回 NullReferenceException。開發套件的 V4 會InvalidOperationException改為傳回。非同步Table方法應與模擬用戶端搭配使用，但當您從 呼叫同步方法時，仍可能會看到例外狀況.NET/Core/Standard。

如需此變更的詳細資訊，請參閱 GitHub 上的 PR 3388。

文件模型： **FromJson**和 **ToJson**方法

文件類別的 FromJson和 ToJson方法現在使用 System.Text.Json而非 LitJson 進行序列化，而且 LitJson 已從 SDK 的 V4 中移除。使用 的好處System.Text.Json是此剖析器支援使用 .NET Decimal類型，該類型支援數值浮點屬性的更高精確度。

物件持久性模型： **DynamoDBOperationConfig**類別

在物件持久性模型中，已對共用的 DynamoDBOperationConfig 類別進行下列變更：

- 類別已分成新的操作特定類別，例如 SaveConfig、LoadConfig 和 QueryConfig。採用的方法DynamoDBOperationConfig已標示為已過時，且未來可能會移除。

  如需此變更的詳細資訊，請參閱 GitHub 上的 PR 3421。
- 已從 類別中移除 MetadataCachingMode和 DisableFetchingTableMetadata 屬性。這些屬性不包含在先前提及的新操作特定類別中。移除的屬性是資料表層級設定，應在 AWSConfigsDynamoDB 類別的全域Context屬性或 DynamoDBContextConfig 類別上指定。

  如需此變更的詳細資訊，請參閱 GitHub 上的 PR 3422。
- 類別不再繼承自 DynamoDBContextConfig 類別。這可防止您將DynamoDBOperationConfig物件傳遞至 DynamoDBContext 的建構函式，其中操作特定組態 （ 例如 OverrideTableName) 上的某些屬性不適用。

  如需此變更的詳細資訊，請參閱 GitHub 上的 PR 3422。

物件持久性模型：多形性

DynamoDBPolymorphicTypeAttribute 類別已新增至物件持久性模型。此類別支援多態類型的序列化和還原序列化。如需詳細資訊，請參閱 GitHub 上的 PR 3643。

文件模型和物件持久性模型：可模擬操作

已新增新的操作特定界面，允許客戶模擬 DynamoDB 操作。IDynamoDBContext 界面上的原廠方法已更新，以傳回新的界面。

如需此變更的詳細資訊，請參閱 GitHub 上的 PR 3450。

- 物件持續性模型
  - 透過 IBatchGet和 IMultiTableBatchGet 介面的模擬BatchGet操作。
  - 透過 IBatchWrite和 IMultiTableBatchWrite 介面的模擬BatchWrite操作。
  - 透過 ITransactGet和 IMultiTableTransactGet 介面的模擬TransactGet操作。
  - 透過 ITransactWrite和 IMultiTableTransactWrite 介面的模擬TransactWrite操作。
  - 透過 IAsyncSearch 界面的模擬Scan和Query操作。
- 文件模型
  - 透過 ITable 界面的模擬Table操作。
  - 透過 ISearch 界面的模擬Scan和Query操作。
  - 透過 IDocumentTransactWrite和 IMultiTableDocumentTransactWrite 介面的模擬TransactWrite操作。
  - 透過 IDocumentTransactGet和 IMultiTableDocumentTransactGet 介面的模擬TransactGet操作。
  - 透過 IDocumentBatchWrite和 IMultiTableDocumentBatchWrite 介面的模擬BatchWrite操作。
  - 透過 IDocumentBatchGet和 IMultiTableDocumentBatchGet 介面的模擬BatchGet操作。

文件模型和物件持久性模型：支援原生 AOT

原生 AOT 的限制支援巢狀 .NET 類型。在某些情況下，.NET 編譯器的修剪元件可能會忽略這些巢狀類型。在這種情況下，您可能會收到例外狀況，例如："System.InvalidOperationException: Type <type> is unsupported, it cannot be instantiated."

您可以藉由在程式碼路徑中DynamicDependency新增 來解決此限制，通知修剪器有關子類型的相依性。儲存的最上層 .NET 類型的建構函式可能是 位置。下列程式碼範例示範如何使用 DynamicDependency 屬性：

```
[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; }
}
```

DynamoDBStreams

DynamoDBStreams 已從 AWSSDK.DynamoDB NuGet 套件中移除，並可在自己的套件 AWSSDK.DynamoDBStreams 中使用，並有自己的命名空間 Amazon.DynamoDBStreams。

允許移除 **TableNamePrefix**值

您現在可以在個別操作層級的 DynamoDBContextConfig 類別中移除 TableNamePrefix 屬性的值。如需此變更的詳細資訊，請參閱 GitHub 上的 PR 3476。

RetrieveDateTimeInUtc 屬性

對於 DynamoDBContextConfig 類別， RetrieveDateTimeInUtc 屬性的預設值已變更為 true。

DynamoDBContextTableNamePrefix 屬性

從 AWSConfigsDynamoDB 類別移除 DynamoDBContextTableNamePrefix 屬性。使用者應呼叫 AWSConfigsDynamoDB.Context.TableNamePrefix，而不是

EC2 特定的變更

下列變更專屬於 Amazon EC2。其中大多數或全部正在中斷變更。

開啟以檢視項目

`GetDecryptedPassword`

Amazon EC2 的`GetDecryptedPassword`擴充功能已移至名為 AWSSDK.Extensions.EC2.DecryptPassword 的個別擴充功能套件。此變更是為了支援 OpenSSL 3，並對 BouncyCastle.Cryptography 採取相依性。

此套件的原始碼位於 GitHub，網址為 https://github.com/aws/aws-sdk-net/tree/main/extensions/src/AWSSDK.Extensions.EC2.DecryptPassword。

支援 Amazon EC2 IMDSv1

執行個體中繼資料服務第 1 版 (IMDSv1) 的支援已移除。從 IMDS 擷取登入資料和其他中繼資料時，軟體開發套件的 V4 一律使用執行個體中繼資料服務第 2 版 (IMDSv2)。IMDSv2 如需 IMDS 的詳細資訊，請參閱《Amazon EC2 使用者指南》中的使用 IMDS。 Amazon EC2

已變更或移除的程式設計元素

- 已移除整個`Amazon.EC2.Import`命名空間和程式碼。
- 已移除整個`Amazon.EC2.Util`命名空間和程式碼，其中包含用於查詢 Windows EC2 AMIs AMI 公用程式。
- 已將淘汰的 `IpRanges` 屬性從 IpPermission 類別中移除。請改用 `Ipv4Ranges`或 `Ipv6Ranges` 屬性。
- 下列已移除 EC2InstanceMetadata 類別中的過時欄位：`EC2_METADATA_SVC`、`EC2_METADATA_ROOT`、`EC2_USERDATA_ROOT`、`EC2_DYNAMICDATA_ROOT`和 `EC2_APITOKEN_URL`。

S3 特定的變更

下列變更專屬於 Amazon S3。其中大多數或全部正在中斷變更。

開啟以檢視項目

AWS 區域 us-east-1

為 us-east-1區域設定的 Amazon S3 服務用戶端無法再存取其他區域中的儲存貯體。必須使用為儲存貯體所在的區域設定的 S3 服務用戶端來存取儲存貯體。

如需有關此變更的其他資訊，請參閱 V4 適用於 .NET 的 AWS SDK 的部落格文章預覽 V4。

S3 加密用戶端

Amazon.S3.Encryption 命名空間中定義的 Amazon S3 加密用戶端已從 AWSSDK.S3 套件中移除。此用戶端已移至自己的套件，稱為 Amazon.Extensions.S3.Encryption，其文件位於 https://aws.github.io/amazon-s3-encryption-client-dotnet/api/Amazon.Extensions.S3.Encryption.html。如需遷移的相關資訊，請參閱 the section called “S3 加密用戶端遷移”。如需 S3 加密的詳細資訊，請參閱《Amazon S3 加密用戶端開發人員指南》中的支援的加密演算法。

的 S3 標記指令 **CopyObject**

TaggingDirective 屬性已公開為 CopyObjectRequest 類別的公有屬性，由 AmazonS3Client.CopyObject 方法使用。此屬性對應至 Amazon S3 x-amz-tagging-directive 參數，如 CopyObject 動作所定義。

標記指令不會再自動設定為 COPY。如果開發人員未指定標記指令，S3 後端會自動假設它是 COPY，但如果開發人員明確將 屬性設定為 null，則完全不會設定該值。

S3 組態的 **UseArnRegion** 屬性

Amazon.S3.AmazonS3Config 類別的 UseArnRegion 屬性已更新，因此AWS_S3_USE_ARN_REGION環境變數優先於共用 AWS config檔案中s3_use_arn_region的設定。如需這些變數和設定的詳細資訊，請參閱 SDK 和工具參考指南中的設定參考。 AWS SDKs

CopyObject 和 **CopyPart**方法的前斜線

Amazon S3 CopyObject和 CopyPart方法不會再修剪正斜線。DisableTrimmingLeadingSlash 屬性已從 CopyObjectRequest 和 CopyPartRequest 類別中移除。

DoesS3BucketExist... 方法

已將淘汰的 DoesS3BucketExist和 DoesS3BucketExistAsync方法從 AmazonS3Util 類別中移除，這會實作 ICoreAmazonS3 介面。這些方法已移除，因為它們一律使用 HTTP。請改用 DoesS3BucketExistV2 和 DoesS3BucketExistV2Async。

