개발자 가이드

AWS SDK for .NET (V4)

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

AWS SDK for .NET (V4): 개발자 가이드

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

Amazon 상표 및 트레이드 드레스는 Amazon 외 제품 또는 서비스와 함께 사용되어서는 안되며, 고객에게 혼동을 일으키거나 Amazon 브랜드 이미지를 떨어뜨리고 폄하하는 방식으로 이용할 수 없습니다. Amazon이 소유하지 않은 기타 모든 상표는 Amazon과 제휴 관계이거나 관련이 있거나 후원 관계와 관계없이 해당 소유자의 자산입니다.

Table of Contents

..... x
란 무엇입니까? AWS SDK for .NET 1
이 버전의 정보 1
SDK 메이저 버전에 대한 유지 관리 및 지원 2
일반 사용 사례 2
이 섹션의 추가 주제 2
관련 AWS 도구 3
Tools for Windows PowerShell 및 Tools for PowerShell Core 3
VS 코드용 도구 키트 3
Toolkit for Visual Studio 3
Azure DevOps용 도구 키트 4
지원하는 플랫폼 4
.NET Core 4
.NET Standard 2.0 4
.NET Framework 4.5 4
.NET Framework 3.5 4
휴대용 클래스 라이브러리 및 Xamarin 5
Unity 지원 5
SDK 및 도구 참조 5
수정 기록 5
새로운 기능 5
추가 리소스 7
시작 10
도구 체인 설치 및 구성 10
교차 플랫폼 개발 10
Visual Studio 및 .NET Core를 통한 Windows 11
다음 단계 11
를 사용하여 인증 AWS 11
콘솔 자격 증명을 사용하여 로그인 12
IAM Identity Center 사용 및 구성 13
IAM Identity Center를 사용하도록 SDK를 구성합니다. 13
AWS 액세스 포털 세션 시작 14
추가 정보 15
간단한 애플리케이션 생성 15

간단한 교차 플랫폼 앱 ... 16
간단한 Windows 기반 앱 ... 21
다음 단계 ... 27
구성 ... 28
새 프로젝트 시작 ... 28
NuGet을 사용하여 AWSSDK 패키지를 설치 ... 30
명령 프롬프트 또는 터미널에서 NuGet 사용 ... 30
Visual Studio 솔루션 탐색기에서 NuGet 사용 ... 31
Package Manager Console에서 NuGet 사용 ... 31
NuGet 없이 AWSSDK 어셈블리 설치 ... 32
AWS 리전 ... 33
특정 리전을 사용하여 서비스 클라이언트를 생성합니다. ... 34
모든 서비스 클라이언트의 리전 지정 ... 35
리전 해상도 ... 36
중국(베이징) 리전에 대한 특수 정보 ... 36
새 AWS 서비스에 대한 특별 정보 ... 37
보안 인증 정보 및 프로파일 확인 ... 37
프로필 해결 방법 ... 38
페더레이션 사용자 계정 보안 인증 사용 ... 38
역할 또는 임시 자격 증명 지정 ... 39
프록시 자격 증명 사용 ... 39
재시도 및 제한 시간 ... 40
재시도 ... 40
시간 초과 ... 42
관찰성 ... 44
추가 리소스 ... 44
TelemetryProvider 구성 ... 45
Metrics ... 46
원격 측정 공급자 ... 48
사용자 및 역할 ... 50
사용자 및 권한 집합 ... 50
서비스 역할 ... 50
고급 구성 ... 51
AWSSDK.Extensions.NETCore.Setup 및 IConfiguration ... 51
기타 애플리케이션 파라미터 구성 ... 56
에 대한 구성 파일 참조 AWS SDK for .NET ... 64

레거시 보안 인증 사용 76
보안 인증에 대한 중요 경고 및 지침 76
공유 AWS 자격 증명 파일 사용 77
SDK 스토어 사용(Windows만 해당) 81
SDK 사용 85
비동기 프로그래밍 85
페이지 매김 87
페이지네이터는 어디서 찾을 수 있나요? 87
페이지네이터는 무엇을 제공합니까? 87
동기 페이지 매김과 비동기 페이지 매김 비교 88
예제 88
페이지네이터에 관한 추가 고려 사항 92
HTTP 2 지원 92
추가 고려 사항 95
추가 도구 95
AWS 배포 도구 95
AWS .NET용 메시지 처리 프레임워크 95
.NET Aspire와의 통합 95
고급 인증 96
Single Sign-On 96
사전 조건 97
SSO 프로필 설정 97
SSO 토큰 생성 및 사용 99
추가 리소스 103
자습서 103
자습서: .NET 애플리케이션 전용 104
자습서: AWS CLI 및 .NET 애플리케이션 112
에 배포 AWS 121
.NET CLI에서 배포 121
IDE 툴킷에서 배포 121
사용 사례 122
ASP.NET Core 앱 122
.NET 콘솔 앱 123
Blazor WebAssembly 앱 124
AWS Lambda 프로젝트 124
사전 조건 124

사용 가능한 Lambda 명령 125
배포 단계 125
AWS 서비스 호출 127
가이드 코드 예제 127
CloudFormation 128
Amazon Cognito 132
DynamoDB 140
Amazon EC2 169
IAM 230
Amazon S3 249
Amazon SNS 259
Amazon SQS 263
AWS Lambda 296
API 296
사전 조건 296
추가 정보 296
주제 296
Lambda 주석 297
개요 수준 라이브러리 및 프레임워크 298
메시지 처리 프레임워크 299
.NET Aspire AWS 와 통합 318
기타 서비스 및 구성 320
코드 예제 321
Aurora 322
시작하기 322
기본 사항 324
작업 348
Auto Scaling 363
시작 322
기본 사항 324
작업 348
Amazon Bedrock 398
시작하기 322
작업 348
Amazon Bedrock 런타임 402
Anthropic Claude 403

Cohere Command 408
Meta Llama 412
Mistral AI 416
CloudFormation 419
시작하기 322
CloudWatch 423
시작하기 322
기본 사항 324
작업 348
CloudWatch Logs 480
작업 348
시나리오 484
Amazon Cognito 자격 증명 공급자 506
작업 348
AWS Control Tower 507
시작하기 322
기본 사항 324
작업 348
DynamoDB 546
시작하기 322
기본 사항 324
작업 348
Amazon EC2 587
시작하기 322
Amazon ECS 589
시작하기 322
AWS IoT 591
시작하기 322
기본 사항 324
작업 348
AWS IoT data 650
작업 348
Amazon Redshift 653
시작하기 322
기본 사항 324
작업 348

Amazon S3 ... 683
시작하기 ... 322
기본 사항 ... 324
작업 ... 348
시나리오 ... 484
프로젝트 마이그레이션 ... 719
버전 4로 마이그레이션 ... 719
.NET Framework ... 719
값 유형 ... 719
컬렉션 ... 720
AWS Security Token Service (STS) ... 720
와 관련된 변경 사항 `ClientConfig` ... 721
AWSSDK.Extensions.NETCore.Setup NuGet 패키지 ... 721
`CookieSigner` 및 `UrlSigner` ... 722
DateTime과 UTC DateTime 비교 ... 722
DateTime 구문 분석 ... 723
`ConvertFromUnixEpochSeconds` 및 `ConvertFromUnixEpochMilliseconds` ... 723
로깅 ... 724
HTTP 2 지원 ... 725
Single Sign-On ... 725
DynamoDB 관련 변경 사항 ... 725
EC2 관련 변경 사항 ... 728
S3 관련 변경 사항 ... 729
제거된 프로그래밍 요소 ... 732
.NET Standard 1.3에서 마이그레이션 ... 735
보안 ... 737
데이터 보호 ... 737
자격 증명 및 액세스 관리 ... 738
대상 ... 739
ID를 통한 인증 ... 739
정책을 사용하여 액세스 관리 ... 741
에서 IAM AWS 서비스 을 사용하는 방법 ... 742
AWS 자격 증명 및 액세스 문제 해결 ... 742
규정 준수 검증 ... 744
복원력 ... 744
인프라 보안 ... 745

최소 TLS 버전 적용 ... 746
.NET Core ... 746
.NET Framework ... 747
AWS Tools for PowerShell ... 748
Xamarin ... 748
Unity ... 749
브라우저(Blazor WebAssembly용) ... 749
S3 암호화 클라이언트 마이그레이션(V1에서 V2로) ... 750
마이그레이션 개요 ... 750
새 형식을 읽으려면 기존 클라이언트를 V1 전환 클라이언트로 업데이트 ... 751
V1 전환 클라이언트를 V2 클라이언트로 마이그레이션하여 새 형식 작성 ... 751
V1 형식을 더 이상 읽지 않도록 V2 클라이언트 업데이트 ... 755
S3 암호화 클라이언트 마이그레이션(V2에서 V4로) ... 755
V4 개념 이해 ... 755
V4 형식을 읽도록 기존 클라이언트 업데이트 ... 757
V4 클라이언트로 마이그레이션 ... 758
추가 구성 예제 ... 761
특별 고려 사항 ... 764
AWSSDK 어셈블리 가져오기 ... 764
ZIP 파일 다운로드 및 압축 ... 764
애플리케이션에서 자격 증명 및 프로필에 액세스 ... 765
클래스 CredentialProfileStoreChain의 예 ... 765
클래스 SharedCredentialsFile 및 AWSCredentialsFactory의 예 ... 767
Unity 지원 ... 768
Xamarin 지원 ... 769
API 참조 ... 770
API 참조 버전 정보 ... 770
문서 기록 ... 773

버전 4(V4) AWS SDK for .NET 가 릴리스되었습니다.

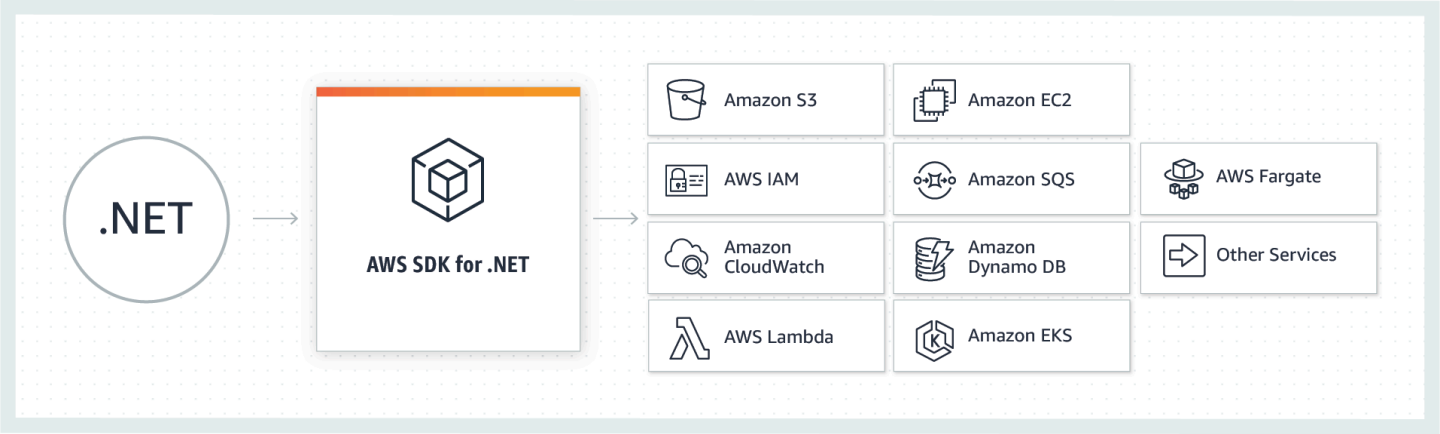
변경 사항 해제 및 애플리케이션 마이그레이션에 대한 자세한 내용은 마이그레이션 주제를 참조하세요.

Click here for details

기계 번역으로 제공되는 번역입니다. 제공된 번역과 원본 영어의 내용이 상충하는 경우에는 영어 버전이 우선합니다.

란 무엇입니까? AWS SDK for .NET

를 AWS SDK for .NET 사용하면 Amazon Simple Storage Service(Amazon S3 Amazon EC2)와 같은 비용 효율적이고 확장 가능하며 안정적인 AWS 서비스를 활용하는 .NET 애플리케이션을 더 쉽게 구축할 수 있습니다. SDK는 .NET 개발자에게 일관되고 친숙한 라이브러리 세트를 제공하여 AWS 서비스 사용을 간소화합니다.



(좋습니다! 를 설정하고 빠른 둘러보기를 할 준비가 되었습니다.)

이 버전의 정보

현재 및 과거 설명서

이 설명서는의 버전 4.0 이상(V4)에 대한 것입니다 AWS SDK for .NET. 주로 .NET Core와 ASP.NET Core를 중심으로 작성되지만, .NET Framework 및 ASP.NET 4.x에 대한 정보도 포함되어 있습니다. Windows 및 Visual Studio 외에도 크로스 플랫폼 개발도 동등하게 고려합니다.

SDK의 이전 버전에서 마이그레이션하는 방법에 대한 자세한 내용은 섹션을 참조하세요프로젝트 마이그레이션.

향후 더 이상 사용되지 않을 이전 버전의 SDK V3에 대한 자세한 내용은 AWS SDK for .NET (V3) 개발자 안내서를 참조하세요.

SDK의 이전 버전에서 더 이상 사용되지 않는 콘텐츠를 찾으려면 다음 항목(들)을 참조하세요.

- AWS SDK for .NET (버전 2, 더 이상 사용되지 않음) 개발자 안내서
- 에 대해 더 이상 사용되지 않는 API 참조 AWS SDK for .NET

SDK 메이저 버전에 대한 유지 관리 및 지원

SDK 메이저 버전 및 기본 종속성의 유지 관리 및 지원에 대한 자세한 내용은 AWS SDK 및 도구 참조 안내서에서 다음 내용을 참조하세요.

- AWS SDKs 및 도구 유지 관리 정책
- AWS SDKs 및 도구 버전 지원 매트릭스

일반 사용 사례

는 다음을 포함한 몇 가지 매력적인 사용 사례를 실현하는 AWS SDK for .NET 데 도움이 됩니다.

- AWS Identity and Access Management (IAM)으로 사용자와 역할을 관리합니다.
- Amazon Simple Storage Service(S3)에 액세스하여 버킷을 생성하고 객체를 저장할 수 있습니다.
- 주제에 대한 Amazon Simple Notification Service(SNS) HTTP 구독을 관리합니다.
- S3 전송 유틸리티를 사용하여 Xamarin 애플리케이션에서 Amazon S3로 파일을 전송합니다.
- Amazon Simple Queue Service(Amazon SQS)를 사용하여 시스템의 구성 요소 간에 메시지 및 워크플로를 처리합니다.
- SQL 문을 Amazon S3 Select로 전송하여 효율적인 Amazon S3 전송을 수행합니다.
- Amazon EC2 인스턴스를 생성 및 시작하고 Amazon EC2 스팟 인스턴스를 구성 및 요청합니다.

이 섹션의 추가 주제

- AWS 와 관련된 도구 AWS SDK for .NET
- 에서 지원하는 플랫폼 AWS SDK for .NET
- AWS SDKs 및 도구 참조 가이드
- 수정 기록
- 의 새로운 기능 AWS SDK for .NET
- 추가 리소스

AWS 와 관련된 도구 AWS SDK for .NET

Tools for Windows PowerShell 및 Tools for PowerShell Core

AWS Tools for Windows PowerShell 및 AWS Tools for PowerShell Core 는에서 노출되는 기능을 기반으로 구축된 PowerShell 모듈입니다 AWS SDK for .NET. AWS PowerShell 도구를 사용하면 PowerShell 프롬프트에서 AWS 리소스에 대한 작업을 스크립팅할 수 있습니다. SDK의 서비스 클라이언트 및 메서드를 사용하여 cmdlet을 구현했더라도, cmdlet은 파라미터 지정 및 결과 처리에 대한 자연스러운 PowerShell 환경을 제공합니다.

시작하려면 AWS Tools for Windows PowerShell 섹션을 참조하세요.

VS 코드용 도구 키트

AWS Toolkit for Visual Studio Code는 Visual Studio Code(VS Code) 편집기용 플러그인입니다. 이 도구 키트를 통해 AWS를 사용하는 애플리케이션을 보다 쉽게 개발, 디버그 및 배포할 수 있습니다.

이 도구 키트를 사용하면 다음과 같은 작업을 수행할 수 있습니다.

- AWS Lambda 함수가 포함된 서버리스 애플리케이션을 생성한 다음 애플리케이션을 AWS CloudFormation 스택에 배포합니다.
- Amazon EventBridge 스키마 작업.
- Amazon ECS 작업 정의 파일로 작업할 때 IntelliSense를 사용합니다.
- AWS Cloud Development Kit (AWS CDK) 애플리케이션을 시각화합니다.

Toolkit for Visual Studio

AWS Toolkit for Visual Studio 는 Amazon Web Services를 사용하는 .NET 애플리케이션을 더 쉽게 개발, 디버깅 및 배포할 수 있는 Visual Studio IDE용 플러그인입니다. Toolkit for Visual Studio는 Lambda와 같은 서비스용 Visual Studio 템플릿과 웹 애플리케이션 및 서버리스 애플리케이션용 배포 마법사를 제공합니다. 사용자는 Visual Studio 내에서 Amazon EC2 인스턴스 관리, Amazon DynamoDB 테이블 사용, Amazon Simple Notification Service(SNS) 대기열로 메시지 게시 등에 AWS 탐색기를 이용할 수 있습니다.

시작하려면 AWS Toolkit for Visual Studio설정을 참조하세요.

Azure DevOps용 도구 키트

는 Azure DevOps 및 Azure DevOps Server의 빌드 및 릴리스 파이프라인이 AWS 서비스와 함께 작동하도록 하는 작업을 AWS Toolkit for Microsoft Azure DevOps 추가합니다. Amazon S3, AWS Elastic Beanstalk, AWS CodeDeploy, Lambda, AWS CloudFormation, Amazon Simple Queue Service(Amazon SQS) 및 Amazon SNS로 작업할 수 있습니다. Windows PowerShell 모듈 및 AWS Command Line Interface ()를 사용하여 명령을 실행할 수도 있습니다AWS CLI.

를 시작하려면 AWS Toolkit for Microsoft Azure DevOps 사용 설명서를 AWS Toolkit for Azure DevOps참조하세요.

에서 지원하는 플랫폼 AWS SDK for .NET

는 개발자가 서로 다른 플랫폼을 대상으로 할 수 있는 고유한 어셈블리 그룹을 AWS SDK for .NET 제공합니다. 그러나 모든 SDK 기능이 이 플랫폼들 각각에서 동일한 것은 아닙니다. 이 주제에서는 각 플랫폼에 대한 지원의 차이점을 설명합니다.

.NET Core

는 .NET Core(.NET Core 3.1, .NET 5, .NET 6 등)용으로 작성된 애플리케이션을 AWS SDK for .NET 지원합니다. AWS 서비스 클라이언트는 .NET 코어의 비동기식 호출 패턴만 지원합니다. 이것은 .NET Core 환경에서 비동기식 호출만 지원하는 Amazon S3 `TransferUtility`와 같은 서비스 클라이언트 상단에 구축되는 다수의 상위 수준 추상화에 영향을 미칩니다.

.NET Standard 2.0

의 비프레임워크 변형은 .NET Standard 2.0을 AWS SDK for .NET 준수합니다. AWS SDK for .NET 는.NET Standard에 따라 작성된 애플리케이션에 대한 비동기 메서드만 제공합니다.

.NET Framework 4.5

이 버전의 AWS SDK for .NET 는 .NET Framework 4.7.2에 대해 컴파일되고 .NET 4.0 런타임에서 실행됩니다. AWS 서비스 클라이언트는 동기 및 비동기 호출 패턴을 지원하고 C# 5.0에 도입된 비동기 및 대기 키워드를 사용합니다.

.NET Framework 3.5

의 버전 4 AWS SDK for .NET 는 .NET Framework 3.5를 지원하지 않습니다.

휴대용 클래스 라이브러리 및 Xamarin

에는 휴대용 클래스 라이브러리 구현 AWS SDK for .NET 도 포함되어 있습니다. Portable Class Library 구현은 Universal Windows Platform(UWP)과 iOS 및 Android 기반 Xamarin과 같은 다수의 플랫폼을 대상으로 삼을 수 있습니다. 자세한 내용은 Mobile SDK for .NET and Xamarin을 참조하세요. AWS 서비스 클라이언트는 비동기식 호출 패턴만 지원합니다.

Unity 지원

Unity 지원에 대한 자세한 내용은 Unity 지원에 대한 특별 고려 사항 단원을 참조하세요.

AWS SDKs 및 도구 참조 가이드

AWS SDKs 및 도구 참조 가이드에는 많은 AWS SDKs 및 도구 키트와에 관련되고 중요한 정보가 포함되어 있습니다 AWS CLI. 다음은 참조에 포함된 정보의 몇 가지 예제입니다.

- 공유 AWSconfig 및 `credentials` 파일과 해당 위치에 대한 정보.
- AWS 계정, 사용자 및 역할 설정
- 구성 및 인증 설정 참조
- AWS 공통 런타임(CRT) 라이브러리
- AWS SDKs 및 도구 유지 관리 정책
- AWS SDKs 및 도구 버전 지원 매트릭스

수정 기록

다양한 릴리스에서 변경된 내용을 찾으려면 다음을 참조하세요.

- SDK 변경 로그
- 의 새로운 기능 AWS SDK for .NET
- 문서 기록

의 새로운 기능 AWS SDK for .NET

와 관련된 새 개발에 대한 자세한 내용은 https://aws.amazon.com/sdk-for-net/ 제품 페이지와 SDK 변경 로그를 AWS SDK for .NET참조하세요.

다음은 AWS SDK for .NET의 새로운 기능입니다.

2025년 10월 17일: AWS 배포 도구 버전 2.0

.NET CLI용 AWS 배포 도구 버전 2.0이 릴리스되었습니다. 배포 도구에 대한 자세한 내용은 섹션을 참조하세요에 애플리케이션 배포 AWS. 배포 도구 버전 2.0에 대한 자세한 내용은 블로그 게시물 .NET용 배포 도구의 AWS 새로운 기능을 참조하세요.

2025년 8월 21일:의 버전 3에 대한 end-of-support 예정 AWS SDK for .NET

의 버전 3(V3)에 대한 end-of-support AWS SDK for .NET 가 발표되었습니다. V3 애플리케이션을 마이그레이션하고 중단을 방지하려면 마이그레이션 가이드를 참조하세요. 자세한 내용은 블로그 게시물 Announcing the end-of-support for the AWS SDK for .NET v3를 참조하세요.

2025년 7월 2일: 분산 캐시 공급자의 정식 출시

Amazon DynamoDB용 AWS .NET 분산 캐시 공급자를 정식 버전으로 사용할 수 있습니다. 이 공급자는 ASP.NET Core IDistributedCache 인터페이스를 구현하므로 코드 변경을 최소화하면서 DynamoDB의 완전 관리형 인프라와 내구성 있는 인프라를 캐싱 계층에 통합할 수 있습니다. 분산 캐시 공급자는 NuGet의 AWS.AspNetCore.DistributedCacheProvider 패키지를 통해 사용할 수 있습니다. 자세한 내용은 블로그 게시물 AWS .NET Distributed Cache Provider for Amazon DynamoDB now Generally Available를 참조하세요.

> **Note**
>
> 분산 캐시 공급자의 릴리스 버전은의 V4에서만 지원됩니다 AWS SDK for .NET. 분산 캐시 공급자의 미리 보기 버전을 시도한 경우 애플리케이션의 패키지 종속성을 업데이트해야 할 수 있습니다.

2025년 5월 5일: .NET용 AWS 메시지 처리 프레임워크의 정식 출시

AWS .NET용 메시지 처리 프레임워크를 정식 버전으로 사용할 수 있습니다. 프레임워크는 Amazon Simple Queue Service(SQS), Amazon Simple Notification Service(SNS) 및 Amazon EventBridge와 같은 AWS 서비스를 사용하는 .NET 메시지 처리 애플리케이션의 개발을 간소화하는 AWS네이티브 프레임워크입니다.

2025년 4월 28일:의 버전 4 AWS SDK for .NET

버전 4 AWS SDK for .NET 를 정식 버전으로 사용할 수 있습니다. 애플리케이션을 V4로 마이그레이션하는 방법에 대한 자세한 내용은 섹션을 참조하세요버전 4로 마이그레이션. 또한 일반 가용성을 발표하는 AWS SDK for .NET V4.0의 일반 가용성 블로그 게시물을 참조하세요.

2025년 2월 15일: .NET Aspire와 통합

내부 개발 루프를 개선하기 위해 .NET Aspire와의 통합이 릴리스되었습니다. 자세한 내용은 에서 .NET Aspire AWS 와 통합 AWS SDK for .NET 단원을 참조하세요.

2025년 2월 10일: 관찰성을 위한 GA 릴리스

관찰성은 시스템의 현재 상태를 내보내는 데이터에서 추론할 수 있는 범위입니다. 원격 측정 공급자의 구현을 AWS SDK for .NET포함하여에 관찰성이 추가되었습니다. 자세한 내용은 관찰성이 가이드의 및 블로그 게시물 Announcing the general availability of AWS .NET OpenTelemetry library를 참조하세요.