SDK 一律使用 SigV4

第 4 版 適用於 .NET 的 AWS SDK 一律使用 AWS Signature 第 4 版 (SigV4) 簽署請求。此變更會導致下列相關變更：

- AWSConfigsS3 類別的 `UseSignatureVersion4` 屬性已移除。
- Amazon.Runtime.ClientConfig 類別的 `SignatureVersion` 屬性已移除。此屬性僅供 Amazon S3 用於回溯相容性。
- 已移除 `RegionEndpoint.Endpoint`類別。這包括 `SignatureVersionOverride` 屬性，用於覆寫 Amazon S3 的簽章版本。請改用 服務特定的`client.DetermineServiceOperationEndPoint()`方法。
- 已更新方法 AmazonS3Util.PostUpload 和 S3PostUploadSignedPolicy.GetSignedPolicy，以使用 SigV4。因此，`S3PostUploadSignedPolicy.GetSignedPolicyV4`方法已移除，因為 `GetSignedPolicy` 現在會執行相同的 函數。此外， `GetSignedPolicy` 已獲得區域端點的第三個參數。

`GetACL` 和 `PutACL`方法

AmazonS3Client 類別的 `GetACL`和 `PutACL`方法已標記為已過時。若要存取這些方法的功能，請改用下列新方法：`GetBucketACL`、`GetObjectACL`、 `PutBucketACL`和 `PutObjectACL`。

已移除過時的程式設計元素

Amazon S3 實作的許多程式設計元素已從 SDK 的 V4 中移除，包括列舉值、類型、方法、命名空間等。如果先前尚未涵蓋這些項目，以及您可以採取以適應其移除的潛在步驟，則會列出如下。

- `DisableMD5Stream` 屬性已從 TransferUtilityUploadRequest 類別中移除。請改用 `DisableDefaultChecksumValidation` 屬性。

  此外， `CalculateContentMD5Header` 屬性已從 `TransferUtilityUploadRequest`類別中移除。此屬性已不再需要，因為 SDK 預設會計算檢查總和。
- `ServerSideEncryptionMethod` 和 `ServerSideEncryptionKeyManagementServiceKeyId` 屬性已從 CopyPartRequest 類別中移除。請改用 InitiateMultipartUploadRequest 類別中名稱相同的屬性，這些屬性用於 AmazonS3Client 類別的某些`InitiateMultipartUpload...`方法。
- `Expires` 屬性已從 GetObjectResponse 類別中移除。請改用 `ExpiresString` 屬性。字串可能不是有效的時間戳記格式，因此您的程式碼在轉換為 時應使用 `TryParse`方法`DateTime`。

- 已將過時的 AWS 區域 識別符從 S3Region 列舉中移除。
- Prefix 屬性已從 LifecycleRule 類別中移除。請改用 Filter 屬性。

  此外，NoncurrentVersionTransition和 Transition 屬性已從 LifecycleRule類別中移除。請改用 NoncurrentVersionTransitions和 Transitions集合。
- Event 屬性已從 TopicConfiguration 類別中移除。請改用Events集合。
- CalculateContentMD5Header 屬性。此屬性不再需要設定，因為 SDK 預設會計算檢查總和。
- Bucket 屬性已從 SelectObjectContentRequest 類別中移除。請改用 BucketName 屬性。
- NumberOfUploadThreads 屬性已從 TransferUtilityConfig 類別中移除。請改用 ConcurrentServiceRequests 屬性。

已移除的程式設計元素

已從 SDK 的 V4 中移除許多程式設計元素，包括列舉值、類型、方法、命名空間等。如果先前尚未涵蓋這些項目，以及您可以採取以適應其移除的潛在步驟，則會列出如下。

開啟以檢視項目

Amazon.Auth.AccessControlPolicy.ActionIdentifiers 命名空間

已移除Amazon.Auth.AccessControlPolicy.ActionIdentifiers命名空間。這包括在 IdentityandAccessManagementActionIdentifiers類別中定義的 IAM 動作識別符。使用這些動作識別符的程式碼應變更為使用動作名稱的字串值。

如需詳細資訊，請參閱《IAM 使用者指南》中的 the section called “從 JSON 建立受管政策”和 JSON 政策概觀。

ClientConfig 類別

Amazon.Runtime.ClientConfig 類別是服務用戶端組態類別的基本類別，例如 AmazonS3Config。已從此類別中移除下列程式設計元素。

- DetermineServiceURL 和 DetermineDnsSuffix方法已移除。請改用 服務用戶端的 DetermineServiceOperationEndpoint方法；例如，AmazonS3Client.DetermineServiceOperationEndpoint。
- ReadEntireResponse 屬性已移除。請改用下列其中一項：
  - AWSConfigs.LoggingConfig 類別的 LogResponses 屬性。
  - 用戶端組態的 LogResponse 屬性，例如 AmazonS3Config。

`Amazon.Runtime` 命名空間

Amazon.Runtime 命名空間已更新，如下所示：

- 已將淘汰的`ECSTaskCredentials`類別從命名空間中移除。請改用 GenericContainerCredentials 提供者，這也支援 Amazon EKS Pod 身分。
- 已將淘汰的 `StoredProfileAWSCredentials`和 `StoredProfileCredentials`類別從命名空間中移除。請改用 Amazon.Runtime.CredentialManagement 命名空間的 NetSDKCredentialsFile 或 SharedCredentialsFile 類別。 CredentialManagement
- SSOAWSCredentials 類別的淘汰`HasCachedAccessTokenAvailable`方法已從命名空間中移除。
- 已將淘汰的`EnvironmentAWSCredentials`類別從命名空間中移除。請改用 AppConfigAWSCredentials 類別。
- 已將淘汰的`StoredProfileFederatedCredentials`類別從命名空間中移除。請改用 FederatedAWSCredentials 類別。
- 已從命名空間中移除下列已淘汰的類別：`EnvironmentVariableAWSEndpointDiscoveryEnabled`、`ProfileAWSEndpointDiscoveryEnabled`和`FallbackEndpointDiscoveryEnabledFactory`。
- 已將淘汰的 `UseSigV4` 屬性從 AmazonWebServiceRequest 類別中移除。請改用 `SignatureVersion` 屬性。
- `Amazon.Runtime.Internal.Util` 命名空間中的 `ProfileIniFile`類別具有稱為 的過載方法`TryGetSection`。不支援 的 `out` 參數的 方法版本`nestedProperties`已從 類別中移除。
- 已移除`Amazon.Runtime.Internal.Auth`命名空間中已淘汰的`EventBridgeSigner`類別。
- 已將過時的`Parameters`字典屬性從 WebServiceRequestEventArgs 類別中移除。請改用 `ParameteCollection` 屬性。

BouncyCastle

BouncyCastle 的來源複本已從 SDK 的 V4 中移除。

`StoredProfileSAMLCredentials` 類別

Amazon.SecurityToken.SAML 命名空間中的淘汰`StoredProfileSAMLCredentials`類別已移除。請改用 Amazon.Runtime 命名空間中的 FederatedAWSCredentials 類別。

`AWSSDKUtils` 類別

下列方法已從 AWSSDKUtils 類別中移除：`ProtectEncodedSlashUrlEncode`、`ResolveResourcePath`和 `ConvertToUnixEpochMilliSeconds`。

`ProfileManager` 類別

已將淘汰的`ProfileManager`類別從 Amazon.Util 命名空間中移除。請改用 Amazon.Runtime.CredentialManagementCredentialManagement 命名空間中的 NetSDKCredentialsFile 或 SharedCredentialsFile 類別。

`AWSConfigs` 類別

下列淘汰屬性已從 AWSConfigs 類別中移除：`ResponseLogging`、 `Logging`和 `LogMetrics`。請改用 `LoggingConfig` 屬性。

`ConditionFactory` 類別

具有下列簽章的 方法已從 ConditionFactory 類別中移除：`NewCondition(ConditionFactory.DateComparisonType, DateTime)`。請改用 NewConditionUtc 方法。

Amazon CloudFront 公用程式

已移除過時的`Amazon.CloudFront.Util`命名空間和`AmazonCloudFrontUtil`類別。

AWS IoT

在 ListPrincipalThingsResponse 類別中，`NextToken`覆寫的舊版自訂已移除，有利於分頁。

AWS Lambda

AmazonLambdaClient 類別的下列`Invoke...`方法已移除，因為名稱令人混淆。

- 具有下列簽章的 V3 方法已移除：`InvokeAsyncResponse InvokeAsync(InvokeAsyncRequest)`。這是 SDK V3 中的同步方法。請改用 `InvokeResponse Invoke(InvokeRequest)`（用於同步處理）或 `Task InvokeAsync(InvokeRequest, CancellationToken)`（用於非同步處理）。
- 具有下列簽章的 V3 方法已移除：`Task InvokeAsyncAsync(InvokeAsyncRequest, CancellationToken)`。這是 SDK V3 中的非同步方法。請改用 `Task InvokeAsync(InvokeRequest, CancellationToken)`。

Amazon SageMaker 執行期

已移除已淘汰的 PayloadPart 類別建構函數。

從 .NET Standard 1.3 遷移

在 2019 年 6 月 27 日，Microsoft 結束了對 .NET Core 1.0 和 .NET Core 1.1 版本的支援。在此公告之後， AWS 已於 2020 年 12 月 31 適用於 .NET 的 AWS SDK 日終止對 .NET Standard 1.3 的支援。

AWS 持續提供 適用於 .NET 的 AWS SDK 針對 .NET Standard 1.3 的服務更新和安全性修正，直到 2020 年 10 月 1 日為止。在該日期之後，.NET Standard 1.3 目標進入維護模式，這表示沒有發佈新的更新；僅 AWS 套用關鍵錯誤修正和安全性修補程式。

在 2020 年 12 月 31 日， 上的 .NET Standard 1.3 支援 適用於 .NET 的 AWS SDK 已終止。在該日期之後，不會套用任何錯誤修正或安全修補程式。使用該目標建置的成品仍可在 NuGet 上下載。

您需要執行的事項

- 如果您使用 .NET Framework 執行應用程式，則不會受到影響。
- 如果您使用 .NET Core 2.0 或更高版本執行應用程式，則不會受到影響。
- 如果您使用 .NET Core 1.0 或 .NET Core 1.1 執行應用程式，請依照 Microsoft 遷移說明，將應用程式遷移到較新版本的 .NET Core。我們建議至少使用 .NET Core 3.1。
- 如果您執行的是目前無法升級的關鍵業務應用程式，您可以繼續使用目前的 適用於 .NET 的 AWS SDK版本。

如果您有問題或疑慮，請聯絡 AWS 支援部門。

此 AWS 產品或服務的安全性

雲端安全是 Amazon Web Services (AWS) 最重視的一環。身為 AWS 客戶的您，將能從資料中心和網路架構的建置中獲益，以滿足組織最為敏感的安全要求。安全性是 AWS 與您之間的共同責任。共同責任模型 將此描述為雲端本身的安全和雲端內部的安全。

雲端的安全性 – AWS 負責保護執行 AWS 雲端中提供的所有服務的基礎設施，並為您提供可安全使用的服務。我們的安全責任是 的最高優先順序 AWS，我們的安全有效性由第三方稽核人員定期測試和驗證，作為AWS 合規計劃的一部分。

雲端的安全性 – 您的責任取決於您使用 AWS 的服務，以及其他因素，包括資料的敏感度、組織的需求，以及適用的法律和法規。

此 AWS 產品或服務會透過其支援的特定 Amazon Web Services (AWS) 服務，遵循共同責任模型。如需 AWS 服務安全資訊，請參閱AWS 服務安全文件頁面，以及AWS 合規計劃在 AWS 合規工作範圍內的服務。

主題

- 此 AWS 產品或服務中的資料保護
- 身分和存取權管理
- 此 AWS 產品或服務的合規驗證
- 此 AWS 產品或服務的彈性
- 此 AWS 產品或服務的基礎設施安全
- 在 中強制執行最低 TLS 版本 適用於 .NET 的 AWS SDK
- Amazon S3 加密用戶端遷移

此 AWS 產品或服務中的資料保護

AWS 共同責任模型適用於此 AWS 產品或服務中的資料保護。如此模型所述， AWS 負責保護執行所有 的 全球基礎設施 AWS 雲端。您負責維護在此基礎設施上託管內容的控制權。您也同時負責所使用 AWS 服務 的安全組態和管理任務。如需資料隱私權的詳細資訊，請參閱資料隱私權常見問答集。如需有關歐洲資料保護的相關資訊，請參閱 AWS 安全性部落格上的 AWS 共同的責任模型和 GDPR 部落格文章。

基於資料保護目的，建議您保護 AWS 帳戶 登入資料，並使用 AWS IAM Identity Center 或 AWS Identity and Access Management (IAM) 設定個別使用者。如此一來，每個使用者都只會獲得授與完成其任務所必須的許可。我們也建議您採用下列方式保護資料：

- 每個帳戶均要使用多重要素驗證 (MFA)。
- 使用 SSL/TLS 與 AWS 資源通訊。我們需要 TLS 1.2 並建議使用 TLS 1.3。
- 使用 設定 API 和使用者活動記錄 AWS CloudTrail。如需有關使用 CloudTrail 追蹤擷取 AWS 活動的資訊，請參閱AWS CloudTrail 《 使用者指南》中的使用 CloudTrail 追蹤。
- 使用 AWS 加密解決方案，以及其中的所有預設安全控制 AWS 服務。
- 使用進階的受管安全服務 (例如 Amazon Macie)，協助探索和保護儲存在 Amazon S3 的敏感資料。
- 如果您在 AWS 透過命令列界面或 API 存取 時需要 FIPS 140-3 驗證的密碼編譯模組，請使用 FIPS 端點。如需有關 FIPS 和 FIPS 端點的更多相關資訊，請參閱聯邦資訊處理標準 (FIPS) 140-3。

我們強烈建議您絕對不要將客戶的電子郵件地址等機密或敏感資訊，放在標籤或自由格式的文字欄位中，例如名稱欄位。這包括當您使用此 AWS 產品或服務，或使用主控台、API AWS CLI或 AWS SDKs的其他 AWS 服務 時。您在標籤或自由格式文字欄位中輸入的任何資料都可能用於計費或診斷日誌。如果您提供外部伺服器的 URL，我們強烈建議請勿在驗證您對該伺服器請求的 URL 中包含憑證資訊。

身分和存取權管理

AWS Identity and Access Management (IAM) 是 AWS 服務 ，可協助管理員安全地控制對 AWS 資源的存取。IAM 管理員可控制誰可以進行驗證 （ 登入 ） 和授權 （ 具有許可 ） 來使用 AWS 資源。IAM 是您可以免費使用 AWS 服務 的 。

主題

- 目標對象
- 使用身分驗證
- 使用政策管理存取權
- AWS 服務 如何使用 IAM
- 對 AWS 身分和存取進行故障診斷

目標對象

使用方式 AWS Identity and Access Management (IAM) 會有所不同，取決於您在 中執行的工作 AWS。

服務使用者 – 如果您使用 AWS 服務 執行任務，管理員會為您提供所需的登入資料和許可。當您使用更多 AWS 功能來執行工作時，您可能需要額外的許可。了解存取許可的管理方式可協助您向管理員請求正確的許可。如果您無法存取 中的功能 AWS，請參閱 對 AWS 身分和存取進行故障診斷或 AWS 服務 您正在使用的 使用者指南。

服務管理員 – 如果您負責公司 AWS 的資源，您可能擁有 的完整存取權 AWS。您的任務是判斷服務使用者應存取哪些 AWS 功能和資源。接著，您必須將請求提交給您的 IAM 管理員，來變更您服務使用者的許可。檢閱此頁面上的資訊，了解 IAM 的基本概念。若要進一步了解貴公司如何使用 IAM AWS，請參閱您正在使用的 使用者指南 AWS 服務 。

IAM 管理員：如果您是 IAM 管理員，建議您掌握如何撰寫政策以管理 AWS存取權的詳細資訊。若要檢視您可以在 IAM 中使用的以 AWS 身分為基礎的政策範例，請參閱 AWS 服務 您正在使用的 使用者指南。

使用身分驗證

身分驗證是您 AWS 使用身分憑證登入 的方式。您必須以 AWS 帳戶根使用者身分、IAM 使用者身分或擔任 IAM 角色來驗證 （ 登入 AWS)。

您可以使用透過身分來源提供的憑證，以聯合身分 AWS 身分身分登入 。 AWS IAM Identity Center (IAM Identity Center) 使用者、您公司的單一登入身分驗證，以及您的 Google 或 Facebook 登入資料，都是聯合身分的範例。您以聯合身分登入時，您的管理員先前已設定使用 IAM 角色的聯合身分。當您使用聯合 AWS 身分存取 時，您會間接擔任角色。

根據您的使用者類型，您可以登入 AWS Management Console 或 AWS 存取入口網站。如需登入的詳細資訊 AWS，請參閱AWS 登入 《 使用者指南》中的如何登入您的 AWS 帳戶 。

如果您以 AWS 程式設計方式存取 ， AWS 會提供軟體開發套件 (SDK) 和命令列界面 (CLI)，以使用您的 憑證以密碼編譯方式簽署您的請求。如果您不使用 AWS 工具，則必須自行簽署請求。如需使用建議的方法自行簽署請求的詳細資訊，請參閱《IAM 使用者指南》中的適用於 API 請求的AWS Signature 第 4 版。

無論您使用何種身分驗證方法，您可能都需要提供額外的安全性資訊。例如， AWS 建議您使用多重驗證 (MFA) 來提高帳戶的安全性。如需更多資訊，請參閱《AWS IAM Identity Center 使用者指南》中的多重要素驗證和《IAM 使用者指南》中的 IAM 中的AWS 多重要素驗證。

AWS 帳戶 根使用者

當您建立 時 AWS 帳戶，您會從一個登入身分開始，該身分可完整存取帳戶中的所有 AWS 服務 和資源。此身分稱為 AWS 帳戶 Theroot 使用者，可透過使用您用來建立帳戶的電子郵件地址和密碼登入來存取。強烈建議您不要以根使用者處理日常任務。保護您的根使用者憑證，並將其用來執行只能由根使用者執行的任務。如需這些任務的完整清單，了解需以根使用者登入的任務，請參閱 IAM 使用者指南中的需要根使用者憑證的任務。

聯合身分

最佳實務是要求人類使用者，包括需要管理員存取權的使用者，使用 聯合身分提供者 AWS 服務 來使用臨時憑證來存取 。

聯合身分是您企業使用者目錄、Web 身分提供者、 AWS Directory Service、Identity Center 目錄，或 AWS 服務 是透過身分來源提供的登入資料存取的任何使用者。當聯合身分存取時 AWS 帳戶，它們會擔任 角色，而角色會提供臨時登入資料。