2025년 1월 15일: 무결성 보호를 위한 새로운 기본 동작

버전 3.7.412.0부터 SDK는 업로드 AWS SDK for .NET를 위한 CRC32 체크섬을 자동으로 계산하여 기본 무결성 보호를 제공합니다. 자세한 내용은의 GitHub 공지를 참조하세요https://github.com/aws/aws-sdk-net/issues/3610. 또한 SDK는 외부에서 설정할 수 있는 데이터 무결성 보호에 대한 전역 설정을 제공하며, SDK 및 도구 참조 안내서의 데이터 무결성 보호에서 확인할 수 있습니다. AWS SDKs

2024년 11월 15일: 버전 4용 미리 보기 4 릴리스 블로그 게시물

AWS SDK for .NET 버전 4의 미리 보기 4는 2024년 11월 15일에 릴리스되었습니다. 이 미리 보기에 대한 자세한 내용은 블로그 게시물 Preview 4 of AWS SDK for .NET V4를 참조하세요.

2024년 8월 16일: 버전 4의 미리 보기 1 릴리스 블로그 게시물

AWS SDK for .NET 버전 4는 2024년 8월 16일에 첫 번째 미리 보기로 릴리스되었습니다. 이 미리 보기에 대한 자세한 내용은 블로그 게시물 Preview 1 of AWS SDK for .NET V4를 참조하세요.

추가 리소스

지원되는 서비스

는 대부분의 AWS 인프라 제품을 AWS SDK for .NET 지원하며 더 많은 서비스가 자주 추가됩니다. SDK에서 지원하는 AWS 서비스 목록은 SDK README 파일을 참조하세요.

의 홈 페이지 AWS SDK for .NET

에 대한 자세한 내용은 https://aws.amazon.com/sdk-for-net/ SDK 홈 페이지를 AWS SDK for .NET참조하세요.

SDK 참조 설명서

SDK 참조 설명서는 검색 기능을 제공하여 SDK와 함께 포함된 모든 코드를 검색할 수 있습니다. 완결된 설명과 사용 예제를 제공합니다. 자세한 내용은 AWS SDK for .NET API 참조를 참조하세요.

AWS re:Post(이전 AWS 포럼)

AWS re:Post, 특히 주제에서 AWS SDK for .NET 질문하거나 피드백을 제공합니다 AWS. 각 설명서 페이지 하단에 관련 re:Post 주제로 연결되는 AWS re:Post 사용해 보기 링크가 있습니다. AWS 엔지니어는 주제를 모니터링하고 질문, 피드백 및 문제에 응답합니다.

re:Post에 로그인한 경우 주제를 팔로우할 수도 있습니다. 의 주제를 따르려면 모든 주제 페이지로 AWS SDK for .NET이동하여 ".NET on AWS"을 찾은 다음 팔로우 버튼을 선택합니다.

툴킷

- AWS Toolkit for Visual Studio: Microsoft Visual Studio IDE를 사용하는 경우 AWS Toolkit for Visual Studio 사용 설명서를 확인해야 합니다.
- AWS Toolkit for Visual Studio Code: Microsoft Visual Studio IDE를 사용하는 경우 AWS Toolkit for Visual Studio Code 사용 설명서를 확인해야 합니다.

유용한 라이브러리, 확장 및 도구

.NET 애플리케이션 및 AWS의 서비스를 구축하는 데 유용한 라이브러리, 도구 및 리소스를 위한 링크는 GitHub 웹 사이트의 aws/dotnet 및 aws/aws-sdk-net 리포지토리를 방문하십시오.

다음은 몇 가지 예시입니다.

- AWS Systems Manager용 .NET 구성 확장
- AWS 확장 .NET Core 설정
- AWS .NET 로깅
- Amazon Cognito Authentication 확장 라이브러리
- AWS X-Ray SDK for .NET

기타 리소스

유용할 수 있는 기타 리소스는 다음과 같습니다.

- 개발자 net
- AWS 클라우드의 .NET 개발 환경 - 빠른 시작 참조 배포
- Hello, Cloud! 블로그
- AWS 백서: 에서 .NET 애플리케이션 개발 및 배포 AWS
- AWS Microservice Extractor for .NET
- Porting Assistant for .NET
- AWS SDKs 및 도구 참조 가이드

시작하기 AWS SDK for .NET

를 설치, 설정 및 사용하는 방법을 알아봅니다 AWS SDK for .NET. 도구 체인을 설치하고 애플리케이션이 AWS 서비스에 액세스하는 데 필요한 여러 가지 필수 항목을 설정해야 합니다. 다음이 포함됩니다.

- 적절한 사용자 계정 또는 역할
- 해당 사용자 계정 또는 해당 역할을 수임하기 위한 인증 정보

프로젝트 및 기타 중요한 부분을 구성하는 방법에 대한 자세한 내용은 섹션을 참조하세요구성 AWS SDK for .NET.

주제

- 용 도구 체인 설치 및 구성 AWS SDK for .NET
- 를 AWS SDK for .NET 사용하여 인증 AWS
- 를 사용하여 간단한 애플리케이션 생성 AWS SDK for .NET

용 도구 체인 설치 및 구성 AWS SDK for .NET

를 사용하려면 특정 개발 도구가 설치되어 있어야 AWS SDK for .NET합니다.

교차 플랫폼 개발

Windows, Linux 또는 macOS에서 교차 플랫폼 .NET 개발에 대한 필수 항목은 다음과 같습니다.

- Microsoft .NET Core SDK, 버전 2.1, 3.1 이상. 여기에는 .NET 명령줄 인터페이스(CLI)(**`dotnet`**) 및 .NET Core 런타임이 포함되어 있습니다.

- 운영 체제 및 요구 사항에 적합한 코드 편집기 또는 통합 개발 환경(IDE)입니다. 이는 일반적으로 .NET Core에 대한 지원을 제공합니다.

  예를 들면 Microsoft Visual Studio Code(VS Code), JetBrains Rider, Microsoft Visual Studio 등이 있습니다.

- (선택 사항) 선택한 편집기와 운영 체제에 사용 가능한 AWS 도구 키트입니다.

예를 들면 AWS Toolkit for Visual Studio Code, AWS Toolkit for JetBrains 및 AWS Toolkit for Visual Studio입니다.

Visual Studio 및 .NET Core를 통한 Windows

Visual Studio 및 .NET Core를 통한 Windows 개발에 필요한 항목은 다음과 같습니다.

- Microsoft Visual Studio
- Microsoft .NET Core 2.1, 3.1 이상

  일반적으로 Visual Studio의 최신 버전을 설치할 때 기본적으로 포함됩니다.

- (선택 사항) Visual Studio에서 AWS 리소스 및 로컬 프로파일을 관리하기 위한 사용자 인터페이스를 제공하는 플러그인 AWS Toolkit for Visual Studio인 입니다. 툴킷을 설치하려면 AWS Toolkit for Visual Studio설정을 참조하세요.

  자세한 내용은 AWS Toolkit for Visual Studio 사용 설명서를 참조하십시오.

다음 단계

를 AWS SDK for .NET 사용하여 인증 AWS

를 AWS SDK for .NET 사용하여 인증 AWS

로 개발할 AWS 때 로 코드를 인증하는 방법을 설정해야 합니다 AWS 서비스. 환경 및 사용 가능한 액세스에 따라 AWS 리소스에 대한 프로그래밍 방식 AWS 액세스를 구성할 수 있는 다양한 방법이 있습니다.

SDK에 대한 다양한 인증 방법을 확인하려면 AWS SDK 및 도구 참조 가이드에서 인증 및 액세스를 참조하세요.

이 주제에서는 새 사용자가 로컬에서 개발 중이고, 고용주로부터 인증 방법을 받지 않았으며, AWS IAM Identity Center 를 사용하여 임시 자격 증명을 얻을 것이라고 가정합니다. 사용자 환경이 이러한 가정에 해당하지 않는 경우 이 주제의 일부 정보가 사용자에게 적용되지 않거나 일부 정보가 이미 제공되었을 수 있습니다.

이 환경을 구성하려면 몇 가지 단계를 수행해야 하며, 요약하면 다음과 같습니다.

1. 콘솔 자격 증명을 사용하여 로그인
2. IAM Identity Center 사용 및 구성
3. IAM Identity Center를 사용하도록 SDK를 구성합니다.
4. AWS 액세스 포털 세션 시작

콘솔 자격 증명을 사용하여 로그인

기존 AWS Management Console 로그인 자격 증명을 사용하여 AWS 서비스에 프로그래밍 방식으로 액세스할 수 있습니다. 브라우저 기반 인증 흐름 후는 AWS CLI AWS Tools for PowerShell 및 .NET v4용 AWS SDK와 같은 로컬 개발 도구에서 작동하는 임시 자격 증명을 AWS 생성합니다.

이 프로세스를 사용하면 초기 계정 설정 중에 생성된 루트 자격 증명, IAM 사용자 또는 자격 증명 공급자의 페더레이션 자격 증명을 사용하여 인증할 수 있으며 AWS CLI 또는 AWS Tools for PowerShell 가 자동으로 임시 자격 증명을 관리합니다. 이 접근 방식은 장기 자격 증명을 로컬에 저장할 필요가 없으므로 보안이 강화됩니다. 에서 `Invoke-AWSLogin` AWS Tools for PowerShell cmdlet을 실행하거나 AWS CLI에서 `aws login` 명령을 실행하면 활성 콘솔 세션에서 선택하거나 브라우저 기반 인증 흐름을 통해 로그인할 수 있으며, 그러면 임시 자격 증명이 자동으로 생성됩니다. 임시 자격 증명 토큰은 15분 후에 만료되지만 .NET v4용 AWS SDK는 요청 중에 필요한 경우 토큰을 자동으로 새로 고칩니다.

> **Important**
>
> 인증에 콘솔 자격 증명을 사용하는 경우 애플리케이션이 다음 NuGet 패키지를 참조해야 AWS 로그인 확인이 작동합니다.
>
> - AWSSDK.Signin
>
> 이러한 패키지를 참조하지 않으면 런타임 예외가 발생합니다.

- AWS Tools for PowerShell 로그인 ID 개발자 안내서.
- AWS CLI를 사용하여 콘솔 자격 증명을 사용하여 AWS 로컬 개발을 위해 로그인합니다.

AWS Tools for PowerShell 또는 AWS CLI에 대해 위에 제공된 단계를 따라 로그인하면 `config` 파일은 다음과 같습니다.

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

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

IAM Identity Center 사용 및 구성

IAM Identity Center를 사용하려면 먼저 사용하도록 설정하고 구성해야 합니다. SDK에서 이 작업을 수행하는 방법에 대한 자세한 내용은 AWS SDK 및 도구 참조 가이드의 IAM Identity Center 인증 항목에 있는 1단계를 참조하세요. 특히 IAM Identity Center를 통한 액세스가 설정되어 있지 않습니다 아래에 있는 필요한 지침을 따르세요.

IAM Identity Center를 사용하도록 SDK를 구성합니다.

IAM Identity Center를 사용하도록 SDK를 구성하는 방법에 대한 정보는 AWS SDK 및 도구 참조 가이드의 IAM Identity Center 인증에 대한 항목의 2단계에 나와 있습니다. 이 구성을 완료하면 시스템에 다음 요소가 포함되어야 합니다.

- 애플리케이션을 실행하기 전에 AWS 액세스 포털 세션을 시작하는 데 AWS CLI사용하는 입니다.
- SDK에서 참조할 수 있는 구성 값 세트가 있는 `[default]` 프로파일이 포함된 공유 AWS `config` 파일입니다. 이 파일의 위치를 찾으려면 AWS SDK 및 도구 참조 가이드에서 공유 파일의 위치를 참조하세요. 는 요청을 보내기 전에 프로필의 SSO 토큰 공급자를 AWS SDK for .NET 사용하여 자격 증명을 획득합니다 AWS. `sso_role_name` 값은 IAM 신원 센터 권한 집합에 연결된 IAM 역할로, 애플리케이션에서 사용되는 AWS 서비스 서비스에 대한 액세스를 허용해야 합니다.

  다음 샘플 `config` 파일은 SSO 토큰 공급자로 설정된 기본 프로필을 보여줍니다. 프로필의 `sso_session` 설정은 이름이 지정된 `sso-session` 섹션을 참조합니다. `sso-session` 섹션에는 AWS 액세스 포털 세션을 시작하는 설정이 포함되어 있습니다.

  ```
  [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

인증 AWS IAM Identity Center 에를 사용하는 경우 SSO 확인이 작동하려면 애플리케이션이 다음 NuGet 패키지를 참조해야 합니다.

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

이러한 패키지를 참조하지 않으면 런타임 예외가 발생합니다.

AWS 액세스 포털 세션 시작

에 액세스하는 애플리케이션을 실행하기 전에 SDK가 IAM Identity Center 인증을 사용하여 자격 증명을 확인하려면 활성 AWS 액세스 포털 세션이 AWS 서비스필요합니다. 구성된 세션 길이에 따라 결국 액세스가 만료되고 SDK에 인증 오류가 발생합니다. AWS 액세스 포털에 로그인하려면에서 다음 명령을 실행합니다 AWS CLI.

```
aws sso login
```

기본 프로필이 설정되어 있으므로 `--profile` 옵션으로 명령을 호출할 필요가 없습니다. SSO 토큰 공급자 구성에서 명명된 프로필을 사용하는 경우 `aws sso login --profile named-profile` 명령을 사용합니다.

이미 활성 세션이 있는지 테스트하려면 다음 AWS CLI 명령을 실행합니다.

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

이 명령에 대한 응답은 공유 `config` 파일에 구성된 IAM Identity Center 계정 및 권한 집합을 보고해야 합니다.

> **Note**
>
> 이미 활성 AWS 액세스 포털 세션이 있고를 실행하는 경우 `aws sso login`자격 증명을 제공할 필요가 없습니다.
> 로그인 프로세스에서 데이터에 대한 AWS CLI 액세스를 허용하라는 메시지가 표시될 수 있습니다. AWS CLI 는 SDK for Python을 기반으로 구축되므로 권한 메시지에 `botocore` 이름의 변형이 포함될 수 있습니다.

추가 정보

- 개발 환경에서 IAM Identity Center 및 SSO를 사용하는 방법에 대한 추가 정보는 고급 인증섹션의 Single Sign-On를 참조하세요. 이 정보에는 대체 방법 및 고급 방법과 이러한 방법을 사용하는 방법을 보여주는 자습서가 포함되어 있습니다.
- 프로필 및 환경 변수 사용과 같은 SDK 인증에 대한 자세한 옵션은 AWS SDK 및 도구 참조 가이드의 구성 장을 참조하세요.
- 모범 사례에 대해 자세히 알아보려면 IAM 사용 설명서에서 IAM의 보안 모범 사례를 참조하세요.
- 단기 AWS 자격 증명을 생성하려면 IAM 사용 설명서의 임시 보안 자격 증명을 참조하세요.
- 다른 보안 인증 공급자에 대해 알아보려면 AWS SDK 및 도구 참조 가이드에서 표준화된 보안 인증 공급자를 참조하세요.

를 사용하여 간단한 애플리케이션 생성 AWS SDK for .NET

이 섹션은 AWS SDK for .NET에 익숙하지 않은 개발자를 위한 기본 자습서를 제공합니다.

> **Note**
>
> 이 자습서를 사용하기 전에 먼저 도구 체인을 설치하고 SDK 인증을 구성해야 합니다.

코드 예제와 함께 특정 AWS 서비스를 위한 소프트웨어 개발에 대한 자세한 내용은 섹션을 참조하세요 AWS 서비스 호출. 추가 코드 예제는 SDK for .NET (v4) 코드 예제 단원을 참조하세요.

주제

- 를 사용하는 간단한 교차 플랫폼 애플리케이션 AWS SDK for .NET

- 를 사용하는 간단한 Windows 기반 애플리케이션 AWS SDK for .NET
- 다음 단계

를 사용하는 간단한 교차 플랫폼 애플리케이션 AWS SDK for .NET

이 자습서에서는 교차 플랫폼 개발을 위해 AWS SDK for .NET 및 .NET Core를 사용합니다. 이 자습서는 SDK를 사용하여 소유한 Amazon S3 버킷을 나열하고 선택적으로 버킷을 생성하는 방법을 보여줍니다.

.NET 명령줄 인터페이스(CLI)와 같은 교차 플랫폼 도구를 사용하여 이 자습서를 수행합니다. 개발 환경을 구성하는 다른 방법은 용 도구 체인 설치 및 구성 AWS SDK for .NET을 참조하십시오.

Windows, Linux 또는 macOS에서 교차 플랫폼 .NET 개발에 대한 필수 항목:

- Microsoft .NET Core SDK, 버전 2.1, 3.1 이상. 여기에는 .NET 명령줄 인터페이스(CLI)(**dotnet**) 및 .NET Core 런타임이 포함되어 있습니다.

- 운영 체제 및 요구 사항에 적합한 코드 편집기 또는 통합 개발 환경(IDE)입니다. 이는 일반적으로 .NET Core에 대한 지원을 제공합니다.

  예를 들면 Microsoft Visual Studio Code(VS Code), JetBrains Rider, Microsoft Visual Studio 등이 있습니다.

> **Note**
>
> 이 자습서를 사용하기 전에 먼저 도구 체인을 설치하고 SDK 인증을 구성해야 합니다.

단계

- 프로젝트 생성
- 코드 생성
- 애플리케이션을 실행합니다
- 정리

프로젝트 생성

1. 명령 프롬프트 또는 터미널을 엽니다. .NET 프로젝트를 만들 수 있는 운영 체제 폴더를 찾거나 생성합니다.
2. 해당 폴더에서 다음 명령을 실행하여 .NET 프로젝트를 만듭니다.

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

3. 새로 만든 `S3CreateAndList` 폴더로 이동하여 다음 명령을 실행합니다.

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

위의 명령은 NuGet 패키지 관리자에서 NuGet 패키지를 설치합니다. 이 자습서에 필요한 NuGet 패키지를 정확하게 알고 있으므로 이제 이 단계를 수행할 수 있습니다. 또한 필요한 패키지가 개발 중에 알려지게 되는 것이 일반적입니다. 이 경우 비슷한 명령을 실행할 수 있습니다.

코드 생성

1. `S3CreateAndList` 폴더에서 `Program.cs`를 찾아 코드 편집기에서 엽니다.
2. 내용을 다음 코드로 바꾸고 파일을 저장합니다.

```
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;
        }
    }
}
```

애플리케이션을 실행합니다

1. 다음 명령을 실행합니다.

```
dotnet run
```

2. 출력을 검사하여 소유한 Amazon S3 버킷 수(있는 경우)와 해당 이름을 확인합니다.
3. 새 Amazon S3 버킷의 이름을 선택합니다. "dotnet-quicktour-s3-1-cross-"를 기본으로 사용하고 GUID 또는 사용자 이름과 같은 고유한 항목을 추가합니다. Amazon S3 사용 설명서의 버킷 이름 지정 규칙에 설명된 대로 버킷 이름 규칙을 준수해야 합니다.
4. 다음 명령을 실행하여 *amzn-s3-demo-bucket*을 선택한 버킷의 이름으로 바꿉니다.

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

5. 출력을 검사하여 생성된 새 버킷을 확인합니다.

정리

이 자습서를 수행하는 동안 현재 정리하도록 선택할 수 있는 몇 가지 리소스를 만들었습니다.

- 이전 단계에서 애플리케이션이 생성한 버킷을 유지하지 않으려면 https://console.aws.amazon.com/s3/에서 Amazon S3 콘솔을 사용하여 삭제합니다.
- .NET 프로젝트를 유지하지 않으려면 개발 환경에서 `S3CreateAndList` 폴더를 제거하십시오.

다음으로 진행할 단계

간략한 설명 메뉴로 이동하거나 이 간략한 설명의 끝으로 바로 이동합니다.

를 사용하는 간단한 Windows 기반 애플리케이션 AWS SDK for .NET

이 자습서에서는 Windows AWS SDK for .NET 에서 Visual Studio 및 .NET Core와 함께를 사용합니다. 이 자습서는 SDK를 사용하여 소유한 Amazon S3 버킷을 나열하고 선택적으로 버킷을 생성하는 방법을 보여줍니다.

Visual Studio 및 .NET Core를 사용하여 Windows에서 이 자습서를 수행하게 됩니다. 개발 환경을 구성하는 다른 방법은 용 도구 체인 설치 및 구성 AWS SDK for .NET을 참조하십시오.

Visual Studio 및 .NET Core를 통한 Windows 개발에 필요한 항목:

- Microsoft Visual Studio

- Microsoft .NET Core 2.1, 3.1 이상

  일반적으로 Visual Studio의 최신 버전을 설치할 때 기본적으로 포함됩니다.

> **Note**
>
> 이 자습서를 사용하기 전에 먼저 도구 체인을 설치하고 SDK 인증을 구성해야 합니다.

단계

- 프로젝트 생성
- 코드 생성
- 애플리케이션을 실행합니다
- 정리

프로젝트 생성

1. Visual Studio를 열고 콘솔 앱 템플릿의 C# 버전을 사용하는 새 프로젝트를 생성합니다. 즉, "....NET에서 실행할 수 있는 명령줄 애플리케이션 생성의 경우..."라는 설명이 있습니다. `S3CreateAndList` 프로젝트의 이름을 지정합니다.

   > **Note**
   >
   > 콘솔 앱 템플릿의 .NET Framework 버전을 선택하지 마십시오. 선택했다면 .NET Framework 4.7.2 이상을 사용해야 합니다.

2. 새로 생성된 프로젝트를 로드한 상태에서 도구, NuGet Package Manager(NuGet 패키지 관리자), Manage NuGet Packages for Solution(솔루션을 위한 NuGet 패키지 관리)을 선택합니다.
3. `AWSSDK.S3`, `AWSSDK.SecurityToken`, `AWSSDK.SSO` 및 `AWSSDK.SSOOIDC`의 NuGet 패키지를 찾아 프로젝트에 설치합니다.

   이 프로세스는 NuGet 패키지 관리자에서 NuGet 패키지를 설치합니다. 이 자습서에 필요한 NuGet 패키지를 정확하게 알고 있으므로 이제 이 단계를 수행할 수 있습니다. 또한 필요한 패키지가 개발 중에 알려지게 되는 것이 일반적입니다. 이 경우 유사한 프로세스를 따라 설치하십시오.
4. 명령 프롬프트에서 애플리케이션을 실행하려는 경우 이제 명령 프롬프트를 열고 빌드 출력이 포함될 폴더로 이동합니다. 일반적으로 `S3CreateAndList\S3CreateAndList\bin\Debug\net6.0`과 비슷하지만, 환경에 따라 달라집니다.

코드 생성

1. `S3CreateAndList` 프로젝트에서 `Program.cs`를 찾아 IDE에서 엽니다.
2. 내용을 다음 코드로 바꾸고 파일을 저장합니다.

```
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. 애플리케이션을 빌드합니다.

> **Note**
>
> 이전 버전의 Visual Studio를 사용하는 경우 다음과 유사한 빌드 오류가 발생할 수 있습니다.
> “C# 7.0에서는 '비동기 메인' 기능을 사용할 수 없습니다. 언어 버전 7.1 이상을 사용하십시오.”
> 이 오류가 발생하면 해당 언어의 최신 버전을 사용하도록 프로젝트를 설정하세요. 이 작업은 일반적으로 프로젝트 속성, 빌드, 고급에서 수행됩니다.

애플리케이션을 실행합니다

1. 명령줄 인수 없이 애플리케이션을 실행합니다. 명령 프롬프트(이전에 연 경우) 또는 IDE에서 이 작업을 수행합니다.
2. 출력을 검사하여 소유한 Amazon S3 버킷 수(있는 경우)와 해당 이름을 확인합니다.
3. 새 Amazon S3 버킷의 이름을 선택합니다. "dotnet-quicktour-s3-1-winvs-"를 기본으로 사용하고 GUID 또는 사용자 이름과 같은 고유한 항목을 추가합니다. Amazon S3 사용 설명서의 버킷 이름 지정 규칙에 설명된 대로 버킷 이름 규칙을 준수해야 합니다.
4. 애플리케이션을 다시 실행합니다. 이번에는 버킷 이름을 제공합니다.

   명령줄에서 다음 명령의 *amzn-s3-demo-bucket*을 선택한 버킷의 이름으로 바꿉니다.

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

   또는 IDE에서 애플리케이션을 실행하는 경우 프로젝트, S3CreateAndList Properties(S3CreateAndList 속성), 디버깅으로 선택하고 버킷 이름을 입력합니다.
5. 출력을 검사하여 생성된 새 버킷을 확인합니다.

정리

이 자습서를 수행하는 동안 현재 정리하도록 선택할 수 있는 몇 가지 리소스를 만들었습니다.

- 이전 단계에서 애플리케이션이 생성한 버킷을 유지하지 않으려면 https://console.aws.amazon.com/s3/에서 Amazon S3 콘솔을 사용하여 삭제합니다.
- .NET 프로젝트를 유지하지 않으려면 개발 환경에서 `S3CreateAndList` 폴더를 제거하십시오.

다음으로 진행할 단계

간략한 설명 메뉴로 이동하거나 이 간략한 설명의 끝으로 바로 이동합니다.

다음 단계

이 자습서를 수행하는 동안 생성한 리소스 중 남은 리소스를 모두 정리해야 합니다. 이러한 AWS 리소스는 파일 및 폴더와 같은 개발 환경의 리소스일 수 있습니다.

이제를 둘러보았으므로 프로젝트를 시작할 AWS SDK for .NET수 있습니다. ???