對於集中式存取權管理，我們建議您使用 AWS IAM Identity Center。您可以在 IAM Identity Center 中建立使用者和群組，也可以連接並同步到您自己的身分來源中的一組使用者 AWS 帳戶 和群組，以便在所有 和應用程式中使用。如需 IAM Identity Center 的詳細資訊，請參閱 AWS IAM Identity Center 使用者指南中的什麼是 IAM Identity Center？。

IAM 使用者和群組

IAM 使用者是 中的身分 AWS 帳戶 ，具有單一人員或應用程式的特定許可。建議您盡可能依賴臨時憑證，而不是擁有建立長期憑證 (例如密碼和存取金鑰) 的 IAM 使用者。但是如果特定使用案例需要擁有長期憑證的 IAM 使用者，建議您輪換存取金鑰。如需更多資訊，請參閱 IAM 使用者指南中的為需要長期憑證的使用案例定期輪換存取金鑰。

IAM 群組是一種指定 IAM 使用者集合的身分。您無法以群組身分簽署。您可以使用群組來一次為多名使用者指定許可。群組可讓管理大量使用者許可的程序變得更為容易。例如，您可以擁有一個名為 IAMAdmins 的群組，並給予該群組管理 IAM 資源的許可。

使用者與角色不同。使用者只會與單一人員或應用程式建立關聯，但角色的目的是在由任何需要它的人員取得。使用者擁有永久的長期憑證，但角色僅提供臨時憑證。如需更多資訊，請參閱《IAM 使用者指南》中的 IAM 使用者的使用案例。

IAM 角色

IAM 角色是 中具有特定許可 AWS 帳戶 的身分。它類似 IAM 使用者，但不與特定的人員相關聯。若要暫時在 中擔任 IAM 角色 AWS Management Console，您可以從使用者切換至 IAM 角色 （主控台）。

您可以透過呼叫 AWS CLI 或 AWS API 操作或使用自訂 URL 來擔任角色。如需使用角色的方法詳細資訊，請參閱《IAM 使用者指南》中的擔任角色的方法。

使用臨時憑證的 IAM 角色在下列情況中非常有用：

- 聯合身分使用者存取 — 如需向聯合身分指派許可，請建立角色，並為角色定義許可。當聯合身分進行身分驗證時，該身分會與角色建立關聯，並獲授予由角色定義的許可。如需有關聯合角色的相關資訊，請參閱《IAM 使用者指南》中的為第三方身分提供者 (聯合) 建立角色。如果您使用 IAM Identity Center，則需要設定許可集。為控制身分驗證後可以存取的內容，IAM Identity Center 將許可集與 IAM 中的角色相關聯。如需有關許可集的資訊，請參閱 AWS IAM Identity Center 使用者指南中的許可集。
- 暫時 IAM 使用者許可 – IAM 使用者或角色可以擔任 IAM 角色來暫時針對特定任務採用不同的許可。
- 跨帳戶存取權：您可以使用 IAM 角色，允許不同帳戶中的某人 (信任的主體) 存取您帳戶的資源。角色是授予跨帳戶存取權的主要方式。不過，對於某些 AWS 服務，您可以將政策直接連接到資源（而不是使用角色做為代理）。如需了解使用角色和資源型政策進行跨帳戶存取之間的差異，請參閱《IAM 使用者指南》中的 IAM 中的跨帳戶資源存取。
- 跨服務存取 – 有些 AWS 服務 使用其他 中的功能 AWS 服務。例如，當您在服務中進行呼叫時，該服務通常會在 Amazon EC2 中執行應用程式或將物件儲存在 Amazon Simple Storage Service (Amazon S3) 中。服務可能會使用呼叫主體的許可、使用服務角色或使用服務連結角色來執行此作業。
  - 轉送存取工作階段 (FAS) – 當您使用 IAM 使用者或角色在其中執行動作時 AWS，您會被視為委託人。使用某些服務時，您可能會執行某個動作，進而在不同服務中啟動另一個動作。FAS 使用呼叫 的委託人許可 AWS 服務，結合 AWS 服務 請求向下游服務提出請求。只有在服務收到需要與其他 AWS 服務 或 資源互動才能完成的請求時，才會提出 FAS 請求。在此情況下，您必須具有執行這兩個動作的許可。如需提出 FAS 請求時的政策詳細資訊，請參閱《轉發存取工作階段》。
  - 服務角色 – 服務角色是服務擔任的 IAM 角色，可代表您執行動作。IAM 管理員可以從 IAM 內建立、修改和刪除服務角色。如需詳細資訊，請參閱《IAM 使用者指南》中的建立角色以委派許可權給 AWS 服務。
  - 服務連結角色 – 服務連結角色是一種連結至 的服務角色類型 AWS 服務。服務可以擔任代表您執行動作的角色。服務連結角色會出現在您的 中 AWS 帳戶 ，並由服務擁有。IAM 管理員可以檢視，但不能編輯服務連結角色的許可。
- 在 Amazon EC2 上執行的應用程式 – 您可以使用 IAM 角色來管理在 EC2 執行個體上執行之應用程式的臨時登入資料，以及提出 AWS CLI 或 AWS API 請求。這是在 EC2 執行個體內儲存存取金鑰的較好方式。若要將 AWS 角色指派給 EC2 執行個體並將其提供給其所有應用程式，您可以建立連接至執行個體的執行個體描述檔。執行個體設定檔包含該角色，並且可讓 EC2 執行個體上執行的

程式取得臨時憑證。如需詳細資訊，請參閱《IAM 使用者指南》中的使用 IAM 角色來授予許可權給 Amazon EC2 執行個體上執行的應用程式。

使用政策管理存取權

您可以透過建立政策並將其連接到身分或資源 AWS 來控制 AWS 中的存取。政策是 中的物件， AWS 當與身分或資源相關聯時， 會定義其許可。當委託人（使用者、根使用者或角色工作階段）發出請求時， 會 AWS 評估這些政策。政策中的許可決定是否允許或拒絕請求。大多數政策會以 JSON 文件 AWS 的形式存放在 中。如需 JSON 政策文件結構和內容的詳細資訊，請參閱 IAM 使用者指南中的 JSON 政策概觀。

管理員可以使用 AWS JSON 政策來指定誰可以存取內容。也就是說，哪個主體在什麼條件下可以對什麼資源執行哪些動作。

預設情況下，使用者和角色沒有許可。若要授予使用者對其所需資源執行動作的許可，IAM 管理員可以建立 IAM 政策。然後，管理員可以將 IAM 政策新增至角色，使用者便能擔任這些角色。

IAM 政策定義該動作的許可，無論您使用何種方法來執行操作。例如，假設您有一個允許 `iam:GetRole` 動作的政策。具有該政策的使用者可以從 AWS Management Console AWS CLI、 或 AWS API 取得角色資訊。

身分型政策

身分型政策是可以附加到身分 (例如 IAM 使用者、使用者群組或角色) 的 JSON 許可政策文件。這些政策可控制身分在何種條件下能對哪些資源執行哪些動作。如需了解如何建立身分型政策，請參閱《IAM 使用者指南》中的透過客戶管理政策定義自訂 IAM 許可。

身分型政策可進一步分類成內嵌政策或受管政策。內嵌政策會直接內嵌到單一使用者、群組或角色。受管政策是獨立的政策，您可以連接到 中的多個使用者、群組和角色 AWS 帳戶。受管政策包括 AWS 受管政策和客戶受管政策。如需了解如何在受管政策及內嵌政策之間選擇，請參閱《IAM 使用者指南》中的在受管政策和內嵌政策間選擇。

資源型政策

資源型政策是連接到資源的 JSON 政策文件。資源型政策的最常見範例是 IAM 角色信任政策和 Amazon S3 儲存貯體政策。在支援資源型政策的服務中，服務管理員可以使用它們來控制對特定資源的存取權限。對於附加政策的資源，政策會定義指定的主體可以對該資源執行的動作以及在何種條件下執行的動作。您必須在資源型政策中指定主體。委託人可以包含帳戶、使用者、角色、聯合身分使用者或 AWS 服務。

資源型政策是位於該服務中的內嵌政策。您無法在資源型政策中使用來自 IAM 的 AWS 受管政策。

存取控制清單 (ACL)

存取控制清單 (ACL) 可控制哪些主體 (帳戶成員、使用者或角色) 擁有存取某資源的許可。ACL 類似於資源型政策，但它們不使用 JSON 政策文件格式。

Amazon S3 AWS WAF和 Amazon VPC 是支援 ACLs的服務範例。如需進一步了解 ACL，請參閱 Amazon Simple Storage Service 開發人員指南中的存取控制清單 (ACL) 概觀。

其他政策類型

AWS 支援其他較不常見的政策類型。這些政策類型可設定較常見政策類型授予您的最大許可。

- 許可界限 – 許可範圍是一種進階功能，可供您設定身分型政策能授予 IAM 實體 (IAM 使用者或角色) 的最大許可。您可以為實體設定許可界限。所產生的許可會是實體的身分型政策和其許可界限的交集。會在 `Principal` 欄位中指定使用者或角色的資源型政策則不會受到許可界限限制。所有這類政策中的明確拒絕都會覆寫該允許。如需許可界限的詳細資訊，請參閱 IAM 使用者指南中的 IAM 實體許可界限。
- 服務控制政策 SCPs) – SCPs是 JSON 政策，可指定 中組織或組織單位 (OU) 的最大許可 AWS Organizations。 AWS Organizations 是一種服務，用於分組和集中管理您企業擁有 AWS 帳戶 的多個 。若您啟用組織中的所有功能，您可以將服務控制政策 (SCP) 套用到任何或所有帳戶。SCP 會限制成員帳戶中實體的許可，包括每個實體 AWS 帳戶根使用者。如需 Organizations 和 SCP 的詳細資訊，請參閱《AWS Organizations 使用者指南》中的服務控制政策。
- 資源控制政策 (RCP) - RCP 是 JSON 政策，可用來設定您帳戶中資源的可用許可上限，採取這種方式就不需要更新附加至您所擁有的每個資源的 IAM 政策。RCP 會限制成員帳戶中資源的許可，並可能影響身分的有效許可，包括 AWS 帳戶根使用者，無論它們是否屬於您的組織。如需 Organizations 和 RCPs的詳細資訊，包括 AWS 服務 支援 RCPs 清單，請參閱AWS Organizations《 使用者指南》中的資源控制政策 (RCPs)。
- 工作階段政策 – 工作階段政策是一種進階政策，您可以在透過撰寫程式的方式建立角色或聯合使用者的暫時工作階段時，做為參數傳遞。所產生工作階段的許可會是使用者或角色的身分型政策和工作階段政策的交集。許可也可以來自資源型政策。所有這類政策中的明確拒絕都會覆寫該允許。如需詳細資訊，請參閱 IAM 使用者指南中的工作階段政策。

多種政策類型

將多種政策類型套用到請求時，其結果形成的許可會更為複雜、更加難以理解。若要了解如何 AWS 決定是否在涉及多個政策類型時允許請求，請參閱《IAM 使用者指南》中的政策評估邏輯。

AWS 服務 如何使用 IAM

若要全面了解 如何使用 AWS 服務 大多數 IAM 功能，請參閱《IAM 使用者指南》中的AWS 使用 IAM 的 服務。

若要了解如何 AWS 服務 搭配 IAM 使用特定 ，請參閱相關服務使用者指南的安全章節。

對 AWS 身分和存取進行故障診斷

使用以下資訊來協助您診斷和修正使用 AWS 和 IAM 時可能遇到的常見問題。

主題

- 我無權在 中執行動作 AWS
- 我未獲得執行 iam:PassRole 的授權
- 我想要允許 以外的人員 AWS 帳戶 存取我的 AWS 資源

我無權在 中執行動作 AWS

如果您收到錯誤，告知您未獲授權執行動作，您的政策必須更新，允許您執行動作。

下列範例錯誤會在mateojackson IAM 使用者嘗試使用主控台檢視一個虛構 *my-example-widget* 資源的詳細資訊，但卻無虛構 awes:*GetWidget* 許可時發生。

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 awes:GetWidget on resource: my-example-widget
```

在此情況下，必須更新 mateojackson 使用者的政策，允許使用 awes:*GetWidget* 動作存取 *my-example-widget* 資源。

如果您需要協助，請聯絡您的 AWS 管理員。您的管理員提供您的簽署憑證。

我未獲得執行 iam:PassRole 的授權

如果您收到錯誤，告知您未獲授權執行 iam:PassRole 動作，您的政策必須更新，允許您將角色傳遞給 AWS。

有些 AWS 服務 可讓您將現有角色傳遞給該服務，而不是建立新的服務角色或服務連結角色。如需執行此作業，您必須擁有將角色傳遞至該服務的許可。

名為 marymajor 的 IAM 使用者嘗試使用主控台在 AWS中執行動作時，發生下列範例錯誤。但是，動作要求服務具備服務角色授予的許可。Mary 沒有將角色傳遞至該服務的許可。

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

在這種情況下，Mary 的政策必須更新，允許她執行 iam:PassRole 動作。

如果您需要協助，請聯絡您的 AWS 管理員。您的管理員提供您的簽署憑證。

我想要允許 以外的人員 AWS 帳戶 存取我的 AWS 資源

您可以建立一個角色，讓其他帳戶中的使用者或您組織外部的人員存取您的資源。您可以指定要允許哪些信任物件取得該角色。針對支援基於資源的政策或存取控制清單 (ACL) 的服務，您可以使用那些政策來授予人員存取您的資源的許可。

如需進一步了解，請參閱以下內容：

- 若要了解 是否 AWS 支援這些功能，請參閱 AWS 服務 如何使用 IAM。
- 若要了解如何 AWS 帳戶 在您擁有的 資源之間提供存取權，請參閱《IAM 使用者指南》中的在您擁有 AWS 帳戶 的另一個 IAM 使用者中提供存取權。
- 若要了解如何將資源的存取權提供給第三方 AWS 帳戶，請參閱《IAM 使用者指南》中的將存取權提供給第三方 AWS 帳戶 擁有。
- 如需了解如何透過聯合身分提供存取權，請參閱 IAM 使用者指南中的將存取權提供給在外部進行身分驗證的使用者 (聯合身分)。
- 如需了解使用角色和資源型政策進行跨帳戶存取之間的差異，請參閱《IAM 使用者指南》中的 IAM 中的跨帳戶資源存取。

此 AWS 產品或服務的合規驗證

若要了解 是否 AWS 服務 在特定合規計劃的範圍內，請參閱AWS 服務 合規計劃範圍內然後選擇您感興趣的合規計劃。如需一般資訊，請參閱 AWS Compliance Programs。

您可以使用 下載第三方稽核報告 AWS Artifact。如需詳細資訊，請參閱下載報告 in AWS Artifact

您使用 時的合規責任 AWS 服務 取決於資料的機密性、您公司的合規目標，以及適用的法律和法規。AWS 提供下列資源來協助合規：

- 安全合規與治理 - 這些解決方案實作指南內容討論了架構考量，並提供部署安全與合規功能的步驟。
- HIPAA 合格服務參考 – 列出 HIPAA 合格服務。並非所有 AWS 服務 都符合 HIPAA 資格。
- AWS 合規資源 – 此工作手冊和指南集合可能適用於您的產業和位置。
- AWS 客戶合規指南 – 透過合規的角度了解共同責任模型。本指南摘要說明跨多個架構 （包括國家標準技術研究所 (NIST)、支付卡產業安全標準委員會 (PCI) 和國際標準化組織 (ISO)) 保護 AWS 服務和映射指南至安全控制的最佳實務。
- 《 AWS Config 開發人員指南》中的使用 規則評估資源 – AWS Config 服務會評估資源組態符合內部實務、產業準則和法規的程度。
- AWS Security Hub – 這 AWS 服務 可讓您全面檢視其中的安全狀態 AWS。Security Hub 使用安全控制，可評估您的 AWS 資源並檢查您的法規遵循是否符合安全業界標準和最佳實務。如需支援的服務和控制清單，請參閱「Security Hub 控制參考」。
- Amazon GuardDuty – 這會監控您的環境是否有可疑和惡意活動，以 AWS 服務 偵測對您 AWS 帳戶、工作負載、容器和資料的潛在威脅。GuardDuty 可滿足特定合規架構所規定的入侵偵測需求，以協助您因應 PCI DSS 等各種不同的合規需求。
- AWS Audit Manager – 這 AWS 服務 可協助您持續稽核 AWS 用量，以簡化您管理風險的方式，以及符合法規和業界標準的方式。

此 AWS 產品或服務會透過其支援的特定 Amazon Web Services (AWS) 服務，遵循共同責任模型。如需 AWS 服務安全資訊，請參閱AWS 服務安全文件頁面AWS ，以及合規計劃在 AWS 合規工作範圍內的服務。

此 AWS 產品或服務的彈性

AWS 全球基礎設施是以 AWS 區域 和可用區域為基礎建置。

AWS 區域 提供多個實體隔離和隔離的可用區域，這些可用區域與低延遲、高輸送量和高備援聯網連接。

透過可用區域，您可以設計與操作的應用程式和資料庫，在可用區域之間自動容錯移轉而不會發生中斷。可用區域的可用性、容錯能力和擴展能力，均較單一或多個資料中心的傳統基礎設施還高。

如需 AWS 區域和可用區域的詳細資訊，請參閱 AWS 全球基礎設施。

此 AWS 產品或服務會透過其支援的特定 Amazon Web Services (AWS) 服務，遵循共同責任模型。如需 AWS 服務安全資訊，請參閱AWS 服務安全文件頁面AWS ，以及合規計劃在 AWS 合規工作範圍內的服務。

此 AWS 產品或服務的基礎設施安全

此 AWS 產品或服務使用 受管服務，因此受到 全球網路安全的 AWS 保護。如需 AWS 安全服務以及如何 AWS 保護基礎設施的相關資訊，請參閱AWS 雲端安全。若要使用基礎設施安全最佳實務來設計您的 AWS 環境，請參閱安全支柱 AWS Well-Architected Framework 中的基礎設施保護。

您可以使用 AWS 發佈的 API 呼叫，透過網路存取此 AWS 產品或服務。使用者端必須支援下列專案：

- Transport Layer Security (TLS)。我們需要 TLS 1.2 並建議使用 TLS 1.3。
- 具備完美轉送私密(PFS)的密碼套件，例如 DHE (Ephemeral Diffie-Hellman)或 ECDHE (Elliptic Curve Ephemeral Diffie-Hellman)。現代系統(如 Java 7 和更新版本)大多會支援這些模式。

此外，請求必須使用存取金鑰 ID 和與 IAM 主體相關聯的私密存取金鑰來簽署。或者，您可以透過 AWS Security Token Service (AWS STS) 來產生暫時安全憑證來簽署請求。

此 AWS 產品或服務會透過其支援的特定 Amazon Web Services (AWS) 服務，遵循共同責任模型。如需 AWS 服務安全性資訊，請參閱AWS 服務安全性文件頁面AWS ，以及合規計劃在 AWS 合規工作範圍內的服務。

在 中強制執行最低 TLS 版本 適用於 .NET 的 AWS SDK

若要在與 AWS 服務通訊時提高安全性，您應該 適用於 .NET 的 AWS SDK 將 設定為使用 TLS 1.2 或更新版本。

適用於 .NET 的 AWS SDK 使用基礎 .NET 執行期來決定要使用的安全通訊協定。根據預設，.NET 的目前版本會使用作業系統支援之最新設定的通訊協定。應用程式可覆寫此開發套件行為，但「不建議」這麼做。

.NET Core

根據預設，.NET Core 會使用作業系統支援之最新設定的通訊協定。 適用於 .NET 的 AWS SDK 未提供覆寫此行為的機制。

如果您是使用比 2.1 更舊的 .NET Core 版本，「強烈」建議您升級 .NET Core 版本。

請參閱下列各個作業系統的特定資訊。

Windows

Windows 的現代化分發會依預設啟用 TLS 1.2 支援。如果您在 Windows 7 SP1 或 Windows Server 2008 R2 SP1 上執行 ，您需要確保已在登錄檔中啟用 TLS 1.2 支援，如 https：//https://learn.microsoft.com/en-us/windows-server/security/tls/tls-registry-settings#tls-12 所述。如果執行的是較舊的分發，則必須升級作業系統。如需有關 Windows 中 TLS 1.3 支援的資訊，請查看最新的 Microsoft 文件，了解最低必要用戶端或伺服器版本。

macOS

如果執行的是 .NET Core 2.1 或更新版本，會依預設啟用 TLS 1.2。OS X Mavericks v10.9 或更新版本支援 TLS 1.2。.NET Core 2.1 版和更新版本需要更新版本的 macOS，如 https：//https://learn.microsoft.com/en-us/dotnet/core/install/windows?tabs=net80&pivots=os-macos 所述。

如果您使用的是 .NET Core 1.0，.NET Core 會在 macOS 上使用 OpenSSL，這是必須另外安裝的相依項目。OpenSSL 在 1.0.1 版中新增了對 TLS 1.2 的支援，並在 1.1.1 版中新增了對 TLS 1.3 的支援。

Linux

Linux 上的 .NET Core 需要使用許多 Linux 分發隨附綁定的 OpenSSL，但也可以另外個別安裝。OpenSSL 在 1.0.1 版中新增了對 TLS 1.2 的支援，並在 1.1.1 版中新增了對 TLS 1.3 的支援。如果您是使用 .NET Core 的現代化版本 (2.1 或更新版本)，且已安裝套件管理員，可能已為您安裝 OpenSSL 更現代化的版本。

若要確認，您可以在終端機視窗中執行 **`openssl version`**，然後驗證版本確實比 1.0.1 更新。

.NET Framework

如果您執行的是 .NET Framework 的現代化版本 (4.7 或更新版本) 以及 Windows 的現代化版本 (至少是適用於用戶端的 Windows 8、適用於伺服器的 Windows Server 2012 或更新版本)，會依預設啟用並使用 TLS 1.2。

如果您使用的是未使用作業系統設定的 .NET Framework 執行期 (.NET Framework 3.5 到 4.5.2)， 適用於 .NET 的 AWS SDK 會嘗試將 TLS 1.1 和 TLS 1.2 的支援新增至支援的通訊協定。如果您使用的是 .NET Framework 3.5，這只有在已安裝適當的熱修補程式時才會成功，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， 適用於 .NET 的 AWS SDK 結束對 .NET Framework 3.5 的支援，並將最低 .NET Framework 版本變更為 4.7.2。如需詳細資訊，請參閱部落格文章 .NET Framework 3.5 和 4.5 目標的重要變更 適用於 .NET 的 AWS SDK。

如果您的應用程式在 Windows 7 SP1 或 Windows Server 2008 R2 SP1 上較新的 .NET Framework 上執行，您需要確保已在登錄檔中啟用 TLS 1.2 支援，如 https：//https://learn.microsoft.com/en-us/windows-server/security/tls/tls-registry-settings#tls-12 所述。Windows 更新版本會依預設啟用該支援。

如需搭配 .NET Framework 使用 TLS 的詳細最佳實務，請參閱 Microsoft 文章，網址為 https：//https://learn.microsoft.com/en-us/dotnet/framework/network-programming/tls。

AWS Tools for PowerShell

AWS Tools for PowerShell 使用 適用於 .NET 的 AWS SDK 進行對 AWS 服務的所有呼叫。環境的行為取決於執行的 Windows PowerShell 版本，如下所示。

經由 5.x 的 Windows PowerShell 2.0

經由 5.x 的 Windows PowerShell 2.0 可在 .NET Framework 上執行。您可以使用下列命令，驗證 PowerShell 使用的是哪個 .NET 執行時間 (2.0 或 4.0)。