구성 AWS SDK for .NET

를 사용하려면 환경을 AWS SDK for .NET설정하고 애플리케이션이 AWS 서비스에 액세스하는 데 필요한 여러 필수 항목을 구성해야 합니다. ??? 다음이 포함됩니다.

- AWS 리전의 사양
- AWSSDK 패키지 또는 어셈블리

이 섹션의 일부 주제에서는 이러한 필수 사항을 구성하는 방법에 대한 정보를 제공합니다.

이 섹션 및 다른 섹션의 다른 주제에서는 프로젝트를 구성할 수 있는 고급 방법에 대한 정보를 제공합니다.

주제

- 새 프로젝트 시작
- NuGet을 사용하여 AWSSDK 패키지를 설치
- NuGet 없이 AWSSDK 어셈블리 설치
- 에 대한 AWS 리전 설정 AWS SDK for .NET
- 보안 인증 정보 및 프로파일 확인
- 재시도 및 제한 시간
- 관찰성
- 사용자 및 역할에 대한 추가 정보
- AWS SDK for .NET 프로젝트의 고급 구성
- 레거시 보안 인증 사용

새 프로젝트 시작

AWS 서비스에 액세스하기 위해 새 프로젝트를 시작하는 데 사용할 수 있는 몇 가지 기법이 있습니다. 이러한 일부 기술은 다음과 같습니다.

- 에서 .NET 개발을 처음 AWS 사용하거나 최소한를 처음 AWS SDK for .NET사용하는 경우에서 전체 예제를 볼 수 있습니다간단한 애플리케이션 생성. SDK에 대한 소개를 제공합니다.

- .NET CLI를 사용하여 기본 프로젝트를 시작할 수 있습니다. 예제를 보려면 명령 프롬프트나 터미널을 열고 폴더나 디렉터리를 만들어 탐색한 후 다음을 입력하십시오.

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

  빈 프로젝트가 생성되며, 이 프로젝트에 코드 및 NuGet 패키지를 추가할 수 있습니다. 자세한 내용은 .NET Core 안내서를 참조하십시오.

  프로젝트 템플릿 목록을 보려면 `dotnet new --list`를 사용하세요.

- 에는 다양한 AWS 서비스에 대한 C# 프로젝트 템플릿이 AWS Toolkit for Visual Studio 포함되어 있습니다. Visual Studio에서 도구 키트를 설치한 후 새 프로젝트를 만드는 동안 이러한 템플릿에 액세스할 수 있습니다.

  이를 보려면 AWS Toolkit for Visual Studio 사용 설명서의 AWS 서비스 작업으로 이동합니다. 새 프로젝트를 만드는 몇 가지 예가 이 섹션에 포함되어 있습니다.

- Windows에서 Visual Studio를 사용하지만를 사용하지 않고 개발하는 경우 일반적인 기술을 AWS Toolkit for Visual Studio사용하여 새 프로젝트를 생성합니다.

  예제를 보려면 Visual Studio를 열고 파일, 신규, 프로젝트를 선택합니다. ".net core"를 검색하고 콘솔 앱(.NET Core) 또는 WPF 앱(.NET Core) 템플릿의 C# 버전을 선택합니다. 빈 프로젝트가 생성되며, 이 프로젝트에 코드 및 NuGet 패키지를 추가할 수 있습니다.

에서 AWS 서비스 작업 방법의 몇 가지 예를 찾을 수 있습니다가이드 코드 예제.

> **Important**
>
> 인증 AWS IAM Identity Center 에를 사용하는 경우 애플리케이션이 다음 NuGet 패키지를 참조해야 SSO 확인이 작동합니다.
>
> - `AWSSDK.SSO`
> - `AWSSDK.SSOOIDC`
>
> 이러한 패키지를 참조하지 않으면 런타임 예외가 발생합니다.

NuGet을 사용하여 AWSSDK 패키지를 설치

NuGet은 .NET 플랫폼용 패키지 관리 시스템입니다. NuGet을 사용하면 AWSSDK 패키지와 기타 여러 확장을 프로젝트에 설치할 수 있습니다. 자세한 내용은 GitHub 웹 사이트의 aws/dotnet 리포지토리를 참조하세요.

NuGet에는 항상 최신 버전의 AWSSDK 패키지와 이전 버전이 있습니다. NuGet은 패키지 간 종속성을 인식하며 모든 필수 패키지를 자동으로 설치합니다.

> **Warning**
>
> NuGet 패키지 목록에는 단순히 "AWSSDK"(추가된 식별자 없음)라는 이름의 패키지가 포함될 수 있습니다. 이 NuGet 패키지를 설치하지 마십시오. 이 패키지는 레거시이므로 새 프로젝트에 사용해서는 안 됩니다.

NuGet으로 설치한 패키지는 중앙 위치 대신 프로젝트와 함께 저장됩니다. 따라서 다른 애플리케이션에 대한 호환성 문제 없이 특정 애플리케이션에 고유한 어셈블리 버전을 설치할 수 있습니다. NuGet에 대한 자세한 내용은 NuGet 설명서를 참조하십시오.

> **Note**
>
> 프로젝트별로 NuGet 패키지를 다운로드 및 설치할 수 없거나 설치할 수 없는 경우 AWSSDK 어셈블리를 구하여 로컬(또는 온프레미스)에 저장할 수 있습니다.
> 해당 사항이 해당되고 AWSSDK 어셈블리를 아직 구입하지 않은 경우 AWSSDK 어셈블리 가져오기를 참조하세요. 로컬에 저장된 어셈블리를 사용하는 방법을 알아보려면 NuGet 없이 AWSSDK 어셈블리 설치을 참조하세요.

명령 프롬프트 또는 터미널에서 NuGet 사용

1. NuGet의 AWSSDK 패키지로 이동하여 프로젝트에 필요한 패키지(예: AWSSDK.S3)를 확인합니다.
2. 다음 예에 나와 있는 대로 해당 패키지의 웹 페이지에서.NET CLI 명령을 복사합니다.

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

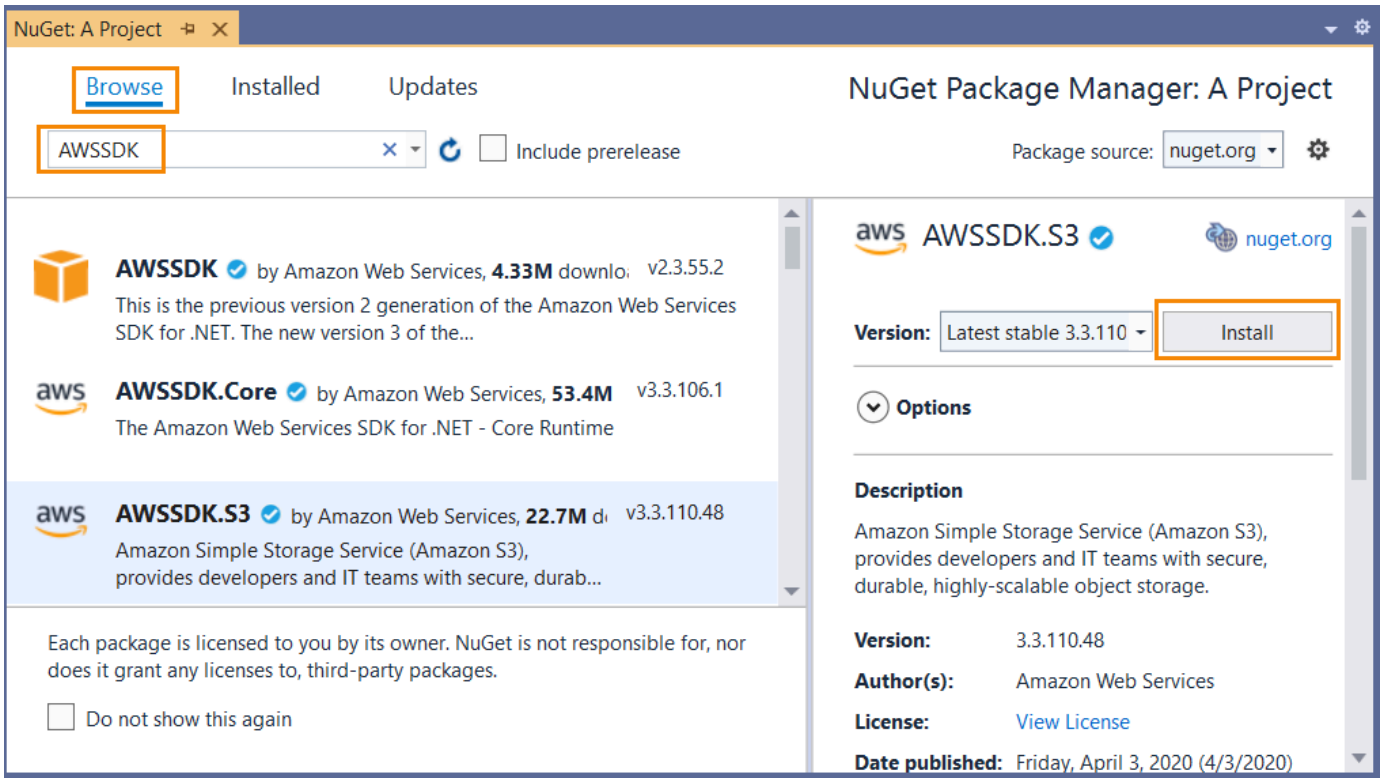
3. 프로젝트의 디렉터리에서 .NET CLI 명령을 실행합니다. NuGet은 AWSSDK.Core와 같은 종속성도 설치합니다.

> **Note**
>
> 최신 버전의 NuGet 패키지만 원하는 경우 다음 예와 같이 명령에서 버전 정보를 제외할 수 있습니다.
>
> **`dotnet add package AWSSDK.S3`**

Visual Studio 솔루션 탐색기에서 NuGet 사용

1. Solution Explorer(솔루션 탐색기)에서 프로젝트를 마우스 오른쪽 버튼으로 클릭하여 컨텍스트 메뉴를 열고 Manage NuGet Packages(NuGet 패키지 관리)를 선택합니다.
2. NuGet Package Manager(NuGet 패키지 관리자)의 왼쪽 창에서 찾아보기를 선택합니다. 그런 다음 검색 상자를 사용하여 설치할 패키지를 검색합니다. NuGet은 AWSSDK.Core와 같은 종속성도 설치합니다.

   다음 그림은 AWSSDK.S3 패키지의 설치를 보여줍니다.



Package Manager Console에서 NuGet 사용

Visual Studio에서 도구, NuGet 패키지 관리자, 패키지 관리자 콘솔을 선택합니다.

`Install-Package` 명령을 사용하여 패키지 관리자 콘솔에서 원하는 AWSSDK 패키지를 설치할 수 있습니다. 예를 들어 AWSSDK.S3을 설치하려면 다음 명령을 사용합니다.

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

NuGet은 AWSSDK.Core와 같은 종속성도 설치합니다.

이전 버전의 패키지를 설치해야 할 경우 다음 예제와 같이 `-Version` 옵션을 사용하여 원하는 패키지 버전을 지정합니다.

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

패키지 관리자 콘솔 명령에 대한 자세한 내용은 Microsoft의 NuGet 설명서에서 PowerShell 참조를 참조하세요.

NuGet 없이 AWSSDK 어셈블리 설치

이 주제에서는 AWSSDK 어셈블리 가져오기에서 설명한 대로 로컬(또는 온프레미스)에서 구하여 저장한 AWSSDK 어셈블리를 사용하는 방법을 설명합니다. 이는 SDK 참조를 처리하는 데 권장되는 방법은 아니지만 일부 환경에서는 필요합니다.

> **Note**
>
> SDK 참조를 처리하는 권장되는 방법은 각 프로젝트에 필요한 NuGet 패키지만 다운로드하여 설치하는 것입니다. 이 방법은 NuGet을 사용하여 AWSSDK 패키지를 설치에 설명되어 있습니다.

AWSSDK 어셈블리를 설치하려면

1. 필요한 AWSSDK 어셈블리를 저장할 폴더를 프로젝트 영역에 생성하세요. 예를 들어 이 `AwsAssemblies` 폴더를 호출할 수 있습니다.
2. 아직 하지 않았다면 AWSSDK 어셈블리를 확보하세요. 그러면 어셈블리가 로컬 다운로드 또는 설치 폴더에 저장됩니다. 해당 다운로드 폴더에서 필요한 어셈블리의 DLL 파일을 프로젝트(이 예제에서는 `AwsAssemblies` 폴더)로 복사합니다.

   종속성도 복사하세요. GitHub 웹 사이트에서 종속성에 대한 정보를 찾을 수 있습니다.
3. 다음과 같이 필수 어셈블리를 참조하세요.

Cross-platform development

1. 프로젝트의 `.csproj` 파일을 열고 `<ItemGroup>` 요소를 추가합니다.
2. `<ItemGroup>` 요소에 각 필수 어셈블리의 `Include` 속성이 있는 `<Reference>` 요소를 추가합니다.

   예를 들어 Amazon S3의 경우 프로젝트의 `.csproj` 파일에 다음 줄을 추가합니다.

   Linux 및 macOS:

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

   Windows의 경우:

   ```
   <ItemGroup>
     <Reference Include="AwsAssemblies\AWSSDK.Core.dll" />
     <Reference Include="AwsAssemblies\AWSSDK.S3.dll" />
   </ItemGroup>
   ```
3. 프로젝트의 `.csproj` 파일을 저장합니다.

Windows with Visual Studio and .NET Core

1. Visual Studio에서 프로젝트를 로드하고 프로젝트, 참조 추가를 엽니다.
2. 대화 상자 아래쪽에 있는 찾아보기 버튼을 선택합니다. 프로젝트 폴더 및 필수 DLL 파일을 복사한 하위 폴더(예: `AwsAssemblies`)로 이동합니다.
3. 모든 DLL 파일을 선택하고 추가를 선택한 다음 확인을 선택합니다.
4. 프로젝트를 저장합니다.

에 대한 AWS 리전 설정 AWS SDK for .NET

AWS 리전을 사용하면 특정 지리적 리전에 물리적으로 상주하는 AWS 서비스에 액세스할 수 있습니다. 이는 중복성에 유용할 뿐 아니라 데이터와 애플리케이션이 이를 사용하기 위해 고객님 및 고객님의 사용자가 접근하는 위치 가까이에서 계속해서 실행되도록 하는 데 도움이 될 수 있습니다.

각 AWS 서비스에 대해 지원되는 모든 리전 및 엔드포인트의 현재 목록을 보려면의 서비스 엔드포인트 및 할당량을 참조하세요AWS 일반 참조. 기존 리전 엔드포인트 목록을 보려면 AWS 서비스 엔드포인트를 참조하십시오. 리전에 대한 자세한 정보를 보려면 계정에서 사용할 수 있는 AWS 리전 지정을 참조하세요.

특정 리전으로 이동하는 AWS 서비스 클라이언트를 생성할 수 있습니다. 모든 AWS 서비스 클라이언트에 사용할 리전으로 애플리케이션을 구성할 수도 있습니다. 다음은 이 두 가지 사례에 대해 설명합니다.

특정 리전을 사용하여 서비스 클라이언트를 생성합니다.

애플리케이션의 모든 AWS 서비스 클라이언트에 대해 리전을 지정할 수 있습니다. 이러한 방식으로 리전을 설정하면 특별한 서비스 클라이언트에 대한 전역 설정에 우선합니다.

기존 리전

이 예제는 기존 리전의 Amazon EC2 클라이언트를 인스턴스화하는 방법을 보여줍니다. 정의된 RegionEndpoint 필드를 사용합니다.

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

RegionEndpoint 클래스를 사용하는 새 리전

이 예제에서는 RegionEndpoint.GetBySystemName을 사용하여 새 리전 엔드포인트를 구하는 방법을 보여줍니다.

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

서비스 클라이언트 구성 클래스를 사용하는 새 리전

이 예제에서는 서비스 클라이언트 구성 클래스의 ServiceURL 속성을 사용하여 리전을 지정하는 방법을 보여줍니다. 이 경우에는 AmazonEC2Config 클래스를 사용합니다.

이 기법은 리전 엔드포인트가 정규 리전 엔드포인트 패턴을 따르지 않는 경우에도 사용할 수 있습니다.

```
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
}
```

모든 서비스 클라이언트의 리전 지정

애플리케이션이 생성하는 모든 AWS 서비스 클라이언트에 대해 리전을 지정하는 방법에는 여러 가지가 있습니다. 이 지역은 특정 리전으로 생성되지 않은 서비스 클라이언트에 사용됩니다.

는 다음 순서로 리전 값을 AWS SDK for .NET 찾습니다.

프로파일

애플리케이션 또는 SDK가 로드한 프로필을 설정합니다. 자세한 내용은 보안 인증 정보 및 프로파일 확인 단원을 참조하십시오.

환경 변수

`AWS_REGION` 환경 변수를 설정합니다.

Linux 또는 macOS에서:

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

Windows의 경우:

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

> **Note**
>
> `export` 또는 `setx`를 사용하여 전체 시스템에 대해 이 환경 변수를 설정하면 AWS SDK for .NET뿐만 아니라 모든 SDK 및 툴킷에 영향을 줍니다.

AWSConfigs 클래스

AWSConfigs.AWSRegion 속성으로 설정합니다.

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

리전 해상도

위에서 설명한 방법 중 어느 것도를 지정하는 데 사용되지 않으면 AWS 리전는 AWS 서비스 클라이언트가 운영할 리전을 찾으려고 AWS SDK for .NET 시도합니다.

리전 해결 순서

1. `app.config` 및 `web.config`와 같은 애플리케이션 구성 파일.
2. 환경 변수(`AWS_REGION` 및 `AWS_DEFAULT_REGION`).
3. `AWSConfigs.AWSProfileName`의 값으로 지정된 이름을 가진 프로필.
4. `AWS_PROFILE` 환경 변수로 이름이 지정된 프로필.
5. `[default]` 프로필.
6. Amazon EC2 인스턴스 메타데이터(EC2 인스턴스에서 실행 중인 경우).

리전을 찾을 수 없는 경우 SDK는 AWS 서비스 클라이언트에 구성된 리전이 없다는 예외를 발생시킵니다.

중국(베이징) 리전에 대한 특수 정보

중국(베이징) 리전에서 서비스를 이용하려면 중국(베이징)과 관련된 계정 및 자격 증명이 있어야 합니다. 중국(베이징) AWS 리전에서는 다른 리전의 계정 및 자격 증명이 작동하지 않습니다. 마찬가지로 중국(베이징) 리전의 계정 및 자격 증명은 다른 AWS 리전에서는 작동하지 않습니다. 중국(베이징) 리전에서 사용 가능한 엔드포인트 및 프로토콜에 대한 자세한 내용은 베이징 리전 엔드포인트를 참조하세요.

새 AWS 서비스에 대한 특별 정보

처음에는 일부 리전에서 새 AWS 서비스를 시작한 다음 다른 리전에서 지원할 수 있습니다. 이와 같은 경우 해당 서비스에 대한 새로운 리전에 액세스하기 위해 최신 SDK를 설치할 필요가 없습니다. 앞서 표시된 대로 새로 추가된 리전을 클라이언트 단위로 또는 전역적으로 지정할 수 있습니다.

보안 인증 정보 및 프로파일 확인

는 특정 순서로 자격 증명을 AWS SDK for .NET 검색하고 현재 애플리케이션에 사용 가능한 첫 번째 세트를 사용합니다.

보안 인증 검색 순서

1. 에 설명된 대로 AWS 서비스 클라이언트에 명시적으로 설정된 자격 증명입니다애플리케이션에서 자격 증명 및 프로필에 액세스.

   > **Note**
   >
   > 이 주제는 보안 인증을 지정하는 데 선호되는 방법이 아니므로 특별 고려 사항 섹션에 있습니다.

2. 세 변수 모두에 값이 있는 경우 `AWS_ACCESS_KEY_ID`, `AWS_SECRET_ACCESS_KEY`및 `AWS_SESSION_TOKEN` 환경 변수에서 생성되는 SessionAWSCredentials입니다.
3. 두 변수 모두에 값이 있는 경우 `AWS_ACCESS_KEY_ID` 및 `AWS_SECRET_ACCESS_KEY` 환경 변수에서 생성되는 BasicAWSCredentials입니다.
4. 두 변수 모두에 값이 있는 경우 `AWS_WEB_IDENTITY_TOKEN_FILE` 및 `AWS_ROLE_ARN` 환경 변수에서 생성된 AssumeRoleWithWebIdentityCredentials입니다.
5. AWSConfigs.AWSProfileName의 값으로 이름이 지정된 보안 인증 프로필입니다.
6. `AWS_PROFILE` 환경 변수로 이름이 지정된 보안 인증 프로필입니다.
7. `[default]` 자격 증명 프로필
8. 컨테이너 자격 증명 공급자입니다.
9. Amazon EC2 인스턴스 메타데이터입니다.

애플리케이션이 Amazon EC2 인스턴스(예: 프로덕션 환경)에서 실행되는 경우 IAM 역할을 사용하여 액세스 권한 부여에 설명된 대로 IAM 역할을 사용하세요. 그렇지 않으면 시험판 테스트와 같이 웹 애플

리케이션이 서버에서 액세스할 수 있는 자격 증명 파일 형식을 사용하는 파일에 AWS 자격 증명을 저장합니다.

환경 변수 및 자격 증명 프로파일에 대한 자세한 내용은 AWS SDKs 및 도구 참조 가이드의 환경 변수, 환경 변수 목록, 공유 구성 및 자격 증명 파일 주제를 참조하세요.

프로필 해결 방법

자격 증명에 대한 두 가지 스토리지 메커니즘을 사용하면 자격 증명을 AWS SDK for .NET 사용하도록 를 구성하는 방법을 이해하는 것이 중요합니다. AWSConfigs.AWSProfilesLocation 속성은가 자격 증명 프로파일을 AWS SDK for .NET 찾는 방법을 제어합니다.

AWSProfilesLocation	프로파일 해결 동작
null 값(설정되지 않음) 또는 비어 있음	플랫폼에서 지원하는 경우 SDK 스토어를 검색한 다음 기본 위치에서 공유 AWS 보안 인증 파일을 검색하세요. 프로필이 해당 위치 중 하나에 없으면 `~/.aws/config` (Linux 또는 macOS) 또는 `%USERPROFILE%\.aws\config` (Windows)를 검색하세요.
자격 AWS 증명 파일 형식의 파일 경로	특정 이름이 있는 프로필에 대해 지정된 파일만 검색합니다.

페더레이션 사용자 계정 보안 인증 사용

AWS SDK for .NET (AWSSDK.Core 버전 3.1.6.0 이상)를 사용하는 애플리케이션은 Active Directory Federation Services(AD FS)를 통해 연동 사용자 계정을 사용하여 Security Assertion Markup Language(SAML)를 사용하여 AWS 서비스에 액세스할 수 있습니다.

연동 액세스 지원이란 사용자가 Active Directory를 사용하여 인증할 수 있음을 뜻합니다. 임시 자격 증명은 사용자에게 자동으로 부여됩니다. 1시간 동안 유효한 이러한 임시 자격 증명은 애플리케이션이 AWS 서비스를 호출할 때 사용됩니다. SDK는 임시 자격 증명 관리 작업을 처리합니다. 도메인 병합 사용자 계정의 경우 애플리케이션에서 호출을 하지만 자격 증명이 만료되었다면 사용자는 자동으로 재인증되어 새로운 자격 증명이 부여됩니다. (도메인에 가입되지 않은 계정의 경우 재인증 전에 자격 증명을 입력하라는 메시지가 표시됩니다.)

.NET 애플리케이션에서 이 지원 기능을 사용하려면 먼저 PowerShell cmdlet을 사용하여 역할 프로필을 설정해야 합니다. 자세한 내용은 AWS Tools for Windows PowerShell 설명서를 참조하세요.

역할 프로파일을 설정한 후에는 애플리케이션에서 해당 프로필을 참조하세요. 여러 가지 방법으로 이 작업을 수행할 수 있는데, 그 중 하나는 다른 보안 인증 프로파일과 동일한 방식으로 AWSConfigs.AWSProfileName 속성을 사용하는 것입니다.

AWS Security Token Service 어셈블리(AWSSDK.SecurityToken)는 자격 AWS 증명을 얻기 위한 SAML 지원을 제공합니다. 페더레이션 사용자 계정 보안 인증을 사용하려면 애플리케이션에서 이 어셈블리를 사용할 수 있어야 합니다.

역할 또는 임시 자격 증명 지정

Amazon EC2 인스턴스에서 실행되는 애플리케이션의 경우 IAM 역할을 사용하여 액세스 권한 부여의 설명과 같이 자격 증명을 관리하는 가장 안전한 방법은 IAM 역할을 사용하는 것입니다.

조직 외부의 사용자가 소프트웨어 실행 파일을 사용할 수 있는 애플리케이션 시나리오의 경우 임시 보안 자격 증명을 사용할 수 있도록 소프트웨어를 설계하는 것이 좋습니다. AWS 리소스에 대한 제한된 액세스를 제공하는 것 외에도 이러한 자격 증명은 지정된 기간 후에 만료된다는 이점이 있습니다. 임시 보안 자격 증명에 관한 자세한 내용은 다음 내용을 확인하십시오.