```
$PSVersionTable.CLRVersion
```

- 若使用 .NET 執行時間 2.0，請按照上述有關 適用於 .NET 的 AWS SDK 和 .NET Framework 3.5 的指示進行。

  > **Warning**
  >
  > 2024 年 8 月 15 日， 適用於 .NET 的 AWS SDK 結束對 .NET Framework 3.5 的支援，並將最低 .NET Framework 版本變更為 4.7.2。如需詳細資訊，請參閱部落格文章 .NET Framework 3.5 和 4.5 目標的重要變更 適用於 .NET 的 AWS SDK。

- 若使用 .NET 執行時間 4.0，請按照上述有關 適用於 .NET 的 AWS SDK 和 .NET Framework 4+ 的指示進行。

Windows PowerShell 6.0

Windows PowerShell 6.0 和更新版本會在 .NET Core 上執行。您可以執行下列命令，驗證 .NET Core 使用的是哪個版本。

```
[System.Reflection.Assembly]::GetEntryAssembly().GetCustomAttributes([System.Runtime.Versioning
 $true).FrameworkName
```

請遵循先前提供的有關 適用於 .NET 的 AWS SDK 和相關 .NET Core 版本的指示。

Xamarin

如需 Xamarin，請參閱 https：//https://learn.microsoft.com/en-us/xamarin/cross-platform/app-fundamentals/transport-layer-security。綜上所述：

適用於 Android

- 需要 Android 5.0 或更新版本。
- Project Properties (專案屬性)、Android Options (Android 選項)：HttpClient 實作必須設為 Android 且 SSL/TLS 實作必須設為 Native TLS 1.2+ (原生 TLS 1.2+)。

適用於 iOS

- 需要 iOS 7 或更新版本。
- Project Properties (專案屬性)、iOS Build (iOS 組建)：HttpClient 實作必須設為 NSUrlSession。

適用於 macOS

- 需要 macOS 10.9 或更新版本。
- Project Options (專案選項)、Build (組建)、Mac Build (Mac 組建)：HttpClient 實作必須設為 NSUrlSession。

Unity

您必須使用 Unity 2018.2 或更新版本，並使用等同於 .NET 4.x 的指令碼編寫執行時間。您可以在 Project Settings (專案設定)、Configuration (組態)、Player (播放器) 中設定此選項，如 https://docs.unity3d.com/2019.1/Documentation/Manual/ScriptingRuntimeUpgrade.html 所述。等同於 .NET 4.x 的指令碼編寫執行時間會對執行 Mono or IL2CPP 的所有 Unity 平台啟用 TLS 1.2 支援。

瀏覽器 (適用於 Blazor WebAssembly)

WebAssembly 會在瀏覽器 (而不是伺服器) 中執行，並使用瀏覽器來處理 HTTP 流量。因此，TLS 支援是取決於瀏覽器支援。

Blazor WebAssembly 在 ASP.NET Core 3.1 預覽版中僅支援支援 WebAssembly 的瀏覽器，如 https：//https://learn.microsoft.com/en-us/aspnet/core/blazor/supported-platforms 中所述。所有主流瀏覽器在支援 WebAssembly 之前均已支援 TLS 1.2。如果您瀏覽器的情況是像這樣，則在應用程式執行時，該應用程式就能透過 TLS 1.2 進行通訊。

若要取得詳細資訊並進行確認，請參閱瀏覽器的文件。

Amazon S3 加密用戶端遷移

本主題說明如何將應用程式從 Amazon Simple Storage Service (Amazon S3) 加密用戶端的第 1 版 (V1) 遷移到第 2 版 (V2)，並確保在整個遷移過程中的應用程式可用性。

使用 V2 用戶端加密的物件無法使用 V1 用戶端解密。為了簡化遷移到新用戶端，而無需一次重新加密所有物件，已提供「V1-transitional」用戶端。此用戶端可以解密 V1 和 V2-encrypted物件，但只會以 V1-compatible格式加密物件。V2 用戶端可以解密 V1 和 V2-encrypted物件 （ 啟用 V1 物件時 ），但只會以 V2-compatible格式加密物件。

遷移概觀

此遷移分三個階段進行。此處會介紹這些階段，稍後會詳細說明。在下一個階段開始之前，所有使用共用物件的用戶端都必須完成每個階段。

1. 將現有用戶端更新為 V1-transitional 用戶端以讀取新格式。首先，更新您的應用程式，以對 V1-transitional 用戶端取代 V1 用戶端取得相依性。V1-transitional用戶端可讓您現有的程式碼解密由新的 V2 用戶端寫入的物件，以及V1-compatible格式寫入的物件。

   > **Note**
   >
   > V1-transitional 用戶端僅供遷移之用。移至 V2V1-transitional2 用戶端。

2. 將 V1-transitional用戶端遷移至 V2 用戶端以寫入新格式。接著，使用 V1-transitional用戶端，並將安全性設定檔設定為 `V2AndLegacy`。 V2 在 V2 用戶端上設定此安全性設定檔可讓這些用戶端解密以 V1-compatible格式加密的物件。

3. 將 V2 用戶端更新為不再讀取 V1 格式。最後，在所有用戶端都遷移至 V2 且所有物件都以 V2-compatible格式加密或重新加密後，請將 V2 安全設定檔設定為 ，V2而不是 V2AndLegacy。這可防止解密 V1-compatible格式的物件。

將現有用戶端更新為 V1-transitional用戶端以讀取新格式

V2 加密用戶端使用舊版用戶端不支援的加密演算法。遷移的第一步是更新您的 V1 解密用戶端，以便他們可以讀取新格式。

V1-transitional用戶端可讓您的應用程式解密 V1 和 V2-encrypted物件。此用戶端是 Amazon.Extensions.S3.Encryption NuGet 套件的一部分。在每個應用程式上執行下列步驟，以使用 V1-transitional用戶端。

1. 對 Amazon.Extensions.S3.Encryption 套件採取新的相依性。如果您的專案直接依賴 AWSSDK.S3 或 AWSSDK.KeyManagementService 套件，您必須更新或移除這些相依性，以便使用此新套件提取其更新版本。
2. 將適當的`using`陳述式從 `Amazon.S3.Encryption`變更為 `Amazon.Extensions.S3.Encryption`，如下所示：

```
// using Amazon.S3.Encryption;
  using Amazon.Extensions.S3.Encryption;
```

3. 重建並重新部署您的應用程式。

V1-transitional用戶端與 V1 用戶端完全 API 相容，因此不需要其他程式碼變更。

將 V1-transitional用戶端遷移至 V2 用戶端以寫入新格式

V2 用戶端是 Amazon.Extensions.S3.Encryption NuGet 套件的一部分。它可讓您的應用程式解密 V1 和 V2-encrypted的物件（如果設定為這樣做），但只會以 V2-compatible格式加密物件。

更新現有用戶端以讀取新的加密格式後，您可以繼續安全地將應用程式更新為 V2 加密和解密用戶端。在每個應用程式上執行下列步驟，以使用 V2 用戶端：

1. 將 `EncryptionMaterials` 變更為 `EncryptionMaterialsV2`。
    a. 使用 KMS 時：
        i. 提供 KMS 金鑰 ID。

ii. 宣告您正在使用的加密方法，也就是 KmsType.KmsContext。

iii. 提供加密內容給 KMS，以與此資料金鑰建立關聯。您可以傳送空白字典 (Amazon 加密內容仍會合併），但建議提供額外的內容。

b. 使用使用者提供的金鑰包裝方法（對稱或非對稱加密）時：

i. 提供包含加密資料的 AES或 RSA執行個體。

ii. 宣告要使用的加密演算法，也就是 SymmetricAlgorithmType.AesGcm或 AsymmetricAlgorithmType.RsaOaepSha1。

2. AmazonS3CryptoConfiguration 將 SecurityProfile 屬性設定為 AmazonS3CryptoConfigurationV2時，將 變更為 SecurityProfile.V2AndLegacy。

3. 將 AmazonS3EncryptionClient 變更為 AmazonS3EncryptionClientV2。此用戶端會採用先前步驟中新轉換的 AmazonS3CryptoConfigurationV2和 EncryptionMaterialsV2 物件。

範例：KMS 到 KMS+Context

預遷移

```
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);
```

遷移後

```
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);
```

範例：對稱演算法 (AES-CBC 到 AES-GCM 金鑰包裝)

StorageMode 可以是 ObjectMetadata 或 InstructionFile。

預遷移

```
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);
```

遷移後

```
using System.Security.Cryptography;
using Amazon.Extensions.S3.Encryption;
using Amazon.Extensions.S3.Encryption.Primitives;

var symmetricAlgorithm = Aes.Create();
var encryptionMaterial = new EncryptionMaterialsV2(symmetricAlgorithm,
 SymmetricAlgorithmType.AesGcm);
var configuration = new AmazonS3CryptoConfigurationV2(SecurityProfile.V2AndLegacy)
{
    StorageMode = CryptoStorageMode.ObjectMetadata
};
var encryptionClient = new AmazonS3EncryptionClientV2(configuration,
 encryptionMaterial);
```

> **Note**
>
> 使用 AES-GCM 解密時，請先將整個物件讀到結尾，再開始使用解密的資料。這是為了驗證物件在加密後尚未修改。

範例：非對稱演算法 (RSA 到 RSA-OAEP-SHA1 金鑰包裝)

`StorageMode` 可以是 `ObjectMetadata` 或 `InstructionFile`。

預遷移

```
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);
```

遷移後

```
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);
```

將 V2 用戶端更新為不再讀取 V1 格式

最後，所有物件都將使用 V2 用戶端進行加密或重新加密。此轉換完成後，您可以透過將 SecurityProfile 屬性設定為 來停用 V2 用戶端中的 V1 相容性SecurityProfile.V2，如下列程式碼片段所示。 V2

```
//var configuration = new AmazonS3CryptoConfigurationV2(SecurityProfile.V2AndLegacy);
var configuration = new AmazonS3CryptoConfigurationV2(SecurityProfile.V2);
```

的特殊考量 適用於 .NET 的 AWS SDK

本節包含一般組態或程序不適當或不足的特殊情況考量。

主題

- 取得 的組件 適用於 .NET 的 AWS SDK
- 在應用程式中存取登入資料和設定檔
- Unity 支援的特殊考量
- Xamarin 支援的特殊考量

取得 的組件 適用於 .NET 的 AWS SDK

本主題說明如何取得 AWSSDK 組件，並將其存放在本機（或內部部署）以供專案使用。這不是處理 SDK 參考的建議方法，但在某些環境中是必要的。

> **Note**
>
> 處理 SDK 參考的建議方法是僅下載並安裝每個專案所需的 NuGet 套件。該方法在 中描述使用 NuGet 安裝 AWSSDK 套件。

如果您無法或無法依專案下載和安裝 NuGet 套件，您可以使用下列選項。

下載並擷取 ZIP 檔案

（請記住，這不是處理 參考的建議方法 適用於 .NET 的 AWS SDK。）

1. 下載下列其中一個 ZIP 檔案：
   - aws-sdk-net8.0.zip - 支援 .NET 8 和更新版本的組件。
   - aws-sdk-netcoreapp3.1.zip - 支援 .NET Core 3.1 和更新版本的組件。
   - aws-sdk-netstandard2.0.zip - 支援 .NET Standard 2.0 和 2.1 的組件。
   - aws-sdk-net472.zip - 支援 .NET Framework 4.5 和更新版本的組件。
2. 將組件解壓縮到檔案系統上的一些「下載」資料夾；位置並不重要。請記下此資料夾。
3. 當您設定專案時，您可以從此資料夾取得必要的組件，如中所述安裝不含 NuGet 的 AWSSDK 組件。

在應用程式中存取登入資料和設定檔

使用登入資料的偏好方法是允許 為您 適用於 .NET 的 AWS SDK 尋找和擷取登入資料，如 中所述憑證和設定檔解析。

不過，您也可以設定應用程式主動擷取設定檔和登入資料，然後在建立 AWS 服務用戶端時明確使用這些登入資料。

若要主動擷取設定檔和登入資料，請使用 Amazon.Runtime.CredentialManagement 命名空間中的類別。

- 若要在使用 AWS 登入資料檔案格式的檔案中尋找設定檔 (其預設位置中的共用 AWS 登入資料檔案或自訂登入資料檔案），請使用 SharedCredentialsFile 類別。為了簡潔起見，此格式的檔案有時會在此文字中簡單地稱為登入資料檔案。
- 若要在 SDK Store 中尋找設定檔，請使用 NetSDKCredentialsFile 類別。
- 若要在登入資料檔案和 SDK Store 中搜尋，根據類別屬性的組態，請使用 CredentialProfileStoreChain 類別。

  您可以使用此類別來尋找設定檔。您也可以使用此類別直接請求 AWS 登入資料，而不是使用`AWSCredentialsFactory`類別 （接下來說明）。
- 若要從設定檔擷取或建立各種類型的登入資料，請使用 AWSCredentialsFactory 類別。

下列各節提供這些類別的範例。

類別 CredentialProfileStoreChain 的範例

您可以使用 TryGetAWSCredentials 或 TryGetProfile 方法，從 CredentialProfileStoreChain 類別取得登入資料或設定檔。類別的 `ProfilesLocation` 屬性決定方法的行為，如下所示：

- 如果 `ProfilesLocation` 是 null 或空白，請在平台支援時搜尋 SDK Store，然後在預設位置搜尋共用的 AWS 登入資料檔案。
- 如果 `ProfilesLocation` 屬性包含值，請搜尋 屬性中指定的登入資料檔案。

從 SDK Store 或共用的登入資料檔案取得 AWS 登入資料

此範例說明如何使用 `CredentialProfileStoreChain`類別取得登入資料，然後使用登入資料來建立 AmazonS3Client 物件。登入資料可以來自軟體開發套件存放區，也可以來自預設位置的共用 AWS 登入資料檔案。

此範例也使用 Amazon.Runtime.AWSCredentials 類別。

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

從 SDK Store 或共用 AWS 登入資料檔案取得設定檔

此範例說明如何使用 CredentialProfileStoreChain 類別取得設定檔。登入資料可以來自軟體開發套件存放區，也可以來自預設位置的共用 AWS 登入資料檔案。

此範例也會使用 CredentialProfile 類別。