- 임시 보안 자격 증명
- Amazon Cognito 자격 증명 풀

프록시 자격 증명 사용

소프트웨어가 프록시를 AWS 통해와 통신하는 경우 서비스 `Config` 클래스의 `ProxyCredentials` 속성을 사용하여 프록시에 대한 자격 증명을 지정할 수 있습니다. 서비스의 `Config` 클래스는 일반적으로 서비스의 기본 네임스페이스의 일부입니다. Amazon.CloudDirectory 네임스페이스의 AmazonCloudDirectoryConfig와 Amazon.GameLift 네임스페이스의 AmazonGameLiftConfig를 예로 들 수 있습니다.

예를 들어 Amazon S3의 경우 다음과 유사한 코드를 사용할 수 있습니다. 여기에서 `SecurelyStoredUserName` 및 `SecurelyStoredPassword`는 NetworkCredential 객체에서 지정된 프록시 사용자 이름과 암호입니다.

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

> **Note**
>
> 이전 버전의 SDK에서는 `ProxyUsername`과 `ProxyPassword`를 사용하였으나 이 속성들은 사용이 중단되었습니다.

재시도 및 제한 시간

를 AWS SDK for .NET 사용하면 AWS 서비스에 대한 HTTP 요청의 재시도 횟수와 제한 시간 값을 구성할 수 있습니다. 재시도 횟수 및 제한 시간의 기본값이 해당 애플리케이션에 적절하지 않은 경우 특정 요구사항에 맞게 조정할 수 있지만, 그렇게 함으로써 해당 애플리케이션의 작동에 어떤 영향을 미칠지 먼저 이해하는 것이 중요합니다.

재시도 횟수 및 제한 시간에 어떤 값을 사용할지 결정하려면 다음 사항을 고려하십시오.

- 네트워크 연결이 저하되거나 AWS 서비스에 연결할 수 없는 경우 AWS SDK for .NET 및 애플리케이션은 어떻게 응답해야 합니까? 호출이 신속하게 실패하길 원하는가, 아니면 호출이 사용자를 대신하여 계속 재시도하는 것이 적절한가?
- 해당 애플리케이션은 응답이 필수적인 사용자 지향 애플리케이션이나 웹 사이트인가, 아니면 지연 시간 증가를 더 많이 허용하는 백그라운드 처리 작업인가?
- 애플리케이션이 지연 시간이 짧은 안정적인 네트워크에 배포되는가, 아니면 연결성이 불안정한 원격 위치에 배포되는가?

재시도

개요

는 서버 측 제한 또는 연결 끊김으로 인해 실패한 요청을 재시도할 AWS SDK for .NET 수 있습니다. 서비스 구성 클래스에는 서비스 클라이언트의 재시도 동작을 지정하는 데 사용할 수 있는 두 가지 속성이 있습니다. 서비스 구성 클래스는 AWS SDK for .NET API 참조의 추상 Amazon.Runtime.ClientConfig 클래스에서 다음과 같은 속성을 상속합니다.

- `RetryMode`는 Amazon.Runtime.RequestRetryMode 열거에 정의된 세 가지 재시도 모드 중 하나를 지정합니다.

  애플리케이션의 기본값은 `AWS_RETRY_MODE` 환경 변수 또는 공유 AWS 구성 파일의 retry_mode 설정을 사용하여 제어할 수 있습니다.

- `MaxErrorRetry`는 서비스 클라이언트 수준에서 허용되는 재시도 횟수를 지정합니다. SDK는 작업이 실패하고 예외가 발생하기 전에 지정된 횟수만큼 작업을 재시도합니다.

  애플리케이션의 기본값은 `AWS_MAX_ATTEMPTS` 환경 변수 또는 공유 AWS 구성 파일의 max_attempts 설정을 사용하여 제어할 수 있습니다.

이러한 속성에 대한 자세한 설명은 AWS SDK for .NET API 참조의 추상 Amazon.Runtime.ClientConfig 클래스에서 확인할 수 있습니다. `RetryMode`의 각 값은 기본적으로 다음 표에 표시된 것처럼 `MaxErrorRetry`의 특정 값에 해당합니다.

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

동작

애플리케이션이 시작되는 시점

애플리케이션이 시작되면 SDK에서 `RetryMode` 및 `MaxErrorRetry`의 기본값을 구성합니다. 이러한 기본값은 다른 값을 지정하지 않는 한 서비스 클라이언트를 생성할 때 사용됩니다.

- 환경에 속성이 설정되지 않은 경우 `RetryMode`의 기본값은 레거시로 구성되고 `MaxErrorRetry`의 기본값은 위 표의 해당 값으로 구성됩니다.
- 환경에서 재시도 모드가 설정된 경우 해당 값이 `RetryMode`의 기본값으로 사용됩니다. 환경에 최대 오류 값도 설정되어 있지 않은 경우(다음 설명 참조) `MaxErrorRetry`의 기본값은 위 표의 해당 값으로 구성됩니다.
- 환경에서 최대 오류 값이 설정된 경우 해당 값이 `MaxErrorRetry`의 기본값으로 사용됩니다. Amazon DynamoDB는 이 규칙의 예외입니다. `MaxErrorRetry`에 대한 DynamoDB의 기본 값은 항상 위 표의 값입니다.

애플리케이션이 실행될 때

서비스 클라이언트를 생성할 때 앞에서 설명한 대로 RetryMode 및 MaxErrorRetry의 기본값을 사용하거나 다른 값을 지정할 수 있습니다. 다른 값을 지정하려면 서비스 클라이언트를 생성할 때 AmazonDynamoDBConfig 또는 AmazonSQSConfig와 같은 서비스 구성 객체를 생성하여 포함시킵니다.

서비스 클라이언트가 생성된 이후에는 그 값을 바꿀 수 없습니다.

고려 사항

재시도가 이루어지면 해당 요청의 지연 시간이 늘어납니다. 총 요청 지연 시간 및 오류율에 대한 애플리케이션의 제한에 따라 재시도를 구성해야 합니다.

시간 초과

를 AWS SDK for .NET 사용하면 서비스 클라이언트 수준 및 메서드 호출당 요청 제한 시간을 구성할 수 있습니다. 제한 시간을 구성하는 두 가지 메커니즘이 있으며, 이는 후속 섹션에서 다룹니다.

- 비동기 직접 호출을 사용하는 경우 메서드의 CancellationToken 파라미터를 사용할 수 있습니다.
- .NET Framework에서 동기 직접 호출을 사용하는 경우 추상 Amazon.Runtime.ClientConfig 클래스의 Timeout 및 ReadWriteTimeout 속성을 사용할 수 있습니다.

제한 시간에 **CancellationToken** 파라미터 사용

를 AWS SDK for .NET 사용하면 CancellationToken 파라미터를 사용하여 비동기 호출에 대한 요청 제한 시간을 구성할 수 있습니다. 다음 코드 조각은 예제를 보여 줍니다. 요청이 10초 이내에 완료되지 않으면 코드에서 System.Threading.Tasks.TaskCanceledException이 발생합니다.

```
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);
}
```

제한 시간에 **Timeout** 및 **ReadWriteTimeout** 속성 사용

> **Note**
>
> `Timeout` 속성은 비동기 직접 호출에 영향을 주지 않습니다. 비동기 직접 호출을 사용하는 경우 제한 시간에 `CancellationToken` 파라미터 사용을 대신 참조하세요.

를 AWS SDK for .NET 사용하면 서비스 클라이언트 수준에서 요청 제한 시간 및 소켓 읽기/쓰기 제한 시간 값을 구성할 수 있습니다. 이러한 값은 추상 Amazon.Runtime.ClientConfig 클래스의 `Timeout` 및 `ReadWriteTimeout` 속성에 지정됩니다. 이러한 값은 AWS 서비스 클라이언트 객체에 의해 생성된 HttpWebRequest 객체의 `Timeout` 및 `ReadWriteTimeout` 속성으로 전달됩니다. `Timeout` 값은 100초, `ReadWriteTimeout` 값은 300초로 기본 설정되어 있습니다.

해당 네트워크의 지연 시간이 길거나 작업 재시도를 유발하는 조건이 존재하는 경우 제한 시간 및 재시도 횟수의 값을 높이면 일부 SDK 작업이 응답하지 않는 것처럼 보일 수 있습니다.

> **Note**
>
> 이동식 클래스 라이브러리(PCL)를 대상으로 AWS SDK for .NET 하는의 버전은 클래스 대신 HttpClient `HttpWebRequest` 클래스를 사용하며 제한 시간 속성만 지원합니다.

다음은 기본 설정된 제한 시간 값에 대한 예외입니다. 제한 시간 값을 명시적으로 설정하면 이 값들은 무시됩니다.

- `Timeout` 및 `ReadWriteTimeout`은 호출되는 메서드가 AmazonS3Client.PutObjectAsync(), AmazonS3Client.UploadPartAsync(), AmazonGlacierClient.UploadArchiveAsync()와 같은 스트림을 업로드하는 경우 최대값으로 설정됩니다.
- .NET Framework를 대상으로 AWS SDK for .NET 하는의 버전은 `Timeout` 및 `ReadWriteTimeout`를 모든 AmazonS3Client 및 AmazonGlacierClient 객체의 최대값으로 설정합니다.

- 휴대용 클래스 라이브러리(PCL) 및 .NET Core를 대상으로 AWS SDK for .NET 하는의 버전은 모든 AmazonS3Client 및 AmazonGlacierClient 객체의 최대값`Timeout`으로 설정됩니다.

다음 예제에서는 표준 재시도 모드, 최대 3회의 재시도, 제한 시간 10초, 쓰기/읽기 제한 시간 10초를 지정하는 방법을 보여줍니다(해당하는 경우). AmazonS3Client 생성자에는 AmazonS3Config 객체가 제공됩니다.

```
var s3Client = new AmazonS3Client(
  new AmazonS3Config
  {
    Timeout = TimeSpan.FromSeconds(10),
    // NOTE: The following property is obsolete for
    //       versions of the AWS SDK for .NET that target .NET Core.
    ReadWriteTimeout = TimeSpan.FromSeconds(10),
    RetryMode = RequestRetryMode.Standard,
    MaxErrorRetry = 3
  });
```

관찰성

관찰성은 시스템의 현재 상태를 내보내는 데이터에서 추론할 수 있는 범위입니다. 방출되는 데이터를 일반적으로 원격 측정이라고 합니다.

는 두 가지 일반적인 원격 측정 신호, 지표 및 추적과 로깅을 제공할 AWS SDK for .NET 수 있습니다. TelemetryProvider를 연결하여 원격 측정 데이터를 관찰성 백엔드(예: AWS X-Ray 또는 Amazon CloudWatch)로 전송한 다음 조치를 취할 수 있습니다.

기본적으로 원격 측정 신호는 SDK에서 비활성화됩니다. 이 주제에서는 원격 측정 출력을 활성화하고 구성하는 방법을 설명합니다.

추가 리소스

관찰성 활성화 및 사용에 대한 자세한 내용은 다음 리소스를 참조하세요.

- OpenTelemetry
- 블로그 게시물 Enhancing Observability in the AWS SDK for .NET with OpenTelemetry.
- 블로그 게시물 Announcing the general availability of AWS .NET OpenTelemetry library.
- OpenTelemetry용 익스포터

- 의 관찰성 예제는 Tools for PowerShell 사용 설명서의 관찰성을 AWS Tools for PowerShell참조하세요.

`TelemetryProvider` 구성

다음 예제와 같이 모든 서비스 클라이언트 또는 개별 클라이언트에 대해 애플리케이션에서 `TelemetryProvider`를 전역적으로 구성할 수 있습니다. the section called “원격 측정 공급자” 섹션에는 SDK와 함께 제공되는 구현에 대한 정보를 포함하여 원격 측정 구현에 대한 정보가 포함되어 있습니다.

기본 글로벌 원격 측정 공급자 구성

기본적으로 모든 서비스 클라이언트는 전역적으로 사용 가능한 원격 측정 공급자를 사용하려고 시도합니다. 이렇게 하면 공급자를 한 번 설정할 수 있으며, 모든 클라이언트가 공급자를 사용합니다. 이 작업은 서비스 클라이언트를 생성하기 전에 한 번만 수행해야 합니다.

다음 코드 조각은 글로벌 원격 측정 공급자를 설정하는 방법을 보여 줍니다. 그런 다음 Amazon S3 서비스 클라이언트를 생성한 다음 실패하는 작업을 수행하려고 시도합니다. 코드는 애플리케이션에 추적 및 지표를 모두 추가합니다. 이 코드는 `OpenTelemetry.Exporter.Console` 및 `OpenTelemetry.Instrumentation.AWS`의 NuGet 패키지를 사용합니다.

> **Note**
>
> 인증 AWS IAM Identity Center 에를 사용하는 경우 `AWSSDK.SSO` 및 도 추가해야 합니다`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();
```

특정 서비스 클라이언트에 대한 원격 측정 공급자 구성

특정 원격 측정 공급자(글로벌 공급자 제외)를 사용하여 개별 서비스 클라이언트를 구성할 수 있습니다. 이렇게 하려면 서비스 클라이언트 생성자의 Config 객체 `TelemetryProvider` 클래스를 사용합니다. 예를 들어 AmazonS3Config를 참조하고 `TelemetryProvider` 속성을 찾습니다. 사용자 지정 원격 측정 구현에 대한 자세한 내용은 the section called “원격 측정 공급자” 섹션을 참조하세요.

주제

- Metrics
- 원격 측정 공급자

Metrics

다음 표에는 SDK가 내보내는 원격 측정 지표가 나열되어 있습니다. 지표를 관찰할 수 있도록 원격 측정 공급자를 구성합니다.

방출되는 지표는 무엇입니까?

메트릭 이름	단위	유형	속성	설명
client.call.duration	s	히스토그램(Histogram)	rpc.service, rpc.method	전체 통화 기간(재시도 및 요청 및 응답 본문을 보내거나 받는 시간 포함)
client.uptime	s	히스토그램(Histogram)	rpc.service	클라이언트가 생성된 이후 경과한 시간
client.call.attempts	{시도}	MonotonCounter	rpc.service, rpc.method	개별 작업에 대한 시도 횟수
client.call.errors	{오류}	MonotonCounter	rpc.service, rpc.method, exception.type	작업의 오류 수
client.call.attempt_duration	s	히스토그램(Histogram)	rpc.service, rpc.method	서비스에 연결하고, 요청을 보내고, HTTP 상태 코드 및 헤더를 다시 가져오는 데 걸리는 시간(전송 대기 시간 포함)
client.call.resolve_endpoint_duration	s	히스토그램(Histogram)	rpc.service, rpc.method	요청에 대한 엔드포인트(DNS가 아닌 엔드포인트 해석기)를 해결하는 데 걸리는 시간
client.call.serialization_duration	s	히스토그램(Histogram)	rpc.service, rpc.method	메시지 본문을 직렬화하는 데 걸리는 시간
client.call.deserialization_duration	s	히스토그	rpc.service, rpc.method	메시지 본문을 역직렬화하는 데 걸리는 시간

메트릭 이름	단위	유형	속성	설명
		램(His togram)		
client.call.auth.s igning_기간	s	히스 토그 램(His togram)	rpc.service, rpc.method	요청에 서명하는 데 걸리는 시간
client.call.auth.r esolve_id entity_duration	s	히스 토그 램(His togram)	rpc.service, rpc.method	자격 증명 공급자로부터 자격 증명(예: AWS 자격 증명 또는 보유자 토큰)을 획득하는 데 걸리는 시간
client.http.bytes_ sent	By	Monoton Counter	server.address	HTTP 클라이언트가 보낸 총 바이트 수
client.http.bytes_ received	By	Monoton Counter	server.address	HTTP 클라이언트가 수신한 총 바이트 수

다음은 열 설명입니다.

- 지표 이름 - 내보낸 지표의 이름입니다.
- 단위 - 지표의 측정 단위입니다. 단위는 UCUM 대/소문자를 구분하는(“c/s”) 표기법으로 제공됩니다.
- 유형 - 지표를 캡처하는 데 사용되는 계측의 유형입니다.
- 속성 - 지표로 내보낸 속성(차원) 세트입니다.
- 설명 - 지표가 측정하는 항목에 대한 설명입니다.

원격 측정 공급자

SDK는 다음 섹션에 설명된 원격 측정 공급자로 OpenTelemetry를 구현합니다.

특정 원격 측정 요구 사항이 있거나, 이미 원격 측정 솔루션을 염두에 두고 있거나, 원격 측정 데이터를 캡처하고 처리하는 방법을 세밀하게 제어해야 하는 경우 자체 원격 측정 공급자를 구현할 수도 있습니다.

TelemetryProvider 클래스에 자체 구현을 등록합니다. 다음은 자체 `TracerProvider` 및를 등록하는 방법의 간단한 예입니다`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());
```

주제

- OpenTelemetry 기반 원격 측정 공급자 구성

OpenTelemetry 기반 원격 측정 공급자 구성

에는 OpenTelemetry 기반 원격 측정 공급자의 구현이 AWS SDK for .NET 포함되어 있습니다. 이 공급자를 글로벌 원격 측정 공급자로 설정하는 방법에 대한 자세한 내용은 `TelemetryProvider` 구성 섹션을 참조하세요. 이 원격 측정 공급자를 사용하려면 프로젝트에 다음 리소스가 필요합니다.

- OpenTelemetry.Instrumentation.AWS NuGet 패키지입니다.
- OTLP 또는 콘솔과 같은 원격 측정 익스포터입니다. 자세한 내용은 OpenTelemetry 설명서의 익스포터 섹션을 참조하세요.

SDK에 포함된 OpenTelemetry 구현은 HTTPS 요청, 자격 증명 및 압축에 대한 추적 양을 줄이도록 구성할 수 있습니다. 이렇게 하려면 다음과 같이 `SuppressDownstreamInstrumentation` 옵션을 `true`로 설정합니다.

```
Sdk.CreateTracerProviderBuilder()
```

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

이 공급자에 대한 자세한 내용은 OpenTelemetry를 AWS SDK for .NET 사용하여의 관찰성 향상 블로그 게시물을 참조하세요.

사용자 및 역할에 대한 추가 정보

에서 .NET 개발을 수행 AWS 하거나에서 .NET 애플리케이션을 실행 AWS하려면 이러한 작업에 적합한 사용자, 권한 세트 및 서비스 역할의 조합이 있어야 합니다.

만드는 특정 사용자, 권한 집합 및 서비스 역할과 이를 사용하는 방식은 애플리케이션의 요구 사항에 따라 달라집니다. 다음은 사용 이유 및 생성 방법에 대한 몇 가지 추가 정보입니다.

사용자 및 권한 집합

장기 보안 인증 정보가 있는 IAM 사용자 계정을 사용하여 AWS 서비스에 액세스할 수는 있지만 이는 더 이상 모범 사례가 아니므로 피해야 합니다. 개발 중에도에서 사용자 및 권한 세트를 생성하고 자격 증명 소스에서 제공하는 임시 자격 증명을 AWS IAM Identity Center 사용하는 것이 가장 좋습니다.

개발 시에는 직접 생성했거나 를 사용하여 인증 AWS에서 부여받은 사용자를 사용할 수 있습니다. 적절한 AWS Management Console 권한이 있는 경우 해당 사용자에 대해 최소 권한이 있는 다른 권한 세트를 생성하거나 개발 프로젝트에 대해 특별히 새 사용자를 생성하여 최소 권한이 있는 권한 세트를 제공할 수도 있습니다. 어떤 방법을 선택할지는 상황에 따라 달라집니다.

이러한 사용자 및 권한 세트와 생성 방법에 대한 자세한 내용은 AWS SDK 및 도구 참조 가이드의 인증 및 액세스 및 AWS IAM Identity Center 사용 설명서의 시작을 참조하세요.

서비스 역할

사용자를 대신하여 AWS 서비스에 액세스하도록 AWS 서비스 역할을 설정할 수 있습니다. 이러한 유형의 액세스는 여러 사람이 원격으로 애플리케이션을 실행하는 경우(예: 이러한 목적으로 생성한 Amazon EC2 인스턴스)에 적합합니다.

서비스 역할을 만드는 프로세스는 상황에 따라 다르지만 기본적으로 다음과 같습니다.

1. 에 로그인 AWS Management Console 하고 https://console.aws.amazon.com/iam/://https://https://https://https://://https://://https://://https://://https://://https://://

2. 역할을 선택한 다음 역할 생성을 선택합니다.
3. AWS 서비스를 선택하고, EC2(예: EC2)를 찾아 선택한 다음 EC2 사용 사례(예: EC2)를 선택합니다.
4. 다음: 권한을 선택하고 애플리케이션이 사용할 AWS 서비스에 적합한 정책을 선택합니다.

> **Warning**
>
> 이 정책은 계정의 거의 모든 항목에 대한 읽기 및 쓰기 권한을 허용하므로 AdministratorAccess 정책을 선택하지 않습니다.

5. 다음: 태그를 선택하고 원하는 태그를 입력합니다.

   IAM 사용 설명서의 AWS 리소스 태그를 사용하여 액세스 제어에서 태그에 대한 정보를 찾을 수 있습니다.
6. 다음: 검토를 선택하고 역할 이름과 역할 설명을 입력합니다. 그런 다음 역할 생성을 선택합니다.

IAM 사용 설명서의 ID(사용자, 그룹 및 역할)에서 IAM 역할에 대한 개략적인 정보를 찾을 수 있습니다. 역할에 대한 자세한 내용은 해당 설명서의 IAM 역할 주제를 참조하세요.

역할에 대한 추가 정보

- Amazon Elastic Container Service(Amazon ECS) 작업에 작업용 IAM 역할을 사용하십시오.
- Amazon EC2 인스턴스에서 실행 중인 애플리케이션에 IAM 역할을 사용하십시오.

AWS SDK for .NET 프로젝트의 고급 구성

이 섹션의 주제에는 관심이 있을 만한 추가 구성 작업 및 방법에 대한 정보가 포함되어 있습니다.

주제

- AWSSDK.Extensions.NETCore.Setup 및 IConfiguration 인터페이스 사용
- 기타 애플리케이션 파라미터 구성
- 에 대한 구성 파일 참조 AWS SDK for .NET

AWSSDK.Extensions.NETCore.Setup 및 IConfiguration 인터페이스 사용

(이 주제의 이전 제목은 ".NET Core AWS SDK for .NET 로 구성"입니다.)

.NET Core의 가장 큰 변경 사항 중 하나는 `ConfigurationManager`의 제거와 .NET Framework 및 ASP.NET 애플리케이션에서 사용되던 표준 `app.config` 및 `web.config` 파일의 제거입니다.

.NET Core 구성은 구성 공급자에 의해 설정된 키-값 페어를 기반으로 합니다. 구성 공급자는 명령줄 인수, 디렉터리 파일, 환경 변수, 설정 파일 등 다양한 구성 소스에서 구성 데이터를 키-값 페어로 읽어 들입니다.

> **Note**
>
> 자세한 내용은 ASP.NET Core 구성을 참조하십시오.

.NET Core AWS SDK for .NET 에서를 쉽게 사용할 수 있도록 AWSSDK.Extensions.NETCore.Setup NuGet 패키지를 사용할 수 있습니다. 많은 .NET Core 라이브러리와 마찬가지로 `IConfiguration` 인터페이스에 확장 메서드를 추가하여 AWS 구성을 원활하게 만듭니다.

이 패키지의 소스 코드는 https://github.com/aws/aws-sdk-net/tree/main/extensions/src/AWSSDK.Extensions.NETCore.Setup의 GitHub에 있습니다.

AWSSDK.Extensions.NETCore.Setup 사용법

Visual Studio의 ASP.NET Core 웹 애플리케이션 템플릿을 사용하거나 .NET Core CLI에서 `dotnet new mvc ...`를 실행하여 수행할 수 있는 ASP.NET Core MVC(모델-뷰-컨트롤러) 애플리케이션을 생성한다고 가정하겠습니다. 이러한 애플리케이션을 생성하는 경우 `Startup.cs`의 생성자는 `appsettings.json`과 같은 구성 공급자의 다양한 입력 소스에서 읽어 구성을 처리합니다.

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

`Configuration` 객체를 사용하여 AWS 옵션을 얻으려면 먼저 `AWSSDK.Extensions.NETCore.Setup` NuGet 패키지를 추가합니다. 그런 다음, 다음에 설명한 대로 구성 파일에 해당 옵션을 추가합니다.

프로젝트에 추가된 파일 중 하나가 `appsettings.Development.json`입니다. 이는 `EnvironmentName`을 개발로 설정한 것에 해당합니다. 개발 중에는 로컬 테스트 중에만 읽을 수 있는 이 파일에 구성을 입력합니다. 가 프로덕션으로 `EnvironmentName` 설정된 Amazon EC2 인스턴스를 배포하면이 파일은 무시되고는 Amazon EC2 인스턴스에 대해 구성된 IAM 자격 증명 및 리전으로 AWS SDK for .NET 돌아갑니다.

다음 구성에서는 AWS 설정 값을 제공하기 위해 프로젝트의 appsettings.Development.json 파일에서 추가할 수 있는 값들의 예를 보여줍니다.

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

CSHTML 파일의 설정에 액세스하려면 Configuration 명령을 사용합니다.

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

<h1>Contact</h1>

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

코드에서 파일에 설정된 AWS 옵션에 액세스하려면에 추가된 GetAWSOptions 확장 방법을 호출합니다IConfiguration.

이 옵션을 통해 서비스 클라이언트를 생성하려면 CreateServiceClient를 호출합니다. 다음 예제에서는 Amazon S3 서비스 클라이언트를 생성하는 방법을 보여줍니다. (AWSSDK.S3 NuGet 패키지를 프로젝트에 추가해야 합니다.)

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

service1의 구성에는 us-west-2 리전이 포함되고 service2의 구성에 특수 엔드포인트 URL이 포함된 다음 예제와 같이 appsettings.Development.json 파일의 여러 항목을 사용하여 설정이 호환되지 않는 여러 서비스 클라이언트를 생성할 수도 있습니다.

```
{
  "service1": {
    "Profile": "default",
    "Region": "us-west-2"
```

```
  },
  "service2": {
    "Profile": "default",
    "ServiceURL": "URL"
  }
}
```

그러면 JSON 파일의 항목을 사용하여 특정 서비스에 대한 옵션을 가져올 수 있습니다. 예를 들어 `service1`의 설정을 얻으려면 다음을 사용합니다.

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

appsettings 파일에서 허용되는 값

다음 앱 구성 값은 `appsettings.Development.json` 파일에서 설정할 수 있습니다. 제시된 필드 이름은 소문자이어야 합니다. 이러한 설정에 대한 자세한 내용은 `AWS.Runtime.ClientConfig` 클래스를 참조하세요.

- 리전
- 프로필
- ProfilesLocation
- SignatureVersion
- RegionEndpoint
- UseHttp
- ServiceURL
- AuthenticationRegion
- AuthenticationServiceName
- MaxErrorRetry
- LogResponse
- BufferSize
- ProgressUpdateInterval
- ResignRetries
- AllowAutoRedirect
- LogMetrics
- DisableLogging

- UseDualstackEndpoint

ASP.NET Core 종속성 주입

AWSSDK.Extensions.NETCore.Setup NuGet 패키지는 ASP.NET Core의 새로운 종속성 주입 시스템과도 통합됩니다. 애플리케이션의 `Startup` 클래스에서 `ConfigureServices` 메서드는 MVC 서비스가 추가되는 곳입니다. 애플리케이션에서 개체 프레임워크를 사용하는 경우 이 메서드는 그 프레임워크가 초기화되는 곳이기도 합니다.

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

> **Note**
>
> .NET Core의 종속성 주입에 대한 배경 정보는 .NET Core 설명서 사이트에서 사용할 수 있습니다.

`AWSSDK.Extensions.NETCore.Setup` NuGet 패키지는 종속성 주입에 AWS 서비스를 추가하는 데 사용할 수 `IServiceCollection` 있는 새로운 확장 방법을에 추가합니다. 다음 코드는에서 읽는 AWS 옵션을 추가하여 서비스 목록에 Amazon S3 및 DynamoDB를 `IConfiguration` 추가하는 방법을 보여줍니다. (AWSSDK.S3 및 AWSSDK.DynamoDBv2 NuGet 패키지를 프로젝트에 추가해야 합니다.)

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

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

현재 MVC 컨트롤러가 자신의 생성자에서 `IAmazonS3` 또는 `IAmazonDynamoDB`를 파라미터로 사용하는 경우 종속성 주입 시스템은 그러한 서비스로 전달됩니다.

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

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

    ...

}
```

기타 애플리케이션 파라미터 구성

> **Note**
>
> 이 항목의 정보는 .NET Framework를 기반으로 하는 프로젝트에만 해당됩니다. .NET Core 기반 프로젝트에는 기본적으로 `App.config` 및 `Web.config` 파일이 표시되지 않습니다.

열어서 .NET Framework 콘텐츠 확인

구성할 수 있는 애플리케이션 파라미터는 여러 가지입니다.

- `AWSLogging`
- `AWSLogMetrics`
- `AWSRegion`
- `AWSResponseLogging`
- `AWS.DynamoDBContext.TableNamePrefix`
- `AWS.S3.UseSignatureVersion4`
- `AWSEndpointDefinition`
- AWS 서비스 생성 엔드포인트

이 파라미터들은 애플리케이션의 `App.config` 또는 `Web.config` 파일에서 구성할 수 있습니다. AWS SDK for .NET API를 사용하여 이를 구성할 수도 있지만 애플리케이션의 `.config` 파일을 사용하는 것이 좋습니다. 여기에 두 방식이 모두 설명되어 있습니다.

이 주제 후반부에 설명된 <aws> 요소 사용에 대한 자세한 내용은 AWS SDK for .NET에 대한 구성 파일 참조를 참조하세요.

AWSLogging

SDK에서 이벤트를 로깅하는 방식을 구성합니다. 예를 들면 권장할 만한 방식은 다음과 같이 <logging> 요소의 하위 요소인 <aws> 요소를 사용하는 것입니다.

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

대안:

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

가능한 값은 다음과 같습니다.

None

이벤트 로깅을 해제합니다. 기본값입니다.

log4net

log4net을 사용하여 로깅합니다.

SystemDiagnostics

System.Diagnostics 클래스를 사용하여 로깅합니다.

logTo 속성에 대해 쉼표로 구분된 여러 값을 설정할 수 있습니다. 다음 예제에서는 log4net 파일에서 System.Diagnostics 및 .config 로깅을 설정합니다.

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

대안:

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

또는 AWS SDK for .NET API를 사용하여 LoggingOptions 열거의 값을 결합하고 AWSConfigs.Logging 속성을 설정합니다.

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

이 설정에 대한 변경 사항은 새 AWS 클라이언트 인스턴스에만 적용됩니다.

AWSLogMetrics

SDK가 성능 측정치를 로깅해야 하는지 여부를 지정합니다. `.config` 파일에서 측정치 로깅 구성을 설정하려면 `logMetrics` 요소의 하위 요소인 `<logging>` 요소에서 `<aws>` 속성 값을 설정합니다.

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

또는 다음과 같이 `AWSLogMetrics` 섹션에서 `<appSettings>` 키를 설정할 수 있습니다.

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

또는 AWS SDK for .NET API를 사용하여 지표 로깅을 설정하려면 AWSConfigs.LogMetrics 속성을 설정합니다.

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

이 설정에서는 모든 클라이언트/구성에 대해 기본값인 `LogMetrics` 속성을 구성합니다. 이 설정에 대한 변경 사항은 새 AWS 클라이언트 인스턴스에만 적용됩니다.

AWSRegion

AWS 리전을 명시적으로 지정하지 않은 클라이언트의 기본 리전을 구성합니다. `.config` 파일에서 리전을 설정할 때 권장할 만한 방식은 다음과 같이 `region` 요소에서 `aws` 속성 값을 설정하는 것입니다.

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

또는 다음과 같이 `<appSettings>` 섹션에서 AWSRegion 키를 설정할 수 있습니다.

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

또는 AWS SDK for .NET API를 사용하여 리전을 설정하려면 AWSConfigs.AWSRegion 속성을 설정합니다.

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

특정 리전에 대한 AWS 클라이언트 생성에 대한 자세한 내용은 AWS 리전 선택을 참조하세요. 이 설정에 대한 변경 사항은 새 AWS 클라이언트 인스턴스에만 적용됩니다.

AWSResponseLogging

SDK에서 서비스 응답을 로깅해야 할 경우에 구성. 가능한 값은 다음과 같습니다.

Never

서비스 응답을 로깅하지 않음. 기본값입니다.

Always

서비스 응답을 항상 로깅.

OnError

오류가 발생할 때만 서비스 응답을 로깅.

`.config` 파일에서 서비스 로깅 구성을 설정하는 경우 권장할 만한 방식은 `logResponses` 요소의 하위 요소인 `<logging>` 요소에서 `<aws>` 속성 값을 설정하는 것입니다.

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

또는 다음과 같이 `<appSettings>` 섹션에서 AWSResponseLogging 키를 설정할 수 있습니다.

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

또는 AWS SDK for .NET API를 사용하여 서비스 로깅을 설정하려면 AWSConfigs.ResponseLogging 속성을 ResponseLoggingOption 열거의 값 중 하나로 설정합니다.

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

이 설정의 변경 사항은 즉시 적용됩니다.

AWS.DynamoDBContext.TableNamePrefix

수동으로 구성되지 않은 경우 TableNamePrefix에서 사용할 기본 DynamoDBContext를 구성합니다.

.config 파일에서 테이블 이름 접두사를 설정하는 경우 권장할 만한 방식은 그 자체로 tableNamePrefix 요소의 하위 요소이자 <dynamoDBContext> 요소의 하위 요소인 <dynamoDB> 요소에서 <aws> 속성 값을 설정하는 것입니다.

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

또는 다음과 같이 AWS.DynamoDBContext.TableNamePrefix 섹션에서 <appSettings> 키를 설정할 수 있습니다.

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

또는 API로 테이블 이름 접두사를 설정하려면 AWSConfigs.DynamoDBContextTableNamePrefix 속성을 AWS SDK for .NET 설정합니다.

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

이 설정을 변경하면 DynamoDBContextConfig 및 DynamoDBContext의 새로 생성된 인스턴스에만 적용됩니다.

AWS.S3.UseSignatureVersion4

Amazon S3 클라이언트가 서명 버전 4 서명을 요청에 사용해야 하는지 여부를 구성합니다.

.config 파일에서 Amazon S3의 서명 버전 4 서명을 설정하려면 <aws> 요소의 하위 요소인 <s3> 요소의 useSignatureVersion4 속성을 설정하는 것이 좋습니다.

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

또는 다음과 같이 <appSettings> 섹션에서 AWS.S3.UseSignatureVersion4 키를 true로 설정할 수 있습니다.

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

또는 AWS SDK for .NET API를 사용하여 서명 버전 4 서명을 설정하려면 AWSConfigs.S3UseSignatureVersion4 속성을 로 설정합니다. `true`

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

이 설정은 `false`로 기본 설정되어 있으나 어떤 경우 또는 일부 리전에서는 서명 버전 4로 기본 설정할 수도 있습니다. 설정이 `true`인 경우 모든 요청에 서명 버전 4를 사용합니다. 이 설정에 대한 변경 사항은 새로운 Amazon S3 클라이언트 인스턴스에만 적용됩니다.

AWSEndpointDefinition

SDK에서 리전 및 엔드포인트를 정의하는 사용자 지정 구성 파일을 사용해야 하는지 여부를 구성합니다.

`.config` 파일에서 엔드포인트 정의 파일을 설정하려면 `endpointDefinition` 요소에서 `<aws>` 속성 값을 설정하는 것이 좋습니다.

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

또는 다음과 같이 `<appSettings>` 섹션에서 AWSEndpointDefinition 키를 설정할 수 있습니다.

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

또는 AWS SDK for .NET API를 사용하여 엔드포인트 정의 파일을 설정하려면 AWSConfigs.EndpointDefinition 속성을 설정합니다.

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

파일 이름이 제공되지 않는 경우 사용자 지정 구성 파일은 사용되지 않습니다. 이 설정에 대한 변경 사항은 새 AWS 클라이언트 인스턴스에만 적용됩니다. `endpoints.json` 파일은에서 사용할 수 있습니다https://github.com/aws/aws-sdk-net/blob/main/sdk/src/Core/endpoints.json.

AWS 서비스 생성 엔드포인트

일부 AWS 서비스는 리전 엔드포인트를 사용하는 대신 자체 엔드포인트를 생성합니다. 이러한 서비스의 클라이언트는 해당 서비스 및 리소스에 고유한 서비스 URL을 사용합니다. 이러한 서비스의 두 가지 예는 Amazon CloudSearch 및 입니다 AWS IoT. 다음 예제에서는 그러한 서비스에 대한 엔드포인트를 얻는 방법을 보여줍니다.

Amazon CloudSearch 엔드포인트 예제

Amazon CloudSearch 클라이언트는 Amazon CloudSearch 구성 서비스에 액세스하는 데 사용됩니다. Amazon CloudSearch 구성 서비스를 사용하여 검색 도메인을 생성, 구성 및 관리합니다. 검색 도메인을 생성하려면 CreateDomainRequest 객체를 생성하고 DomainName 속성을 제공합니다. 요청 객체를 사용하여 AmazonCloudSearchClient 객체를 생성합니다. CreateDomain 메서드를 호출합니다. 이 호출에서 반환되는 CreateDomainResponse 객체에는 DocService 및 SearchService 엔드포인트가 모두 있는 DomainStatus 속성이 포함되어 있습니다. AmazonCloudSearchDomainConfig 객체를 생성하고 이를 사용하여 AmazonCloudSearchDomainClient 클래스의 DocService 및 SearchService 인스턴스를 초기화합니다.

```
// 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 엔드포인트 예제

엔드포인트를 가져오려면 AmazonIoTClient 객체를 AWS IoT생성하고 DescribeEndPoint 메서드를 호출합니다. 반환되는 DescribeEndPointResponse 객체에는 `EndpointAddress`가 포함되어 있습니다. AmazonIotDataConfig 객체를 생성하고 `ServiceURL` 속성을 설정한 다음 이 객체를 사용하여 AmazonIotDataClient 클래스를 인스턴스화합니다.

```
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");
}
```

에 대한 구성 파일 참조 AWS SDK for .NET

> **Note**
>
> 이 항목의 정보는 .NET Framework를 기반으로 하는 프로젝트에만 해당됩니다. .NET Core 기반 프로젝트에는 기본적으로 `App.config` 및 `Web.config` 파일이 표시되지 않습니다.

열어서 .NET Framework 콘텐츠 확인

.NET 프로젝트의 `App.config` 또는 `Web.config` 파일을 사용하여 AWS 자격 증명, 로깅 옵션, AWS 서비스 엔드포인트 및 AWS 리전과 같은 AWS 설정과 Amazon DynamoDB, Amazon EC2 및 Amazon S3와 같은 AWS 서비스에 대한 일부 설정을 지정할 수 있습니다. 다음 내용은 `App.config` 또는 `Web.config` 파일의 적당한 형식을 설정하여 이러한 유형의 설정을 지정하는 방법을 기술한 것입니다.

> **Note**
>
> `App.config` 또는 `Web.config` 파일의 `<appSettings>` 요소를 계속 사용하여 AWS 설정을 지정할 수 있지만 이 주제의 뒷부분에 설명된 대로 `<configSections>` 및 `<aws>` 요소를 사용하는 것이 좋습니다. `<appSettings>` 요소에 대한 자세한 내용은 애플리케이션 구성의 `<appSettings>` 요소 예제를 참조하세요. AWS SDK for .NET

> **Note**
>
> 코드 파일에서 다음 `AWSConfigs` 클래스 속성을 계속 사용하여 AWS 설정을 지정할 수 있지만 다음 속성은 더 이상 사용되지 않으며 향후 릴리스에서 지원되지 않을 수 있습니다.
>
> - `DynamoDBContextTableNamePrefix`
> - `EC2UseSignatureVersion4`

- LoggingOptions
- LogMetrics
- ResponseLoggingOption
- S3UseSignatureVersion4

일반적으로 코드 파일의 AWSConfigs 클래스 속성을 사용하여 AWS 설정을 지정하는 대신 또는 App.config Web.config 파일의 <configSections> 및 <aws> 요소를 사용하여이 주제의 뒷부분에 설명된 대로 AWS 설정을 지정하는 것이 좋습니다. 이전 속성에 대한 자세한 내용은 애플리케이션 구성의 AWSConfigs 코드 예제를 참조하세요. AWS SDK for .NET

주제

- AWS 설정 섹션 선언
- 허용되는 요소
- 요소 참조

AWS 설정 섹션 선언

<aws> 요소 내에서 App.config 또는 Web.config 파일에 AWS 설정을 지정합니다. <aws> 요소를 사용하기 전에 다음 예제와 같이 <section> 요소(<configSections> 요소의 하위 요소)를 생성하여 name에는 aws 속성을, type에는 Amazon.AWSSection, AWSSDK.Core 속성을 설정해야 합니다.

```
<?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 편집기에서는 <aws> 요소 또는 그 하위 요소에 대해서는 자동 코드 완성(IntelliSense) 기능을 제공하지 않습니다.

<aws> 요소의 올바른 형식 버전을 생성하는 데 도움을 받으려면 Amazon.AWSConfigs.GenerateConfigTemplate 메서드를 호출하십시오. 그렇게 하면 <aws> 요소의 표준 버전이 가독성 좋게 꾸민 문자열로 출력되는데, 이를 필요에 따라 수정할 수 있습니다. 다음 섹션에서는 <aws> 요소의 속성 및 하위 요소에 대해 설명합니다.

허용되는 요소

다음 목록은 AWS 설정 섹션에서 허용되는 요소들 간의 논리적 관계를 나타낸 것입니다. Amazon.AWSConfigs.GenerateConfigTemplate 메서드를 호출하여 이 목록의 최신 버전을 생성할 수 있습니다. 그렇게 하면 <aws> 요소의 표준 버전이 가독성 좋게 꾸민 문자열로 출력되는데, 이를 필요에 따라 수정할 수 있습니다.