```
var chain = new CredentialProfileStoreChain();
CredentialProfile basicProfile;
if (chain.TryGetProfile("basic_profile", out basicProfile))
{
    // Use basicProfile
}
```

從自訂登入資料檔案取得登入資料

此範例說明如何使用 CredentialProfileStoreChain 類別取得登入資料。登入資料來自使用 AWS 登入資料檔案格式的檔案，但位於替代位置。

此範例也使用 Amazon.Runtime.AWSCredentials 類別。

```
var chain = new
    CredentialProfileStoreChain("c:\\Users\\sdkuser\\customCredentialsFile.ini");
AWSCredentials awsCredentials;
if (chain.TryGetAWSCredentials("basic_profile", out awsCredentials))
{
    // Use awsCredentials to create an AWS service client
```

```
}
```

SharedCredentialsFile 和 AWSCredentialsFactory 類別的範例

使用 SharedCredentialsFile 類別建立 AmazonS3Client

此範例說明如何在共用 AWS 登入資料檔案中尋找設定檔、從設定檔建立 AWS 登入資料，然後使用登入資料來建立 AmazonS3Client 物件。此範例使用 SharedCredentialsFile 類別。

此範例也會使用 CredentialProfile 類別和 Amazon.Runtime.AWSCredentials 類別。

```
CredentialProfile basicProfile;
AWSCredentials awsCredentials;
var sharedFile = new SharedCredentialsFile();
if (sharedFile.TryGetProfile("basic_profile", out basicProfile) &&
    AWSCredentialsFactory.TryGetAWSCredentials(basicProfile, sharedFile, out
 awsCredentials))
{
    // use awsCredentials to create an Amazon S3 service client
    using (var client = new AmazonS3Client(awsCredentials, basicProfile.Region))
    {
        var response = await client.ListBucketsAsync();
        Console.WriteLine($"Number of buckets: {response.Buckets.Count}");
    }
}
```

> **Note**
>
> NetSDKCredentialsFile 類別的使用方式完全相同，但您會執行個體化新的 NetSDKCredentialsFile 物件，而不是 SharedCredentialsFile 物件。

Unity 支援的特殊考量

為您的 Unity 應用程式使用 適用於 .NET 的 AWS SDK 和 .NET Standard 2.0 時，您的應用程式必須直接參考 適用於 .NET 的 AWS SDK 組件 (DLL 檔案），而不是使用 NuGet。有鑑於此需求，以下是您需要執行的重要動作。

- 您需要取得 適用於 .NET 的 AWS SDK 組件並將其套用至您的專案。如需如何執行此操作的詳細資訊，請參閱主題 下載並擷取 ZIP 檔案中的 取得 AWSSDK 組件。

- 您需要在 Unity 專案中包含下列 DLLs，以及 AWSSDK.Core 的 DLLs 和您正在使用的其他 AWS 服務。從 3.5.109 版開始 適用於 .NET 的 AWS SDK，.NET Standard ZIP 檔案包含這些額外的 DLLs。
  - Microsoft.Bcl.AsyncInterfaces.dll
  - System.Runtime.CompilerServices.Unsafe.dll
  - System.Threading.Tasks.Extensions.dll

- 如果您使用 IL2CPP 建置 Unity 專案，則必須將`link.xml`檔案新增至資產資料夾，以防止程式碼分割。`link.xml` 檔案必須列出您使用的所有 AWSSDK 組件，且每個組件都必須包含 `preserve="all"` 屬性。下列程式碼片段顯示此檔案的範例。

```
<linker>
    <assembly fullname="AWSSDK.Core" preserve="all"/>
    <assembly fullname="AWSSDK.DynamoDBv2" preserve="all"/>
    <assembly fullname="AWSSDK.Lambda" preserve="all"/>
</linker>
```

> **Note**
>
> 若要閱讀與此需求相關的有趣背景資訊，請參閱文章：https：//https://aws.amazon.com/blogs/developer/referencing-the-aws-sdk-for-net-standard-2-0-from-unity-xamarin-or-uwp/。

Xamarin 支援的特殊考量

Xamarin 專案 (全新和現有) 必須將目標鎖定於 .NET Standard 2.0。請參閱Xamarin.Forms 中的 .NET Standard 2.0 支援和 .NET 實作支援。

另請參閱可攜式類別程式庫和 Xamarin 的相關資訊。

的 API 參考 適用於 .NET 的 AWS SDK

適用於 .NET 的 AWS SDK 提供 API，您可用來存取 AWS 服務。若要查看 API 中有哪些類別和方法可用，請參閱 適用於 .NET 的 AWS SDK API 參考。

除了上述一般參考之外， 引導式程式碼範例區段下的每個範例都包含對該範例中使用的特定方法和類別的參考。

關於 API 參考版本

前述的 API 參考適用於 3.0 版和更新版本 適用於 .NET 的 AWS SDK。

如需從舊版 SDK 遷移的資訊，請參閱 遷移您的專案

若要尋找舊版 SDK API 參考的已棄用內容，請參閱下列項目 （以下項目）：

- 適用於 .NET 的 AWS SDK API 參考 V1 （已棄用）
- 適用於 .NET 的 AWS SDK API 參考 V2 （已棄用）

若要檢視已棄用的 適用於 .NET 的 AWS SDK API 參考，您需要擷取並設定 Web 瀏覽器。接下來顯示的指示是如何執行此操作的範例。它們是以在 Windows 系統上使用 Google Chrome Web 瀏覽器為基礎。將它們適應您的特定 Web 瀏覽器和作業系統。

檢視已棄用的 API 參考 V1

1. 下載已取代 適用於 .NET 的 AWS SDK API 參考 V1 的 ZIP 檔案。根據預設，ZIP 檔案名稱為 `sdkfornet-api-ref_v1_deprecated.zip`。該名稱將在這些說明中使用。
2. 將 ZIP 檔案放在您選擇的資料夾中。對於這些指示，資料夾名稱假設為 `C:\work\temp\api-refs\V1`。
3. 在 ZIP 檔案上按一下滑鼠右鍵，然後選擇全部解壓縮。接受預設位置，這`C:\work\temp\api-refs\V1\sdkfornet-api-ref_v1`適用於這些指示。
4. 在 `C:\work\temp\api-refs\V1` 資料夾中為 Google Chrome 建立捷徑。請小心不要移動原始應用程式。
5. 在新捷徑的屬性中，設定下列欄位：

- 目標： "*<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"

  針對 --user-data-dir引數，請使用適用於您環境的資料夾名稱。資料夾不必存在。
- 從 開始： C:\work\temp\api-refs\V1

6. 為捷徑指定適當的名稱。
7. 開啟捷徑以檢視舊的 API 參考。

檢視已棄用的 API 參考 V2

1. 下載已取代 適用於 .NET 的 AWS SDK API 參考 V2 的 ZIP 檔案。根據預設，ZIP 檔案名稱為 sdkfornet-api-ref_v2_deprecated.zip。該名稱將在這些說明中使用。
2. 將 ZIP 檔案放在您選擇的資料夾中。對於這些指示，資料夾名稱假設為 C:\work\temp\api-refs\V2。
3. 在 ZIP 檔案上按一下滑鼠右鍵，然後選擇全部擷取。接受預設位置，這C:\work\temp\api-refs\V2\sdkfornet-api-ref_v2適用於這些指示。
4. 在 C:\work\temp\api-refs\V2 資料夾中為 Google Chrome 建立捷徑。請小心不要移動原始應用程式。
5. 在新捷徑的屬性中，設定下列欄位：

   - 目標： "*<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"

     針對 --user-data-dir引數，請使用適用於您環境的資料夾名稱。資料夾不必存在。
   - 從 開始： C:\work\temp\api-refs\V2
6. 為捷徑指定適當的名稱。
7. 開啟捷徑以檢視舊的 API 參考。

文件歷史紀錄

下表說明自上次發行 適用於 .NET 的 AWS SDK 開發人員指南以來的重要變更。如需獲得此文件更新的通知，您可以訂閱 RSS 摘要。

變更	描述	日期
最新消息	已宣布 V3 的End-of-support 適用於 .NET 的 AWS SDK。	2025 年 8 月 21 日
最新消息	分散式快取提供者的一般可用性	2025 年 7 月 2 日
訊息處理架構	.NET AWS 的訊息處理架構已達到一般可用性！	2025 年 5 月 5 日
新功能	第 4 版 適用於 .NET 的 AWS SDK 已全面推出！如需有關將應用程式遷移至 V4 的資訊，請參閱遷移至版本 4。	2025 年 4 月 28 日
遷移至 第 4 版 適用於 .NET 的 AWS SDK	包含有關將使用 建置的應用程式遷移 適用於 .NET 的 AWS SDK 至 第 4 版的預覽資訊。	2025 年 3 月 3 日
在 中 AWS 與 .NET ACache 整合 適用於 .NET 的 AWS SDK	包含 中 AWS 如何與 .NET ACache 整合的相關資訊 適用於 .NET 的 AWS SDK。	2025 年 2 月 15 日
可觀測性	已宣布 GA 版本以提供可觀測性	2025 年 2 月 10 日
新功能	新增有關完整性保護新預設行為的資訊。	2025 年 1 月 15 日
新功能	新增第四個 第 4 適用於 .NET 的 AWS SDK 版預覽版本的相關資訊。	2024 年 11 月 15 日