```
<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>
```

요소 참조

다음은 AWS 설정 섹션에서 허용되는 요소의 목록입니다. 각 요소에 허용되는 속성 및 상위-하위 요소가 나열되어 있습니다.

주제

- 별칭
- aws
- dynamoDB
- dynamoDBContext
- EC2
- 로그
- map
- property
- proxy
- s3

별칭

<alias> 요소는 한 가지 형식에 맞게 구성된 테이블과는 다른 테이블을 지정하는 한 개 이상의 테이블 매핑(from-table에서 to-table로 매핑) 모음에 있는 단일 항목을 나타냅니다. 이 요소는 AWS SDK for .NET의 Amazon.AWSConfigs.DynamoDBConfig.Context.TableAliases 속성에서 Amazon.Util.TableAlias 클래스의 인스턴스로 매핑됩니다. 테이블 이름 접두사를 적용하기 전에 다시 매핑이 됩니다.

이 요소에는 다음과 같은 속성이 포함될 수 있습니다.

fromTable

from-table에서 to-table로의 매핑에서 from-table 부분. 이 속성은 AWS SDK for .NET의 `Amazon.Util.TableAlias.FromTable` 속성으로 매핑됩니다.

toTable

from-table에서 to-table로의 매핑에서 to-table 부분. 이 속성은 AWS SDK for .NET의 `Amazon.Util.TableAlias.ToTable` 속성으로 매핑됩니다.

`<alias>` 요소의 부모는 `<tableAliases>` 요소입니다.

`<alias>` 요소에는 하위 요소가 포함되어 있지 않습니다.

다음은 사용 중인 `<alias>` 요소의 예입니다.

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

aws

`<aws>` 요소는 AWS 설정 섹션에서 최상위 요소를 나타냅니다. 이 요소에는 다음과 같은 속성이 포함될 수 있습니다.

endpointDefinition

사용할 AWS 리전과 엔드포인트를 정의하는 사용자 지정 구성 파일의 절대 경로입니다. 이 속성은 AWS SDK for .NET의 `Amazon.AWSConfigs.EndpointDefinition` 속성으로 매핑됩니다.

profileName

서비스 호출에 사용할 저장된 AWS 자격 증명의 프로필 이름입니다. 이 속성은 AWS SDK for .NET의 `Amazon.AWSConfigs.AWSProfileName` 속성으로 매핑됩니다.

profilesLocation

다른 AWS SDKs. 기본적으로 자격 증명 파일은 현재 사용자의 홈 디렉터리에 있는 `.aws` 디렉터리에 저장됩니다. 이 속성은 AWS SDK for .NET의 `Amazon.AWSConfigs.AWSProfilesLocation` 속성으로 매핑됩니다.

region

AWS 리전을 명시적으로 지정하지 않은 클라이언트의 기본 리전 ID입니다. 이 속성은 AWS SDK for .NET의 `Amazon.AWSConfigs.AWSRegion` 속성으로 매핑됩니다.

`<aws>` 요소에는 상위 요소가 없습니다.

`<aws>` 요소에는 다음과 같은 하위 요소가 포함될 수 있습니다.

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

다음은 사용 중인 `<aws>` 요소의 예입니다.

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

dynamoDB

`<dynamoDB>` 요소는 Amazon DynamoDB의 설정 모음을 나타냅니다. 이 요소에는 .NET 및 DynamoDB 객체 간 변환에 사용할 버전을 나타내는 conversionSchema 속성이 포함될 수 있습니다. 허용되는 값은 V1, V2 등입니다. 이 속성은 AWS SDK for .NET의 `Amazon.DynamoDBv2.DynamoDBEntryConversion` 클래스로 매핑됩니다. 자세한 내용은 DynamoDB 시리즈 - 변환 스키마를 참조하십시오.

`<dynamoDB>` 요소의 부모는 `<aws>` 요소입니다.

`<dynamoDB>` 요소에는 하위 요소인 `<dynamoDBContext>`가 포함될 수 있습니다.

다음은 사용 중인 `<dynamoDB>` 요소의 예입니다.

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

dynamoDBContext

<dynamoDBContext> 요소는 Amazon DynamoDB의 상황에 맞는 설정 모음을 나타냅니다. 이 요소에는 수동으로 구성하지 않은 경우 DynamoDB 컨텍스트에서 사용할 기본 테이블 이름 접두사를 나타내는 tableNamePrefix 속성이 포함될 수 있습니다. 이 속성은 AWS SDK for .NET의 Amazon.AWSConfigs.DynamoDBConfig.Context.TableNamePrefix 속성에서 Amazon.Util.DynamoDBContextConfig.TableNamePrefix 속성으로 매핑됩니다. 자세한 내용은 DynamoDB SDK의 개선점을 참조하십시오.

<dynamoDBContext> 요소의 부모는 <dynamoDB> 요소입니다.

<dynamoDBContext> 요소에는 다음과 같은 하위 요소가 포함될 수 있습니다.

- <alias>(하나 이상의 인스턴스)
- <map>(하나 이상의 인스턴스)

다음은 사용 중인 <dynamoDBContext> 요소의 예입니다.

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

EC2

<ec2> 요소는 Amazon EC2 설정 모음을 나타냅니다. 이 요소에는 모든 요청에 서명 버전 4 서명을 사용할 것인지(true) 또는 모든 요청에 서명 버전 4 서명을 사용하지 않을 것인지(false, 기본값)를 지정하는 useSignatureVersion4 속성이 포함될 수 있습니다. 이 속성은 AWS SDK for .NET의 Amazon.AWSConfigs.EC2Config.UseSignatureVersion4 속성에서 Amazon.Util.EC2Config.UseSignatureVersion4 속성으로 매핑됩니다.

<ec2> 요소의 부모가 그 요소입니다.

<ec2> 요소에는 하위 요소가 포함되어 있지 않습니다.

다음은 사용 중인 <ec2> 요소의 예입니다.

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

로그

<logging> 요소는 응답 로깅 및 성능 측정치 로깅에 대한 설정 모음을 나타냅니다. 이 요소에는 다음과 같은 속성이 포함될 수 있습니다.

logMetrics

성능 측정치를 모든 클라이언트 및 구성에 대해 로깅(true)할지 여부를 나타냅니다. 로깅하지 않는 경우 false. 이 속성은 AWS SDK for .NET의 Amazon.AWSConfigs.LoggingConfig.LogMetrics 속성에서 Amazon.Util.LoggingConfig.LogMetrics 속성으로 매핑됩니다.

logMetricsCustomFormatter

로깅 측정치를 위한 사용자 지정 포맷터의 데이터 유형 및 어셈블리 이름입니다. 이 속성은 AWS SDK for .NET의 Amazon.AWSConfigs.LoggingConfig.LogMetricsCustomFormatter 속성에서 Amazon.Util.LoggingConfig.LogMetricsCustomFormatter 속성으로 매핑됩니다.

logMetricsFormat

로깅 측정치가 표시되는 형식입니다(AWS SDK for .NET의 Amazon.AWSConfigs.LoggingConfig.LogMetricsFormat 속성에서 Amazon.Util.LoggingConfig.LogMetricsFormat 속성으로 매핑됨).

허용되는 값은 다음과 같습니다.

JSON

JSON 형식을 사용합니다.

Standard

기본 형식을 사용합니다.

logResponses

서비스 응답을 로깅할 시점입니다(AWS SDK for .NET의 Amazon.AWSConfigs.LoggingConfig.LogResponses 속성에서 Amazon.Util.LoggingConfig.LogResponses 속성으로 매핑됨).

허용되는 값은 다음과 같습니다.

Always

서비스 응답을 항상 로깅.

Never

서비스 응답을 로깅하지 않음.

OnError

오류가 있을 때만 서비스 응답을 로깅.

logTo

에 로그인할 위치(의 LogTo 속성에서 Amazon.AWSConfigs.LoggingConfig.LogTo 속성에 매핑 AWS SDK for .NET).

허용되는 값에는 다음 중 한 개 이상이 포함됩니다.

Log4Net

log4net에 로깅.

None

로깅을 비활성화.

SystemDiagnostics

System.Diagnostics에 로깅.

<logging> 요소의 부모는 <aws> 요소입니다.

<logging> 요소에는 하위 요소가 포함되어 있지 않습니다.

다음은 사용 중인 <logging> 요소의 예입니다.

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

map

<map> 요소는 .NET 형식에서 DynamoDB 테이블로 형식-테이블 간 매핑 모음에 있는 단일 항목을 나타냅니다(AWS SDK for .NET의 Amazon.AWSConfigs.DynamoDBConfig.Context.TypeMappings 속성에서 TypeMapping 클래스의 인스턴스로 매핑됨). 자세한 내용은 DynamoDB SDK의 개선점을 참조하십시오.

이 요소에는 다음과 같은 속성이 포함될 수 있습니다.

targetTable

매핑이 적용되는 DynamoDB 테이블입니다. 이 속성은 AWS SDK for .NET의 Amazon.Util.TypeMapping.TargetTable 속성으로 매핑됩니다.

type

매핑이 적용되는 형식 및 어셈블리 이름입니다. 이 속성은 AWS SDK for .NET의 Amazon.Util.TypeMapping.Type 속성으로 매핑됩니다.

<map> 요소의 부모는 <dynamoDBContext> 요소입니다.

<map> 요소에는 하위 요소인 <property>의 인스턴스가 한 개 이상 포함될 수 있습니다.

다음은 사용 중인 <map> 요소의 예입니다.

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

property

<property> 요소는 DynamoDB 속성을 나타냅니다. (이 요소는의 AddProperty 메서드에서 Amazon.Util.PropertyConfig 클래스의 인스턴스에 매핑됩니다 AWS SDK for .NET.) 자세한 내용은 DynamoDB SDK 및 DynamoDB 속성에 대한 개선 사항을 참조하세요 DynamoDB.

이 요소에는 다음과 같은 속성이 포함될 수 있습니다.

attribute

범위 키 이름과 같은 속성에 대한 속성 이름입니다. 이 속성은 AWS SDK for .NET의 Amazon.Util.PropertyConfig.Attribute 속성으로 매핑됩니다.

converter

이 속성에 사용해야 할 변환기의 유형입니다. 이 속성은 AWS SDK for .NET의 `Amazon.Util.PropertyConfig.Converter` 속성으로 매핑됩니다.

ignore

연결된 속성을 무시(true)할지 여부를 나타냅니다. 무시하지 않을 경우 false. 이 속성은 AWS SDK for .NET의 `Amazon.Util.PropertyConfig.Ignore` 속성으로 매핑됩니다.

name

속성의 이름입니다. 이 속성은 AWS SDK for .NET의 `Amazon.Util.PropertyConfig.Name` 속성으로 매핑됩니다.

version

이 속성이 항목 버전 번호를 저장(true)할지 여부를 나타냅니다. 저장하지 않을 경우 false. 이 속성은 AWS SDK for .NET의 `Amazon.Util.PropertyConfig.Version` 속성으로 매핑됩니다.

`<property>` 요소의 부모는 `<map>` 요소입니다.

`<property>` 요소에는 하위 요소가 포함되어 있지 않습니다.

다음은 사용 중인 `<property>` 요소의 예입니다.

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

proxy

`<proxy>` 요소는 AWS SDK for .NET 에서 사용할 프록시를 구성하기 위한 설정을 나타냅니다. 이 요소에는 다음과 같은 속성이 포함될 수 있습니다.

host

프록시 서버의 호스트 이름 또는 IP 주소입니다. 이 속성은 AWS SDK for .NET의 `Amazon.AWSConfigs.ProxyConfig.Host` 속성에서 `Amazon.Util.ProxyConfig.Host` 속성으로 매핑됩니다.

비밀번호

프록시 서버로 인증하는 데 필요한 암호입니다. 이 속성은 AWS SDK for .NET의 `Amazon.AWSConfigs.ProxyConfig.Password` 속성에서 `Amazon.Util.ProxyConfig.Password` 속성으로 매핑됩니다.

포트

프록시의 포트 번호입니다. 이 속성은 AWS SDK for .NET의 `Amazon.AWSConfigs.ProxyConfig.Port` 속성에서 `Amazon.Util.ProxyConfig.Port` 속성으로 매핑됩니다.

사용자 이름

프록시 서버로 인증하는 데 필요한 사용자 이름입니다. 이 속성은 AWS SDK for .NET의 `Amazon.AWSConfigs.ProxyConfig.Username` 속성에서 `Amazon.Util.ProxyConfig.Username` 속성으로 매핑됩니다.

`<proxy>` 요소의 부모는 `<aws>` 요소입니다.

`<proxy>` 요소에는 하위 요소가 포함되어 있지 않습니다.

다음은 사용 중인 `<proxy>` 요소의 예입니다.

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

s3

`<s3>` 요소는 Amazon S3 설정 모음을 나타냅니다. 이 요소에는 모든 요청에 서명 버전 4 서명을 사용할 것인지(true) 또는 모든 요청에 서명 버전 4 서명을 사용하지 않을 것인지(false, 기본값)를 지정하는 useSignatureVersion4 속성이 포함될 수 있습니다. 이 속성은 AWS SDK for .NET의 `Amazon.AWSConfigs.S3Config.UseSignatureVersion4` 속성으로 매핑됩니다.

`<s3>` 요소의 부모는 `<aws>` 요소입니다.

`<s3>` 요소에는 하위 요소가 포함되어 있지 않습니다.

다음은 사용 중인 <s3> 요소의 예입니다.

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

레거시 보안 인증 사용

이 섹션의 주제에서는 AWS IAM Identity Center를 사용하지 않고 장기 또는 단기 보안 인증 정보를 사용하는 방법에 대한 정보를 제공합니다.

> **Warning**
>
> 보안 위험을 방지하려면 목적별 소프트웨어를 개발하거나 실제 데이터로 작업할 때 IAM 사용자를 인증에 사용하지 마세요. 대신 AWS IAM Identity Center과 같은 보안 인증 공급자를 통한 페더레이션을 사용하십시오.

> **Note**
>
> 이 주제의 정보는 단기 또는 장기 보안 인증 정보를 수동으로 획득하고 관리해야 하는 상황을 위한 것입니다. 단기 및 장기 보안 인증 정보에 대한 자세한 내용은 AWS 및 도구 참조 가이드의 다른 인증 방법을 참조하세요.
> 보안 모범 사례를 보려면에 설명된 AWS IAM Identity Center대로를 사용합니다를 사용하여 인증 AWS.

보안 인증에 대한 중요 경고 및 지침

보안 인증에 대한 경고

- 계정의 루트 자격 증명을 사용하여 AWS 리소스에 액세스하지 마십시오. 이 보안 인증은 계정 액세스에 제한이 없고 취소하기 어렵습니다.
- 금지 사항. 애플리케이션 파일에 리터럴 액세스 키나 보안 인증 정보를 넣으면 안 됩니다. 이를 어기는 경우, 예를 들어 프로젝트를 퍼블릭 리포지토리에 업로드하면 뜻하지 않게 보안 인증이 노출될 위험이 있습니다.
- 금지 사항. 프로젝트 영역에 보안 인증이 포함된 파일을 포함하지 마세요.
- 공유 AWS `credentials` 파일에 저장된 모든 자격 증명은 일반 텍스트로 저장됩니다.

보안 인증 정보를 안전하게 관리하기 위한 추가 지침

자격 AWS 증명을 안전하게 관리하는 방법에 대한 일반적인 설명은의 AWS 보안 자격 증명AWS 일반 참조과 IAM 사용 설명서의 보안 모범 사례 및 사용 사례를 참조하세요. 해당 설명과 더불어 다음 사항을 고려하세요.

- AWS 루트 사용자 보안 인증 정보를 사용하는 대신 IAM Identity Center에 사용자 등 추가 사용자를 만들고 해당 보안 인증 정보를 사용합니다. 다른 사용자의 보안 인증 정보는 필요한 경우 또는 일시적인 경우 해지할 수 있습니다. 또한 각 사용자에게 특정 리소스 및 작업에만 액세스할 수 있도록 정책을 적용하여 최소 권한 권한을 유지할 수 있습니다.

- Amazon EC2 Container Service(Amazon ECS) 작업에 작업용 IAM 역할을 사용하세요.

- Amazon EC2 인스턴스에서 실행 중인 애플리케이션에 IAM 역할을 사용하십시오.

- 조직 외부의 사용자들이 쓸 수 있는 애플리케이션에 대해서는 임시 자격 증명 또는 환경 변수를 사용하십시오.

주제

- 공유 AWS 자격 증명 파일 사용
- SDK 스토어 사용(Windows만 해당)

공유 AWS 자격 증명 파일 사용

(보안 인증에 대한 중요 경고 및 지침을 반드시 검토하세요.)

애플리케이션에 보안 인증을 제공하는 한 가지 방법은 공유 AWS 보안 인증 파일에 프로필을 만든 다음 해당 프로필에 보안 인증을 저장하는 것입니다. 이 파일은 다른 AWS SDKs. 또한 Visual StudioAWS CLI, JetBrains AWS Tools for Windows PowerShell및 VS Code용 , 및 AWS 툴킷에서도 사용할 수 있습니다.

> **Warning**
>
> 보안 위험을 방지하려면 목적별 소프트웨어를 개발하거나 실제 데이터로 작업할 때 IAM 사용자를 인증에 사용하지 마세요. 대신 AWS IAM Identity Center과 같은 보안 인증 공급자를 통한 페더레이션을 사용하십시오.

> **Note**
>
> 이 주제의 정보는 단기 또는 장기 보안 인증 정보를 수동으로 획득하고 관리해야 하는 상황을 위한 것입니다. 단기 및 장기 보안 인증 정보에 대한 자세한 내용은 AWS 및 도구 참조 가이드의 다른 인증 방법을 참조하세요.
> 보안 모범 사례를 보려면에 설명된 AWS IAM Identity Center대로를 사용합니다를 사용하여 인증 AWS.

일반 정보

기본적으로 공유 AWS 자격 증명 파일은 홈 `.aws` 디렉터리 내의 디렉터리에 위치하며 이름이 , `credentials`즉 `~/.aws/credentials` (Linux 또는 macOS) 또는 `%USERPROFILE%\.aws\credentials` (Windows)입니다. 대체 위치에 대한 자세한 내용은 AWS SDK 및 도구 참조 가이드의 공유 파일의 위치를 참조하세요. 애플리케이션에서 자격 증명 및 프로필에 액세스 섹션도 참조하세요.

공유 AWS 자격 증명 파일은 일반 텍스트 파일이며 특정 형식을 따릅니다. 자격 AWS 증명 파일 형식에 대한 자세한 내용은 SDK 및 도구 참조 안내서의 자격 증명 파일 형식을 참조하세요. AWS SDKs

여러 가지 방법으로 공유 AWS 자격 증명 파일의 프로파일을 관리할 수 있습니다.

- 텍스트 편집기를 사용하여 공유 AWS 자격 증명 파일을 생성하고 업데이트합니다.
- 이 주제의 뒷부분에서 볼 수 있듯이 AWS SDK for .NET API의 Amazon.Runtime.CredentialManagement 네임스페이스를 사용합니다.
- AWS Tools for PowerShell 및 Visual Studio, JetBrains 및 VS Code용 AWS 툴킷에 명령과 절차를 사용합니다. https://docs.aws.amazon.com/toolkit-for-vscode/latest/userguide/setup-credentials.html
- AWS CLI 명령을 사용합니다(예: `aws configure set aws_access_key_id` 및 `aws configure set aws_secret_access_key`).

프로필 관리의 예

다음 섹션에서는 공유 AWS 자격 증명 파일의 프로필 예를 보여줍니다. 일부 예제는 앞서 설명한 보안 인증 관리 방법 중 하나를 통해 얻을 수 있는 결과를 보여줍니다. 다른 예제는 특정 방법을 사용하는 방법을 보여줍니다.

기본 프로필

공유 AWS 자격 증명 파일에는 거의 항상 이름이 default인 프로필이 있습니다. 여기서는 다른 프로필이 정의되지 않은 경우 자격 증명을 AWS SDK for .NET 찾습니다.

`[default]` 프로필은 일반적으로 다음과 같습니다.

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

프로그래밍 방식으로 프로필 생성

이 예제에서는 프로파일을 생성하고 프로그래밍 방식으로 공유 AWS 자격 증명 파일에 저장하는 방법을 보여줍니다. CredentialProfileOptions, CredentialProfile 및 SharedCredentialsFile과 같은 Amazon.Runtime.CredentialManagement 네임스페이스의 클래스를 사용합니다.

```
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**
>
> 이러한 코드는 일반적으로 애플리케이션에 없어야 합니다. 애플리케이션에 포함하는 경우 코드, 네트워크 또는 컴퓨터 메모리에서도 일반 텍스트 키가 보이지 않도록 적절한 예방 조치를 취하십시오.

다음은 이 예제에서 생성된 프로필입니다.

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

프로그래밍 방식으로 기존 프로필 업데이트

이 예제에서는 이전에 만든 프로필을 프로그래밍 방식으로 업데이트하는 방법을 보여줍니다. CredentialProfile 및 NetSDKCredentialsFile과 같은 Amazon.Runtime.CredentialManagement 네임스페이스의 클래스를 사용합니다. 또한 Amazon 네임스페이스의 RegionEndpoint 클래스를 사용합니다.

```
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);
    }
}
```

다음은 업데이트된 프로필입니다.

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

```
region=us-west-2
```

> **Note**
>
> 다른 위치에서 다른 방법을 사용하여 AWS 리전을 설정할 수도 있습니다. 자세한 내용은 에 대한 AWS 리전 설정 AWS SDK for .NET 단원을 참조하십시오.

SDK 스토어 사용(Windows만 해당)

(중요 경고 및 지침을 반드시 검토하세요.)

Windows에서 SDK 스토어는 프로필을 생성하고 AWS SDK for .NET 애플리케이션의 암호화된 자격 증명을 저장하는 또 다른 곳입니다. `%USERPROFILE%\AppData\Local\AWSToolkit\RegisteredAccounts.json`에 있습니다. 개발 중에 공유 AWS 보안 인증 파일 대신 SDK 스토어를 사용할 수 있습니다.

> **Warning**
>
> 보안 위험을 방지하려면 목적별 소프트웨어를 개발하거나 실제 데이터로 작업할 때 IAM 사용자를 인증에 사용하지 마세요. 대신 AWS IAM Identity Center과 같은 보안 인증 공급자를 통한 페더레이션을 사용하십시오.

> **Note**
>
> 이 주제의 정보는 단기 또는 장기 보안 인증 정보를 수동으로 획득하고 관리해야 하는 상황을 위한 것입니다. 단기 및 장기 보안 인증 정보에 대한 자세한 내용은 AWS 및 도구 참조 가이드의 다른 인증 방법을 참조하세요.
> 보안 모범 사례를 보려면에 설명된 AWS IAM Identity Center대로를 사용합니다를 사용하여 인증 AWS.

일반 정보

SDK 스토어에는 다음과 같은 이점이 있습니다.

- SDK 스토어의 보안 인증은 암호화되며 SDK 스토어는 사용자의 홈 디렉터리에 상주합니다. 이렇게 하면 예기치 않게 자격 증명이 노출될 위험을 줄일 수 있습니다.

- SDK 스토어도 AWS Tools for Windows PowerShell 및 AWS Toolkit for Visual Studio에 보안 인증을 제공합니다.

SDK 스토어 프로필은 특정 호스트에 있는 특정 사용자에게 고유하며, 다른 호스트나 다른 사용자에게 복사할 수 없습니다. 즉, 다른 호스트 또는 개발자 머신의 개발 머신에 있는 SDK 스토어 프로필을 재사용할 수 없습니다. 또한 프로덕션 애플리케이션에서는 SDK 스토어 프로필을 사용할 수 없습니다.

다음과 같은 방식으로 SDK 스토어에서 프로필을 관리할 수 있습니다.

- AWS Toolkit for Visual Studio에서 그래픽 사용자 인터페이스(GUI)를 사용합니다.
- 이 주제의 뒷부분에서 볼 수 있듯이 AWS SDK for .NET API의 Amazon.Runtime.CredentialManagement 네임스페이스를 사용합니다.
- AWS Tools for Windows PowerShell의 명령을 사용합니다(예: `Set-AWSCredential` 및 `Remove-AWSCredentialProfile`).

프로필 관리의 예

다음 예제에서는 SDK 스토어에서 프로그래밍 방식으로 프로필을 생성하고 업데이트하는 방법을 보여줍니다.

프로그래밍 방식으로 프로필 생성

이 예제에서는 프로그래밍 방식으로 프로필을 만들고 SDK 스토어에 저장하는 방법을 보여줍니다. CredentialProfileOptions, CredentialProfile 및 NetSDKCredentialsFile과 같은 Amazon.Runtime.CredentialManagement 네임스페이스의 클래스를 사용합니다.

```
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**
>
> 이러한 코드는 일반적으로 애플리케이션에 없어야 합니다. 애플리케이션에 포함되어 있는 경우 코드, 네트워크 또는 컴퓨터 메모리에서도 일반 텍스트 키가 보이지 않도록 적절한 예방 조치를 취하십시오.

다음은 이 예제에서 생성된 프로필입니다.

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

프로그래밍 방식으로 기존 프로필 업데이트

이 예제에서는 이전에 만든 프로필을 프로그래밍 방식으로 업데이트하는 방법을 보여줍니다. CredentialProfile 및 NetSDKCredentialsFile과 같은 Amazon.Runtime.CredentialManagement 네임스페이스의 클래스를 사용합니다. 또한 Amazon 네임스페이스의 RegionEndpoint 클래스를 사용합니다.

```
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);
    }
}
```

다음은 업데이트된 프로필입니다.

```
"[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**
>
> 다른 위치에서 다른 방법을 사용하여 AWS 리전을 설정할 수도 있습니다. 자세한 내용은 에 대한 AWS 리전 설정 AWS SDK for .NET 단원을 참조하십시오.

사용 AWS SDK for .NET

이 섹션에서는 애플리케이션을 생성할 때 고려해야 AWS SDK for .NET 할의 기능에 대한 정보를 제공합니다.

시작하기 전에 환경을 설정하고 프로젝트를 구성했는지 확인합니다.

코드 예제와 함께 특정 AWS 서비스를 위한 소프트웨어 개발에 대한 자세한 내용은 섹션을 참조하세요 AWS 서비스 호출. 추가 코드 예제는 SDK for .NET (v4) 코드 예제 단원을 참조하세요.

주제

- 를 사용하여 비동기식 프로그래밍 AWS SDK for .NET
- 에서 페이지 매김된 결과 사용 AWS SDK for .NET
- 에서 HTTP 2 지원 AWS SDK for .NET
- 추가 도구

를 사용하여 비동기식 프로그래밍 AWS SDK for .NET

는 비동기 구현에 작업 기반 비동기 패턴(TAP)을 AWS SDK for .NET 사용합니다. TAP에 대한 자세한 내용은 docs.microsoft.com의 작업 기반 비동기식 패턴(TAP)을 참조하세요.

이 주제에서는 AWS 서비스 클라이언트 호출에서 TAP을 사용하는 방법에 대한 개요를 제공합니다.

AWS SDK for .NET API의 비동기식 메서드는 `Task` 클래스 또는 `Task<TResult>` 클래스를 기반으로 하는 작업입니다. Task 클래스, Task<TResult> 클래스와 같은 클래스에 대한 자세한 내용은 docs.microsoft.com을 참조하세요.

코드에서 이러한 API 메서드를 호출할 때는 다음 예제와 같이 `async` 키워드로 선언된 함수 내에서 호출해야 합니다.

```
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;
}
```

위 코드 조각에서 볼 수 있듯이 async 선언 시 선호되는 범위는 Main 함수입니다. 이 async 범위를 설정하면 AWS 서비스 클라이언트에 대한 모든 호출이 비동기식이어야 합니다. 어떤 이유로 Main을 비동기식으로 선언할 수 없는 경우 다음 예제와 같이 async 키워드를 Main 이외의 함수에서 사용한 다음 해당 함수에서 API 메서드를 호출할 수 있습니다.

```
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;
}
```

이 패턴을 사용할 때 Main에서 필요한 특수 Task<> 구문에 유의해야 합니다. 또한 데이터를 가져오려면 응답의 **Result** 멤버를 사용해야 합니다.

간단한 애플리케이션 생성 섹션(간단한 교차 플랫폼 앱 및 간단한 Windows 기반 앱) 및에서 AWS 서비스 클라이언트에 대한 비동기 호출의 전체 예를 볼 수 있습니다가이드 코드 예제.

에서 페이지 매김된 결과 사용 AWS SDK for .NET

일부 AWS 서비스는의 API 호출을 사용하여 검색할 수 있는 대량의 데이터를 수집하고 저장합니다 AWS SDK for .NET. 검색하려는 데이터 양이 단일 API 직접 호출로 너무 많아지면 페이지 매김을 사용하여 결과를 관리하기 쉬운 부분으로 나눌 수 있습니다.

페이지 매김을 수행할 수 있도록 SDK의 여러 서비스 클라이언트에 대한 요청 및 응답 객체는 연속 토큰(일반적으로 `NextToken`으로 이름이 지정됨)을 제공합니다. 이러한 서비스 클라이언트 중 일부는 페이지네이터도 제공합니다.

페이지네이터를 사용하면 루프, 상태 변수, 다중 API 직접 호출 등이 포함될 수 있는 연속 토큰의 오버헤드를 피할 수 있습니다. 페이지네이터를 사용하면 한 줄의 코드인 `foreach` 루프 선언을 통해 AWS 서비스에서 데이터를 검색할 수 있습니다. 데이터를 검색하는 데 여러 API 직접 호출이 필요한 경우 페이지네이터가 이를 자동으로 처리합니다.

페이지네이터는 어디서 찾을 수 있나요?

모든 서비스가 페이지네이터를 제공하는 것은 아닙니다. 서비스가 특정 API에 페이지네이터를 제공하는지 여부를 확인하는 한 가지 방법은 AWS SDK for .NET API 참조에서 서비스 클라이언트 클래스의 정의를 살펴보는 것입니다.

예를 들어 AmazonCloudWatchLogsClient 클래스의 정의를 검토하면 `Paginators` 속성이 표시됩니다. 이는 Amazon CloudWatch Logs의 페이지네이터를 제공하는 속성입니다.

페이지네이터는 무엇을 제공합니까?

페이지네이터에는 전체 응답을 볼 수 있는 속성이 있습니다. 또한 일반적으로 응답의 가장 흥미로운 부분에 액세스할 수 있는 하나 이상의 속성이 포함되어 있는데, 이를 핵심 결과라고 합니다.

예를 들어 앞서 언급한 `AmazonCloudWatchLogsClient`에서 `Paginator` 객체에는 API 직접 호출에서 가져온 전체 DescribeLogGroupsResponse 객체가 포함된 `Responses` 속성이 포함되어 있습니다. 이 `Responses` 속성에는 무엇보다도 로그 그룹 컬렉션이 포함되어 있습니다.

페이지네이터 객체에는 `LogGroups`로 이름이 지정된 하나의 주요 결과도 포함되어 있습니다. 이 속성은 응답의 로그 그룹 부분만 포함합니다. 이 주요 결과를 통해 많은 상황에서 코드를 줄이고 단순화할 수 있습니다.

동기 페이지 매김과 비동기 페이지 매김 비교

페이지네이터는 동기 및 비동기 페이지 매김 메커니즘을 모두 제공합니다. 동기 페이지 매김은 .NET Framework 4.7.2(또는 이후 버전) 프로젝트에서 사용할 수 있습니다. 비동기 페이지 매김은 .NET Core 프로젝트(.NET Core 3.1, .NET 5 등)에서 사용할 수 있습니다.

비동기 작업과 .NET Core가 권장되므로 다음 예제에서는 비동기 페이지 매김을 보여줍니다. 동기 페이지 매김 및 .NET Framework 4.7.2(또는 이후 버전)를 사용하여 동일한 작업을 수행하는 방법에 대한 정보는의 예제 다음에 나와 있습니다페이지네이터에 관한 추가 고려 사항.

예제

다음 예제에서는 AWS SDK for .NET 를 사용하여 로그 그룹 목록을 표시하는 방법을 보여줍니다. 반대로 이 예에서는 페이지 매김을 사용하거나 사용하지 않고 이 작업을 수행하는 방법을 보여줍니다. 나중에 보여드릴 전체 코드를 살펴보기 전에 다음 스니펫을 살펴보세요.

페이지네이터가 없는 CloudWatch 로그 그룹 가져오기

```
      // 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));
```

페이지네이터를 사용한 CloudWatch 로그 그룹 가져오기

```
      // 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);
      }
```

이 두 스니펫의 결과는 완전히 동일하므로 페이지네이터를 사용할 때의 이점을 분명히 확인할 수 있습니다.

> **Note**
>
> 전체 코드를 빌드하고 실행하기 전에 환경과 프로젝트를 설정했는지 확인하세요.
> 비동기 페이지네이터가 `IAsyncEnumerable` 인터페이스를 사용하기 때문에 Microsoft.Bcl.AsyncInterfaces NuGet 패키지가 필요할 수도 있습니다.

전체 코드

이 섹션에는 이 예제에 대한 관련 참조와 전체 코드가 나와 있습니다.

SDK 레퍼런스

NuGet 패키지:

- AWSSDK.CloudWatch

프로그래밍 요소:

- 네임스페이스 Amazon.CloudWatch

  클래스 AmazonCloudWatchLogsClient
- 네임스페이스 Amazon.CloudWatchLogs.Model

  클래스 DescribeLogGroupsRequest

  클래스 DescribeLogGroupsResponse

  클래스 LogGroup

전체 코드

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

페이지네이터에 관한 추가 고려 사항

- 페이지네이터는 두 번 이상 사용할 수 없음

  코드의 여러 위치에 있는 특정 AWS 페이지네이터의 결과가 필요한 경우 페이지네이터 객체를 두 번 이상 사용해서는 안 됩니다. 대신 필요할 때마다 새 페이지네이터를 만드세요. 이 개념은 `DisplayLogGroupsWithPaginators` 메서드의 이전 예제 코드에 나와 있습니다.

- 동기식 페이지 매김

  동기식 페이지 매김은 .NET Framework 4.7.2(또는 이후 버전) 프로젝트에 사용할 수 있습니다.

  이를 확인하려면 .NET Framework 4.7.2(또는 이후 버전) 프로젝트를 생성하고 이전 코드를 복사합니다. 다음 예제에 표시된 대로 두 개의 `foreach` 페이지네이터 호출에서 `await` 키워드를 제거하기만 하면 됩니다.

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

  프로젝트를 빌드하고 실행하면 비동기 페이지 매김으로 본 것과 동일한 결과를 확인할 수 있습니다.

에서 HTTP 2 지원 AWS SDK for .NET

일부 AWS 서비스 및 작업에는 HTTP 2가 필요합니다. 예를 들어 Amazon Transcribe Streaming의 양방향 스트리밍은 HTTP 1.1을 통해 불가능하므로 대신 HTTP 2가 필요합니다. 애플리케이션에서 이러한 작업을 사용할 수 있도록 HTTP 2에 대한 지원이 버전 4에 AWS SDK for .NET 추가되었습니다. 양방향 HTTP 2 작업의 경우 스트림을 수신할 때 SDK의 동작은 HTTP 1.1의 동작과 유사합니다. 즉, SDK를 사용하는 애플리케이션이 서비스에 이벤트를 전송할 때 요청에는 개발자가 할당한 게시자가 있습니다.

이 동작을 실제로 보려면 Amazon Transcribe Streaming에 대한 다음 예제를 고려하세요. Amazon.TranscribeStreaming 및 Amazon.TranscribeStreaming.Model 네임스페이스를 사용합니다.

이 예제에서 개발자는 .NET 인 콜백 함수를 사용하여 `StartStreamTranscriptionRequest.AudioStreamPublisher` 속성을 정의합니다Func. SDK

는 AudioStreamPublisher에 Func 정의된를 사용하여 사용자 코드에서 이벤트를 가져와 사용자에게 스트리밍합니다. SDK는 null을 반환할 Func 때까지를 호출합니다.

이 코드는 파일의 오디오를 처리를 위해 Amazon Transcribe Streaming으로 스트리밍하는 방법을 보여줍니다.

```
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

Amazon Bedrock의 `InvokeModelWithBidirectionalStreamAsync` 메서드, Amazon.BedrockRuntime 네임스페이스와 같은 일부 양방향 HTTP 2 작업은 일부 이벤트가 게시될 때까지 서비스 클라이언트의 초기 호출에서 응답을 반환하지 않습니다. 이 동작으로 인해 애플리케이션이 차단될 수 있습니다. 이러한 상황을 방지하려면 게시자에게 이벤트를 제공하

는 애플리케이션 코드를 분리하고 서비스 클라이언트에서 작업을 호출하는 스레드와 다른 스레드에서 실행합니다.

추가 고려 사항

- AWS SDK for .NET HTTP 2에 대한 지원은 .NET 8 이상을 대상으로 하는 버전에서만 사용할 수 있습니다. .NET Framework를 대상으로 하는 버전에서는 사용할 수 없습니다.
- 자세한 내용은 aws-sdk-net GitHub 리포지토리의 PR 3730을 참조하세요.

추가 도구

다음은 .NET 애플리케이션을 쉽게 개발, 배포 및 유지 관리하는 데 사용할 수 있는 몇 가지 추가 도구입니다.

AWS 배포 도구

개발 머신에서 클라우드 네이티브 .NET Core 애플리케이션을 개발한 후에는 .NET CLI용 AWS 배포 도구를 사용하여 애플리케이션을에 보다 쉽게 배포할 수 있습니다 AWS.

자세한 내용은 에 애플리케이션 배포 AWS 단원을 참조하십시오.

AWS .NET용 메시지 처리 프레임워크

Amazon SQS, Amazon SNS 또는 Amazon EventBridge와 같은 서비스를 사용하는 경우 .NET용 AWS 메시지 처리 프레임워크를 활용할 수 있습니다. 자세한 내용은 AWS .NET용 메시지 처리 프레임워크 단원을 참조하십시오.

.NET Aspire와의 통합

.NET Aspire와의 통합을 활용하여 내부 개발 루프를 개선할 수 있습니다. 자세한 내용은 에서 .NET Aspire AWS 와 통합 AWS SDK for .NET 단원을 참조하십시오.

를 사용한 고급 인증 및 권한 부여 AWS SDK for .NET

이 섹션의 주제에서는 AWS SDK for .NET 애플리케이션의 인증 및 권한 부여를 위한 고급 기술에 대한 정보를 제공합니다.

주제

- 를 사용한 Single Sign-On AWS SDK for .NET

를 사용한 Single Sign-On AWS SDK for .NET

AWS IAM Identity Center 는 모든 AWS 계정 및 클라우드 애플리케이션에 대한 SSO 액세스를 중앙에서 쉽게 관리할 수 있는 클라우드 기반 SSO(Single Sign-On) 서비스입니다. 자세한 내용은 IAM Identity Center 사용 설명서를 참조하세요.

SDK가 IAM Identity Center와 상호 작용하는 방식에 익숙하지 않은 경우 다음 정보를 참조하십시오.

전반적인 상호 작용 패턴

상위 수준에서 SDK는 다음 패턴과 유사한 방식으로 IAM Identity Center와 상호 작용합니다.

1. IAM Identity Center는 일반적으로 IAM Identity Center 콘솔을 통해 구성되며 SSO 사용자가 참여하도록 초대됩니다.
2. 사용자 컴퓨터의 공유 AWS `config` 파일이 SSO 정보로 업데이트됩니다.
3. 사용자는 IAM Identity Center를 통해 로그인하고 자신에 대해 구성된 AWS Identity and Access Management (IAM) 권한에 대한 단기 자격 증명을 받습니다. 이 로그인은와 같은 비 SDK 도구를 통해 시작 AWS CLI하거나 .NET 애플리케이션을 통해 프로그래밍 방식으로 시작할 수 있습니다.
4. 사용자는 작업을 계속합니다. SSO를 사용하는 다른 애플리케이션을 실행하는 경우 애플리케이션을 열기 위해 다시 로그인할 필요가 없습니다.

이 주제의 나머지 부분에서는 AWS IAM Identity Center설정 및 사용에 대한 참조 정보를 제공합니다. 를 사용하여 인증 AWS의 기본 SSO 설정보다 더 많은 추가 및 고급 정보를 제공합니다. SSO를 처음 사용하는 경우 먼저 해당 주제를 살펴보고 기본 정보를 확인한 다음 다음 자습서에서 SSO가 작동하는지 확인할 AWS수 있습니다.

- 자습서: .NET 애플리케이션 전용
- 자습서: AWS CLI 및 .NET 애플리케이션

이 주제는 다음 섹션을 포함하고 있습니다.

- 사전 조건
- SSO 프로필 설정
- SSO 토큰 생성 및 사용
- 추가 리소스
- 자습서

사전 조건

IAM Identity Center를 사용하기 전에 자격 증명 소스 선택 및 관련 AWS 계정 및 애플리케이션 구성과 같은 특정 작업을 수행해야 합니다. 추가 정보는 다음을 참조하세요.

- 이러한 작업에 관한 일반적인 내용은 IAM Identity Center 사용 설명서의 시작하기를 참조하세요.
- 특정 작업 예제에 대해서는 이 주제의 끝 부분에 있는 자습서 목록을 참조하세요. 하지만 자습서를 사용하기 전에 이 주제의 정보를 검토해야 합니다.

SSO 프로필 설정

관련에서 IAM Identity Center를 구성한 후 SSO에 대한 AWS 계정명명된 프로필을 사용자의 공유 AWS `config` 파일에 추가해야 합니다. 이 프로필은 AWS 액세스 포털에 연결하는 데 사용되며, 이는 사용자에 대해 구성된 IAM 권한에 대한 단기 보안 인증을 반환합니다.

공유 `config` 파일은 일반적으로 Windows에서 `%USERPROFILE%\.aws\config`, Linux 및 macOS에서 `~/.aws/config`로 이름이 지정됩니다. 선호하는 텍스트 편집기를 사용하여 SSO용 새 프로필을 추가할 수 있습니다. 또는 `aws configure sso` 명령을 사용할 수 있습니다. 이 명령에 대한 자세한 내용은 AWS Command Line Interface 사용 설명서의 IAM Identity Center 사용을 위한 AWS CLI 구성을 참조하세요.

새 프로필은 다음과 유사합니다.

```
[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
```

새 프로필의 설정은 아래에 정의되어 있습니다. 처음 두 설정은 AWS 액세스 포털을 정의합니다. 나머지 두 설정은 한 쌍으로 합쳐서 사용자에 대해 구성된 권한을 정의합니다. 모든 4가지 설정은 필수입니다.

`sso_start_url`

조직의 AWS 액세스 포털을 가리키는 URL입니다. 이 값을 찾으려면 IAM Identity Center 콘솔을 열고 설정을 선택한 다음 포털 URL을 찾으십시오.

`sso_region`

액세스 포털 호스트가 AWS 리전 포함된 입니다. 이것은 IAM Identity Center를 활성화할 때 선택한 리전입니다. 다른 작업에 사용하는 리전과 다를 수도 있습니다.

AWS 리전 및 해당 코드의 전체 목록은의 리전 엔드포인트를 참조하세요Amazon Web Services 일반 참조.

`sso_account_id`

AWS Organizations 서비스를 통해 AWS 계정 추가된의 ID입니다. 사용 가능한 계정 목록을 보려면 IAM Identity Center 콘솔로 이동하여 AWS 계정 페이지를 여십시오. 이 설정에 대해 선택하는 계정 ID는 `sso_role_name` 설정에 부여하려는 값(다음에 표시됨)과 일치합니다.

`sso_role_name`

IAM Identity Center 권한 세트의 이름입니다. 이 권한 세트는 IAM Identity Center를 통해 사용자에게 부여되는 권한을 정의합니다.

다음 절차는 이 설정의 값을 찾는 한 가지 방법입니다.

1. IAM Identity Center 콘솔로 이동하여 AWS 계정 페이지를 여십시오.
2. 세부 정보를 표시할 계정을 선택합니다. 선택한 계정은 SSO 권한을 부여하려는 SSO 사용자 또는 그룹이 포함된 계정입니다.
3. 계정에 할당된 사용자 및 그룹 목록을 살펴보고 관심 있는 사용자 또는 그룹을 찾으십시오. `sso_role_name` 설정에서 지정하는 권한 세트는 이 사용자 또는 그룹과 관련된 세트 중 하나입니다.

이 설정에 값을 지정할 때는 Amazon 리소스 이름(ARN)이 아닌 권한 세트의 이름을 사용합니다.

권한 세트에는 IAM 정책 및 사용자 지정 권한 정책이 연결되어 있습니다. 자세한 내용은 IAM Identity Center 사용 설명서의 권한 세트를 참조하세요.

SSO 토큰 생성 및 사용

SSO를 사용하려면 사용자가 먼저 임시 토큰을 생성한 다음 해당 토큰을 사용하여 적절한 AWS 애플리케이션 및 리소스에 액세스해야 합니다. .NET 응용 프로그램의 경우 다음 방법을 사용하여 이러한 임시 토큰을 생성하고 사용할 수 있습니다.

- 필요한 경우 먼저 토큰을 생성한 다음 토큰을 사용하는 .NET 애플리케이션을 생성합니다.
- 를 사용하여 토큰을 생성한 AWS CLI 다음 .NET 애플리케이션에서 토큰을 사용합니다.

이러한 방법은 다음 섹션에 설명되어 있으며 자습서에서도 볼 수 있습니다.

> **Important**
>
> SSO 확인이 작동하려면 애플리케이션에서 다음 NuGet 패키지를 참조해야 합니다.
>
> - `AWSSDK.SSO`
> - `AWSSDK.SSOOIDC`
>
> 이러한 패키지를 참조하지 않으면 런타임 예외가 발생합니다.

.NET 애플리케이션만 해당

이 섹션에서는 필요한 경우 임시 SSO 토큰을 생성한 다음 해당 토큰을 사용하여 .NET 애플리케이션을 만드는 방법을 보여줍니다. 이 프로세스에 대한 전체 자습서는 .NET 애플리케이션만 사용하는 SSO 자습서을 참조하세요.

프로그래밍 방식으로 SSO 토큰 생성 및 사용

를 사용하는 것 외에도 프로그래밍 방식으로 SSO 토큰을 생성할 수도 AWS CLI있습니다.

이를 위해 애플리케이션은 SSO 프로필용 `AWSCredentials` 객체를 생성합니다. 이 객체는 임시 보안 인증이 있는 경우 이를 로드합니다. 그런 다음 애플리케이션은 `AWSCredentials` 객체를 `SSOAWSCredentials` 객체로 캐스팅하고, 필요한 경우 사용자에게 로그인 정보를 요청하는 데 사용되는 콜백 메서드를 비롯한 일부 옵션 속성을 설정해야 합니다.

이 메서드는 다음 코드 조각에 나와 있습니다.

Important

SSO 확인이 작동하려면 애플리케이션에서 다음 NuGet 패키지를 참조해야 합니다.

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

이러한 패키지를 참조하지 않으면 런타임 예외가 발생합니다.

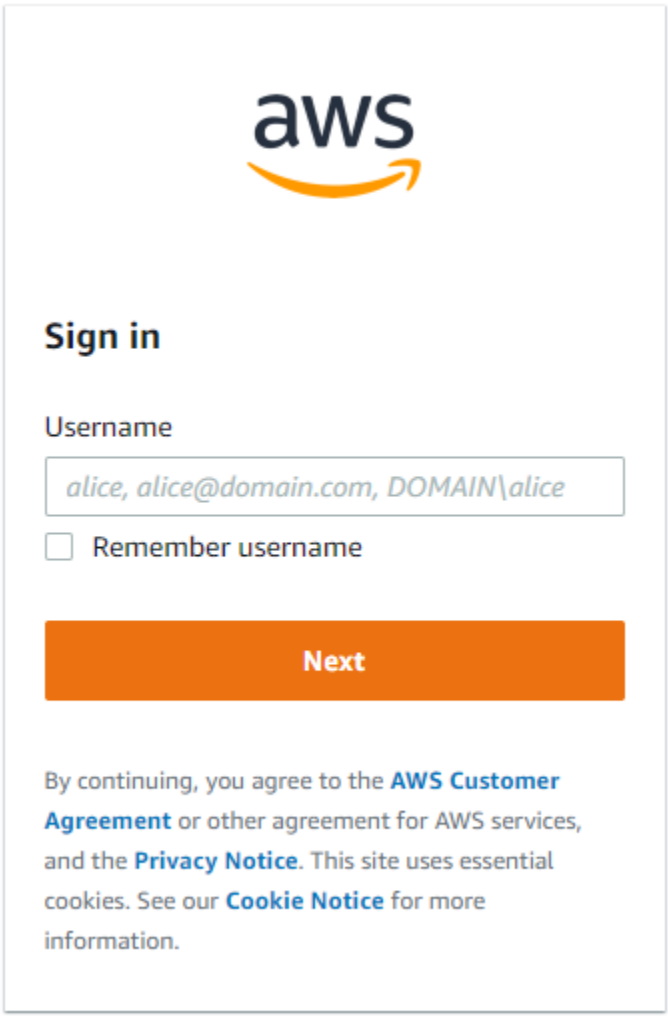
```
static AWSCredentials LoadSsoCredentials()
{
    var chain = new CredentialProfileStoreChain();
    if (!chain.TryGetAWSCredentials("my-sso-profile", out var credentials))
        throw new Exception("Failed to find the my-sso-profile profile");

    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;
}
```

적절한 SSO 토큰을 사용할 수 없는 경우 기본 브라우저 창이 시작되고 적절한 로그인 페이지가 열립니다. 예를 들어 IAM Identity Center를 ID 소스로 사용하는 경우 사용자에게 다음과 비슷한 로그인 페이지가 표시됩니다.



> **Note**
>
> `SSOAWSCredentials.Options.ClientName`에 대해 제공하는 텍스트 문자열에는 공백이 있어서는 안 됩니다. 문자열에 공백이 있는 경우 런타임 예외가 발생합니다.

[.NET 애플리케이션만 사용하는 SSO 자습서]

AWS CLI 및 .NET 애플리케이션

이 섹션에서는를 사용하여 임시 SSO 토큰을 생성하는 방법과 애플리케이션에서 해당 토큰을 사용하는 AWS CLI방법을 보여줍니다. 이 프로세스에 대한 전체 자습서는 AWS CLI 및 .NET 애플리케이션을 사용한 SSO 자습서을 참조하세요.

를 사용하여 SSO 토큰 생성 AWS CLI

프로그래밍 방식으로 임시 SSO 토큰을 생성하는 것 외에도 AWS CLI 를 사용하여 토큰을 생성합니다. 다음 정보에서 이 방법을 보여 줍니다.

사용자는 이전 섹션에 표시된 것처럼 SSO 지원 프로필을 만든 후 AWS CLI에서 `aws sso login` 명령을 실행합니다. SSO 지원 프로필의 이름과 함께 `--profile` 파라미터를 포함해야 합니다. 방법은 다음 예제와 같습니다:

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

현재 토큰이 만료된 후 새 임시 토큰을 생성하려는 경우 동일한 명령을 다시 실행할 수 있습니다.

생성된 SSO 토큰을 .NET 애플리케이션에서 사용하세요.

다음 정보는 이미 생성된 임시 토큰을 사용하는 방법을 보여줍니다.

> **Important**
>
> SSO 확인이 작동하려면 애플리케이션에서 다음 NuGet 패키지를 참조해야 합니다.
>
> - `AWSSDK.SSO`
> - `AWSSDK.SSOOIDC`
>
> 이러한 패키지를 참조하지 않으면 런타임 예외가 발생합니다.

애플리케이션은 SSO 프로필에 대한 `AWSCredentials` 객체를 생성하며, 이 객체는 AWS CLI에서 이전에 생성한 임시 보안 인증을 로드합니다. 이는 애플리케이션에서 자격 증명 및 프로필에 액세스에 표시된 방법과 유사하며 형식은 다음과 같습니다.

```
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;
}
```

그러면 `AWSCredentials` 객체가 서비스 클라이언트의 생성자에 전달됩니다. 예시:

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

> **Note**
>
> 애플리케이션이 SSO용 `[default]` 프로필을 사용하도록 빌드된 경우에는 임시 보안 인증을 로드하는 데 `AWSCredentials`을 사용할 필요가 없습니다. 이 경우 애플리케이션은 "`var client = new AmazonS3Client();`"와 유사한 파라미터 없이 AWS 서비스 클라이언트를 생성할 수 있습니다.

AWS CLI 및 .NET 애플리케이션을 사용한 SSO 자습서

추가 리소스

추가 지원이 필요한 경우 다음 리소스를 참조하세요.

- IAM Identity Center란 무엇입니까?
- IAM Identity Center AWS CLI 를 사용하도록 구성
- 에서 IAM Identity Center 자격 증명 사용 AWS Toolkit for Visual Studio

자습서

주제

- .NET 애플리케이션만 사용하는 SSO 자습서
- AWS CLI 및 .NET 애플리케이션을 사용한 SSO 자습서

.NET 애플리케이션만 사용하는 SSO 자습서

이 자습서에서는 기본 애플리케이션 및 테스트 SSO 사용자를 위해 SSO를 활성화하는 방법을 보여줍니다. AWS CLI를 사용하는 대신 프로그래밍 방식으로 임시 SSO 토큰을 생성하도록 애플리케이션을 구성합니다.

이 자습서에서는에서 SSO 기능의 일부를 보여줍니다 AWS SDK for .NET. 에서 IAM Identity Center를 사용하는 방법에 대한 자세한 내용은 배경 정보가 포함된 주제를 AWS SDK for .NET참조하세요. 해당 항목에서는 특히 .NET 애플리케이션만 해당이라는 하위 섹션의 이 시나리오에 대한 고급 설명을 참조하세요.

> **Note**
>
> 이 자습서의 몇 가지 단계는 AWS Organizations 및 IAM Identity Center와 같은 서비스를 구성하는 데 도움이 됩니다. 해당 구성을 이미 수행했거나 코드에만 관심이 있다면 예제 코드가 있는 섹션으로 건너뛰어도 됩니다.

사전 조건

- 아직 구성하지 않은 경우 개발 환경을 구성합니다. 이에 대해서는 용 도구 체인 설치 및 구성 AWS SDK for .NET 및 시작와 같은 섹션에 설명되어 있습니다.
- SSO를 테스트하는 데 사용할 수 AWS 계정 있는 항목을 하나 이상 식별하거나 생성합니다. 이 자습서에서는 이를 테스트 AWS 계정 또는 간단히 테스트 계정이라고 합니다.
- 대신 SSO를 테스트할 수 있는 SSO 사용자를 식별합니다. 이 사람은 사용자가 생성한 기본 애플리케이션과 SSO를 사용할 사람입니다. 이 자습서에서 그 사람은 사용자(개발자)일 수도 있고 다른 사람일 수도 있습니다. 또한 SSO 사용자가 개발 환경에 있지 않은 컴퓨터에서 작업하도록 설정하는 것이 좋습니다. 그러나 이 작업이 꼭 필요한 것은 아닙니다.
- SSO 사용자 컴퓨터에는 개발 환경을 설정하는 데 사용한 프레임워크와 호환되는 .NET 프레임워크가 설치되어 있어야 합니다.

설정 AWS

이 섹션에서는이 자습서를 위해 다양한 AWS 서비스를 설정하는 방법을 보여줍니다.

이 설정을 수행하려면 먼저 관리자 AWS 계정 로 테스트에 로그인합니다. 뒤이어 다음과 같이 하세요.

Amazon S3

Amazon S3 콘솔로 이동하여 무해한 버킷을 추가하십시오. 이 자습서의 뒷부분에서 SSO 사용자는 이러한 버킷의 목록을 검색하게 됩니다.

AWS IAM

IAM 콘솔로 이동하여 IAM 사용자 몇 명을 추가합니다. IAM 사용자에게 권한을 부여하는 경우 권한을 몇 가지 무해한 읽기 전용 권한으로 제한하십시오. 이 자습서의 뒷부분에서 SSO 사용자는 이러한 IAM 사용자의 목록을 검색하게 됩니다.

AWS Organizations

AWS Organizations 콘솔로 이동하여 조직을 활성화합니다. 자세한 내용은 AWS Organizations 사용 설명서의 조직 생성을 참조하세요.

이 작업은 AWS 계정 조직에 테스트를 관리 계정으로 추가합니다. 추가 테스트 계정이 있는 경우 해당 계정을 조직에 가입하도록 초대할 수 있지만 이 자습서에서는 그렇게 하지 않아도 됩니다.

IAM Identity Center

IAM ID Center 콘솔로 이동하여 SSO를 활성화하십시오. 필요한 경우 이메일 확인을 수행합니다. 자세한 내용은 IAM Identity Center 사용 설명서의 IAM Identity Center 활성화를 참조하세요.

그런 다음 다음과 같이 구성합니다.

IAM Identity Center 구성

1. 설정 페이지로 이동합니다. "액세스 포털 URL"을 찾아 나중에 사용할 수 있도록 `sso_start_url` 설정에 값을 기록해 두세요.
2. 의 배너에서 SSO를 활성화할 때 설정된 AWS 리전 를 AWS Management Console찾습니다. ID 왼쪽의 드롭다운 메뉴입니다 AWS 계정 . 나중에 사용할 수 있도록 `sso_region` 설정에 리전 코드를 기록해 둡니다. 이 코드는 `us-east-1`과 비슷합니다.
3. 다음과 같이 SSO 사용자를 생성합니다.

    a. 사용자 페이지로 이동합니다.

    b. 사용자 추가를 선택하고 사용자의 사용자 이름, 이메일 주소, 이름, 성을 입력합니다. 그리고 다음을 선택합니다.

    c. 그룹 페이지에서 다음을 선택한 다음 정보를 검토하고 사용자 추가를 선택합니다.

4. 다음과 같이 그룹을 생성합니다.

   a. 그룹 페이지로 이동합니다.

   b. 그룹 생성을 선택하고 그룹의 그룹 이름과 설명을 입력합니다.

   c. 그룹에 사용자 추가 섹션에서 이전에 만든 테스트 SSO 사용자를 선택합니다. 그런 다음, 그룹 생성을 선택합니다.

5. 다음과 같이 권한 세트를 생성합니다.

   a. 권한 세트 페이지로 이동하여 권한 세트 생성을 선택합니다.

   b. 권한 세트 유형에서 사용자 지정 권한 세트를 선택하고 다음을 선택합니다.

   c. 인라인 정책을 열고 다음 정책을 입력합니다.

      JSON

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

   d. 이 자습서에서는 권한 세트 이름으로 SSOReadOnlyRole을 입력합니다. 원하는 경우 설명을 추가하고 다음을 선택합니다.

   e. 정보를 검토한 후 생성을 선택합니다.

   f. 나중에 사용할 수 있도록 sso_role_name 설정에 권한 세트의 이름을 기록해 둡니다.

6. AWS 계정 페이지로 이동하여 이전에 조직에 추가한 AWS 계정을 선택합니다.

7. 해당 페이지의 개요 섹션에서 계정 ID를 찾아 나중에 sso_account_id 설정에서 사용할 수 있도록 기록해 둡니다.

8. 사용자 및 그룹 탭을 선택한 다음 사용자 또는 그룹 할당을 선택합니다.

9. 사용자 및 그룹 할당 페이지에서 그룹 탭을 선택하고 이전에 만든 그룹을 선택한 후 다음을 선택합니다.
10. 이전에 만든 권한 세트를 선택하고 다음을 선택한 다음 제출을 선택합니다. 구성은 몇 분 정도 걸립니다.

예제 애플리케이션 생성

다음 애플리케이션을 생성합니다. SSO 사용자 컴퓨터에서 실행됩니다.

Amazon S3 버킷 나열

`AWSSDK.S3`및 `AWSSDK.SecurityToken` 외에도 `AWSSDK.SSO` 및 `AWSSDK.SSOOIDC` NuGet 패키지를 포함합니다.

```
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;
        }
    }
}
```

IAM 사용자 나열

AWSSDK.IdentityManagement및 AWSSDK.SecurityToken 외에도 AWSSDK.SSO 및 AWSSDK.SSOOIDC NuGet 패키지를 포함합니다.

```
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;
        }
    }
}
```

Amazon S3 버킷 및 IAM 사용자 목록을 표시하는 것 외에도 이러한 애플리케이션은 이 자습서에서 `my-sso-profile`인 SSO 지원 프로필에 대한 사용자 자격 증명 ARN도 표시합니다.

이러한 애플리케이션은 SSOAWSCredentials 객체의 옵션 속성에 콜백 메서드를 제공하여 SSO 로그인 작업을 수행합니다.

SSO 사용자 지시

SSO 사용자에게 이메일을 확인하고 SSO 초대를 수락하도록 요청하세요. 비밀번호를 설정하라는 메시지가 표시됩니다. 메시지가 SSO 사용자의 받은 편지함에 도착하는 데 몇 분 정도 걸릴 수 있습니다.

SSO 사용자에게 이전에 생성한 애플리케이션을 제공합니다.

그런 다음 SSO 사용자에게 다음을 수행하도록 하세요.

1. 공유 AWS `config` 파일이 포함된 폴더가 없으면 생성합니다. 폴더가 존재하고 `.sso`라는 하위 폴더가 있는 경우 해당 하위 폴더를 삭제하세요.

   이 폴더의 위치는 일반적으로 Windows의 `%USERPROFILE%\.aws`, Linux와 macOS의 `~/.aws`에 있습니다.

2. 필요한 경우 해당 폴더에 공유 AWS `config` 파일을 생성하고 다음과 같이 해당 폴더에 프로필을 추가합니다.

   ```
   [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. Amazon S3 애플리케이션을 실행합니다.
4. 결과 웹 로그인 페이지에서 로그인합니다. 초대 메시지에 있는 사용자 이름과 메시지에 대한 응답으로 생성된 비밀번호를 사용하세요.
5. 로그인이 완료되면 애플리케이션에 S3 버킷 목록이 표시됩니다.
6. IAM 애플리케이션을 실행합니다. 애플리케이션에 IAM 사용자 목록이 표시됩니다. 두 번째 로그인이 수행되지 않았더라도 마찬가지입니다. IAM 애플리케이션은 이전에 생성된 임시 토큰을 사용합니다.

정리

이 자습서에서 생성한 리소스를 보관하지 않으려면 정리하세요. 이러한 AWS 리소스는 파일 및 폴더와 같은 개발 환경의 리소스일 수 있습니다.

AWS CLI 및 .NET 애플리케이션을 사용한 SSO 자습서

이 자습서에서는 기본 .NET 애플리케이션 및 테스트 SSO 사용자를 위해 SSO를 활성화하는 방법을 보여줍니다. 프로그래밍 방식으로 생성하는 대신 AWS CLI 를 사용하여 임시 SSO 토큰을 생성합니다. ???

이 자습서에서는에서 SSO 기능의 일부를 보여줍니다 AWS SDK for .NET. 에서 IAM Identity Center를 사용하는 방법에 대한 자세한 내용은 배경 정보가 포함된 주제를 AWS SDK for .NET참조하세요. 해당 항목에서는 특히 AWS CLI 및 .NET 애플리케이션이라는 하위 섹션의 이 시나리오에 대한 고급 설명을 참조하세요.

> **Note**
>
> 이 자습서의 몇 가지 단계는 AWS Organizations 및 IAM Identity Center와 같은 서비스를 구성하는 데 도움이 됩니다. 이러한 구성을 이미 수행했거나 코드에만 관심이 있다면 예제 코드가 있는 섹션으로 건너뛰어도 됩니다.

사전 조건

- 아직 구성하지 않은 경우 개발 환경을 구성합니다. 이에 대해서는 용 도구 체인 설치 및 구성 AWS SDK for .NET 및 시작와 같은 섹션에 설명되어 있습니다.
- SSO를 테스트하는 데 사용할 수 AWS 계정 있는 항목을 하나 이상 식별하거나 생성합니다. 이 자습서에서는 이를 테스트 AWS 계정 또는 간단히 테스트 계정이라고 합니다.
- 대신 SSO를 테스트할 수 있는 SSO 사용자를 식별합니다. 이 사람은 사용자가 생성한 기본 애플리케이션과 SSO를 사용할 사람입니다. 이 자습서에서 그 사람은 사용자(개발자)일 수도 있고 다른 사람일 수도 있습니다. 또한 SSO 사용자가 개발 환경에 있지 않은 컴퓨터에서 작업하도록 설정하는 것이 좋습니다. 그러나 이 작업이 꼭 필요한 것은 아닙니다.
- SSO 사용자 컴퓨터에는 개발 환경을 설정하는 데 사용한 프레임워크와 호환되는 .NET 프레임워크가 설치되어 있어야 합니다.

- SSO 사용자의 컴퓨터에 AWS CLI 버전 2가 설치되어 있는지 확인합니다. 명령 프롬프트나 터미널에서 `aws --version`을 실행하여 확인할 수 있습니다.

설정 AWS

이 섹션에서는이 자습서를 위해 다양한 AWS 서비스를 설정하는 방법을 보여줍니다.

이 설정을 수행하려면 먼저 관리자 AWS 계정 로 테스트에 로그인합니다. 뒤이어 다음과 같이 하세요.

Amazon S3

Amazon S3 콘솔로 이동하여 무해한 버킷을 추가하십시오. 이 자습서의 뒷부분에서 SSO 사용자는 이러한 버킷의 목록을 검색하게 됩니다.

AWS IAM

IAM 콘솔로 이동하여 IAM 사용자 몇 명을 추가합니다. IAM 사용자에게 권한을 부여하는 경우 권한을 몇 가지 무해한 읽기 전용 권한으로 제한하십시오. 이 자습서의 뒷부분에서 SSO 사용자는 이러한 IAM 사용자의 목록을 검색하게 됩니다.

AWS Organizations

AWS Organizations 콘솔로 이동하여 조직을 활성화합니다. 자세한 내용은 AWS Organizations 사용 설명서의 조직 생성을 참조하세요.

이 작업은 AWS 계정 조직에 테스트를 관리 계정으로 추가합니다. 추가 테스트 계정이 있는 경우 해당 계정을 조직에 가입하도록 초대할 수 있지만 이 자습서에서는 그렇게 하지 않아도 됩니다.

IAM Identity Center

IAM ID Center 콘솔로 이동하여 SSO를 활성화하십시오. 필요한 경우 이메일 확인을 수행합니다. 자세한 내용은 IAM Identity Center 사용 설명서의 IAM Identity Center 활성화를 참조하세요.

그런 다음 다음과 같이 구성합니다.

IAM Identity Center 구성

1. 설정 페이지로 이동합니다. "액세스 포털 URL"을 찾아 나중에 사용할 수 있도록 `sso_start_url` 설정에 값을 기록해 두세요.
2. 의 배너에서 SSO를 활성화할 때 설정된 AWS 리전 를 AWS Management Console찾습니다. ID 왼쪽의 드롭다운 메뉴입니다 AWS 계정 . 나중에 사용할 수 있도록 `sso_region` 설정에 리전 코드를 기록해 둡니다. 이 코드는 `us-east-1`과 비슷합니다.
3. 다음과 같이 SSO 사용자를 생성합니다.
    a. 사용자 페이지로 이동합니다.
    b. 사용자 추가를 선택하고 사용자의 사용자 이름, 이메일 주소, 이름, 성을 입력합니다. 그리고 다음을 선택합니다.
    c. 그룹 페이지에서 다음을 선택한 다음 정보를 검토하고 사용자 추가를 선택합니다.
4. 다음과 같이 그룹을 생성합니다.
    a. 그룹 페이지로 이동합니다.
    b. 그룹 생성을 선택하고 그룹의 그룹 이름과 설명을 입력합니다.

c. 그룹에 사용자 추가 섹션에서 이전에 만든 테스트 SSO 사용자를 선택합니다. 그런 다음, 그룹 생성을 선택합니다.

5. 다음과 같이 권한 세트를 생성합니다.

a. 권한 세트 페이지로 이동하여 권한 세트 생성을 선택합니다.

b. 권한 세트 유형에서 사용자 지정 권한 세트를 선택하고 다음을 선택합니다.

c. 인라인 정책을 열고 다음 정책을 입력합니다.

JSON

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

d. 이 자습서에서는 권한 세트 이름으로 SSOReadOnlyRole을 입력합니다. 원하는 경우 설명을 추가하고 다음을 선택합니다.

e. 정보를 검토한 후 생성을 선택합니다.

f. 나중에 사용할 수 있도록 sso_role_name 설정에 권한 세트의 이름을 기록해 둡니다.

6. AWS 계정 페이지로 이동하여 이전에 조직에 추가한 AWS 계정을 선택합니다.

7. 해당 페이지의 개요 섹션에서 계정 ID를 찾아 나중에 sso_account_id 설정에서 사용할 수 있도록 기록해 둡니다.

8. 사용자 및 그룹 탭을 선택한 다음 사용자 또는 그룹 할당을 선택합니다.

9. 사용자 및 그룹 할당 페이지에서 그룹 탭을 선택하고 이전에 만든 그룹을 선택한 후 다음을 선택합니다.

10. 이전에 만든 권한 세트를 선택하고 다음을 선택한 다음 제출을 선택합니다. 구성은 몇 분 정도 걸립니다.

예제 애플리케이션 생성

다음 애플리케이션을 생성합니다. SSO 사용자 컴퓨터에서 실행됩니다.

Amazon S3 버킷 나열

AWSSDK.S3및 AWSSDK.SecurityToken 외에도 AWSSDK.SSO 및 AWSSDK.SSOOIDC NuGet 패키지를 포함합니다.

```
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;
        }
    }
}
```

IAM 사용자 나열

AWSSDK.IdentityManagement및 AWSSDK.SecurityToken 외에도 AWSSDK.SSO 및 AWSSDK.SSOOIDC NuGet 패키지를 포함합니다.

```
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;
        }
    }
}
```

Amazon S3 버킷 및 IAM 사용자 목록을 표시하는 것 외에도 이러한 애플리케이션은 이 자습서에서 `my-sso-profile`인 SSO 지원 프로필에 대한 사용자 자격 증명 ARN도 표시합니다.

SSO 사용자 지시

SSO 사용자에게 이메일을 확인하고 SSO 초대를 수락하도록 요청하세요. 비밀번호를 설정하라는 메시지가 표시됩니다. 메시지가 SSO 사용자의 받은 편지함에 도착하는 데 몇 분 정도 걸릴 수 있습니다.

SSO 사용자에게 이전에 생성한 애플리케이션을 제공합니다.

그런 다음 SSO 사용자에게 다음을 수행하도록 하세요.

1. 공유 AWS `config` 파일이 포함된 폴더가 없으면 생성합니다. 폴더가 존재하고 `.sso`라는 하위 폴더가 있는 경우 해당 하위 폴더를 삭제하세요.

이 폴더의 위치는 일반적으로 Windows의 `%USERPROFILE%\.aws`, Linux와 macOS의 `~/.aws`에 있습니다.

2. 필요한 경우 해당 폴더에 공유 AWS `config` 파일을 생성하고 다음과 같이 해당 폴더에 프로필을 추가합니다.

```
[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. Amazon S3 애플리케이션을 실행합니다. 런타임 예외가 나타납니다.
4. 다음 AWS CLI 명령을 실행합니다.

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

5. 결과 웹 로그인 페이지에서 로그인합니다. 초대 메시지에 있는 사용자 이름과 메시지에 대한 응답으로 생성된 비밀번호를 사용하세요.
6. Amazon S3 애플리케이션을 다시 실행합니다. 이제 애플리케이션이 S3 버킷 목록을 표시합니다.
7. IAM 애플리케이션을 실행합니다. 애플리케이션에 IAM 사용자 목록이 표시됩니다. 두 번째 로그인이 수행되지 않았더라도 마찬가지입니다. IAM 애플리케이션은 이전에 생성된 임시 토큰을 사용합니다.

정리

이 자습서에서 생성한 리소스를 보관하지 않으려면 정리하세요. 이러한 AWS 리소스는 파일 및 폴더와 같은 개발 환경의 리소스일 수 있습니다.

에 애플리케이션 배포 AWS

개발 시스템에서 클라우드 네이티브 .NET Core 애플리케이션 또는 서비스를 개발한 후에는 이 애플리케이션 또는 서비스를 AWS에 배포해야 합니다. AWS Management Console 또는와 같은 특정 서비스를 사용하여이 작업을 수행할 수 CloudFormation 있습니다 AWS Cloud Development Kit (AWS CDK). 배포용으로 생성된 AWS 도구를 사용할 수도 있습니다. 이러한 도구를 사용하여 다음 작업을 할 수 있습니다.

.NET CLI에서 배포

.NET CLI에 다음 AWS 도구를 사용하여 애플리케이션을 배포할 수 있습니다. AWS

- AWS .NET CLI용 배포 도구 - AWS App Runner, Amazon Elastic Container Service(Amazon ECS) 및에 대한 배포를 지원합니다AWS Elastic Beanstalk.
- AWS Lambda .NET CLI용 도구 - AWS Lambda 프로젝트 배포를 지원합니다.

IDE 툴킷에서 배포

AWS 도구 키트를 사용하여 선택한 IDE에서 직접 애플리케이션을 배포할 수 있습니다.

- AWS Toolkit for Visual Studio

  > **Note**
  >
  > 도구 키트의 '게시 대상 AWS' 기능은 .NET CLI용 AWS 배포 도구와 동일한 기능을 제공합니다. 자세한 내용은 AWS Toolkit for Visual Studio 사용 설명서의 AWS에 게시를 참조하세요.

- AWS Toolkit for JetBrains

  AWS 서버리스 애플리케이션 작업 및 작업을 참조하세요 AWS App Runner.

- VS 코드용AWS 도구 키트

  서버리스 애플리케이션 작업 및 AWS App Runner사용을 참조하세요.

- AWS Toolkit for Azure DevOps

사용 사례

다음 섹션에는 .NET CLI를 사용하여 해당 애플리케이션을 배포하는 방법에 대한 정보를 포함하여 특정 유형의 애플리케이션에 대한 사용 사례 시나리오가 포함되어 있습니다.

- ASP.NET Core 앱
- .NET 콘솔 앱
- Blazor WebAssembly 앱
- AWS Lambda 프로젝트

ASP.NET Core 앱

AWS Deploy Tool for .NET CLI는 ASP.NET 애플리케이션을 배포하는 데 도움이 되며 배포 프로세스를 안내합니다. 최소한의 AWS 지식으로 .NET 애플리케이션을 배포하는 데 도움이 되는 .NET CLI용 대화형 도구입니다.

배포 도구에는 다음 기능이 있습니다.

- 애플리케이션에 대한 컴퓨팅 권장 사항 - 컴퓨팅 권장 사항을 확인하고 애플리케이션에 가장 적합한 AWS 컴퓨팅을 알아봅니다.
- Dockerfile 생성 - 이 도구는 필요한 경우 Dockerfile을 생성하거나 기존 Dockerfile을 사용합니다.
- 자동 패키징 및 배포 -이 도구는 배포 아티팩트를 빌드하고, 생성된 AWS CDK 배포 프로젝트를 사용하여 인프라를 프로비저닝하고, 선택한 AWS 컴퓨팅에 애플리케이션을 배포합니다.
- 반복 가능하고 공유 가능한 배포 - 특정 사용 사례에 맞게 AWS CDK 배포 프로젝트를 생성하고 수정할 수 있습니다. 또한 프로젝트의 버전을 관리하고 팀과 공유하여 반복 배포할 수 있습니다.
- .NET에 AWS CDK 대한 학습 지원 -이 도구는와 같은 기본 AWS 도구를 점진적으로 학습하는 데 도움이 됩니다 AWS CDK.

AWS 배포 도구는 다음 AWS 서비스에 ASP.NET Core 애플리케이션 배포를 지원합니다.

- 를 사용하는 Amazon ECS 서비스 AWS Fargate - AWS Fargate 서버리스 컴퓨팅 엔진에서 관리하는 컴퓨팅 파워를 사용하여 Amazon Elastic Container Service(Amazon ECS)에 웹 애플리케이션 배포를 지원합니다.
- AWS App Runner - 개발자가 컨테이너화된 웹 애플리케이션 및 API를 대규모로 손쉽게 배포할 수 있도록 지원하는 완전 관리형 서비스로의 배포를 지원합니다. 인프라 관련 사전 경험이 없어도 됩니다.

- AWS Elastic Beanstalk - 개발자가 웹 애플리케이션 및 API를 완전 관리형 환경으로 대규모로 손쉽게 배포할 수 있도록 지원하는 서비스로의 배포를 지원합니다. 인프라 관련 사전 경험이 없어도 됩니다.

자세히 알아보려면 도구 개요를 참조하세요. 여기에서 시작하려면 설명서, 시작하기로 이동한 다음 설치 방법을 선택하여 설치 지침을 확인하세요.

.NET 콘솔 앱

AWS Deploy Tool for .NET CLI는 .NET 콘솔 애플리케이션을 Linux에서 서비스 또는 예약된 작업으로 컨테이너 이미지로 배포하는 데 도움이 되며 배포 프로세스를 안내합니다. 애플리케이션에 Dockerfile이 없는 경우 도구가 Dockerfile을 자동으로 생성합니다. 그렇지 않으면 기존 Dockerfile이 사용됩니다.

배포 도구에는 다음 기능이 있습니다.

- 애플리케이션에 대한 컴퓨팅 권장 사항 - 컴퓨팅 권장 사항을 확인하고 애플리케이션에 가장 적합한 AWS 컴퓨팅을 알아봅니다.
- Dockerfile 생성 - 이 도구는 필요한 경우 Dockerfile을 생성하거나 기존 Dockerfile을 사용합니다.
- 자동 패키징 및 배포 -이 도구는 배포 아티팩트를 빌드하고, 생성된 AWS CDK 배포 프로젝트를 사용하여 인프라를 프로비저닝하고, 선택한 AWS 컴퓨팅에 애플리케이션을 배포합니다.
- 반복 가능하고 공유 가능한 배포 - 특정 사용 사례에 맞게 AWS CDK 배포 프로젝트를 생성하고 수정할 수 있습니다. 또한 프로젝트의 버전을 관리하고 팀과 공유하여 반복 배포할 수 있습니다.
- .NET AWS CDK 학습 지원 -이 도구는와 같이 기본으로 구축된 기본 AWS 도구를 점진적으로 학습하는 데 도움이 됩니다 AWS CDK.

AWS 배포 도구는 다음 AWS 서비스에 .NET 콘솔 애플리케이션 배포를 지원합니다.

- 를 사용하는 Amazon ECS 서비스 AWS Fargate - AWS Fargate 서버리스 컴퓨팅 엔진에서 관리하는 컴퓨팅 파워를 사용하여 Amazon Elastic Container Service(Amazon ECS)에 서비스형 .NET 애플리케이션(예: 백그라운드 프로세서) 배포를 지원합니다.
- 를 사용한 Amazon ECS 예약된 작업 AWS Fargate - AWS Fargate 서버리스 컴퓨팅 엔진에서 관리하는 컴퓨팅 파워를 사용하여 Amazon ECS에 예약된 작업(예: end-of-day 프로세스)으로 .NET 애플리케이션의 배포를 지원합니다.

자세히 알아보려면 도구 개요를 참조하세요. 여기에서 시작하려면 설명서, 시작하기로 이동한 다음 설치 방법을 선택하여 설치 지침을 확인하세요.

Blazor WebAssembly 앱

AWS Deploy Tool for .NET CLI를 사용하면 콘텐츠 네트워크 전송에 Amazon CloudFront를 사용하여 Amazon S3에서 Blazor WebAssembly 애플리케이션을 호스팅할 수 있습니다. 앱은 웹 호스팅을 위해 S3 버킷으로 배포됩니다. 도구는 S3 버킷을 생성하고 구성한 다음, Blazor 애플리케이션을 버킷에 업로드합니다.

배포 도구에는 다음 기능이 있습니다.

- 자동 패키징 및 배포 -이 도구는 배포 아티팩트를 빌드하고, 생성된 AWS CDK 배포 프로젝트를 사용하여 인프라를 프로비저닝하고, 선택한 AWS 컴퓨팅에 애플리케이션을 배포합니다.
- 반복 가능하고 공유 가능한 배포 - 특정 사용 사례에 맞게 AWS CDK 배포 프로젝트를 생성하고 수정할 수 있습니다. 또한 프로젝트의 버전을 관리하고 팀과 공유하여 반복 배포할 수 있습니다.
- .NET에 AWS CDK 대한 학습 지원 -이 도구는와 같은 기본 AWS 도구를 점진적으로 학습하는 데 도움이 됩니다 AWS CDK.

자세히 알아보려면 도구 개요를 참조하세요. 여기에서 시작하려면 설명서, 시작하기로 이동한 다음 설치 방법을 선택하여 설치 지침을 확인하세요.

AWS Lambda 프로젝트

AWS Lambda 는 서버를 프로비저닝하거나 관리하지 않고도 코드를 실행할 수 있는 컴퓨팅 서비스입니다. 고가용성 컴퓨팅 인프라에서 코드를 실행하고 모든 컴퓨팅 리소스 관리를 수행합니다. Lambda에 대한 자세한 내용은 AWS Lambda 개발자 안내서의 AWS Lambda란 무엇입니까?를 참조하세요.

.NET 명령줄 인터페이스(CLI)를 사용하여 Lambda 함수를 배포할 수 있습니다.

주제

- 사전 조건
- 사용 가능한 Lambda 명령
- 배포 단계

사전 조건

.NET CLI를 사용하여 Lambda 함수를 배포하기 전에 먼저 다음 사전 조건을 충족해야 합니다.

- .NET CLI가 설치되어 있는지 확인합니다. 예를 들어 `dotnet --version`입니다. 필요한 경우 https://dotnet.microsoft.com/download으로 이동하여 설치합니다.
- .NET CLI가 Lambda와 함께 작동하도록 설정합니다. 이 작업을 수행하는 방법에 대한 설명은 AWS Lambda 개발자 안내서의 .NET Core CLI를 참조하세요. 이 절차에서 배포 명령은 다음과 같습니다.

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

  이 연습의 IAM 역할을 생성하는 방법을 잘 모르는 경우 해당 `--function-role role` 부분을 포함하지 마세요. 이 도구를 사용하면 새 역할을 생성하는 데 도움이 됩니다.

사용 가능한 Lambda 명령

.NET CLI를 통해 사용할 수 있는 Lambda 명령을 나열하려면 명령 프롬프트 또는 터미널을 열고 `dotnet lambda --help`를 입력합니다. 다음과 같은 명령 결과가 출력됩니다.

```
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>
```

출력에는 현재 사용할 수 있는 모든 명령이 나열됩니다.

배포 단계

다음 지침에서는 사용자가 AWS Lambda .NET 프로젝트를 생성했다고 가정합니다. 이 절차의 목적에 따라 프로젝트 이름은 `DotNetCoreLambdaTest`입니다.

1. 명령 프롬프트 또는 터미널을 열고 .NET Lambda 프로젝트 파일이 포함된 폴더로 이동합니다.
2. `dotnet lambda deploy-function`을 입력합니다.
3. 메시지가 표시되면 AWS 리전(Lambda 함수가 배포될 리전)을 입력합니다.

4. 메시지가 나타나면 배포할 함수의 이름(예: `DotNetCoreLambdaTest`)을 입력합니다. 이미 AWS 계정 에 있는 함수의 이름이거나 아직 배포되지 않은 함수의 이름일 수 있습니다.
5. 메시지가 나타나면 함수를 실행할 때 Lambda가 맡을 IAM 역할을 선택하거나 생성합니다.

성공적으로 완료되면 새 Lambda 함수 생성됨 메시지가 표시됩니다.

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

계정에 이미 있는 함수를 배포하는 경우 배포 함수는 필요한 경우 AWS 리전만 요청합니다. 이 경우 명령 출력은 `Updating code for existing function`으로 끝납니다.

Lambda 함수가 배포되면 해당 함수를 사용할 준비가 되었습니다. 자세한 정보는 AWS Lambda 사용 예제를 참조하세요.

Lambda는 자동으로 Lambda 함수를 모니터링하고 Amazon CloudWatch를 통해 지표를 보고합니다. Lambda 함수를 모니터링하고 문제를 해결하려면 Lambda 애플리케이션 모니터링 및 문제 해결을 참조하세요.

에서 AWS 서비스 호출 AWS SDK for .NET

다음 섹션에는를 사용하여 AWS 서비스를 AWS SDK for .NET 사용하는 방법을 보여주는 예제, 자습서, 작업 및 가이드가 포함되어 있습니다. 이러한 예제와 자습서는 AWS SDK for .NET 에서 제공하는 API를 기반으로 합니다. API에서 사용할 수 있는 클래스와 메서드를 확인하려면 AWS SDK for .NET API 참조를 참조하세요.

를 처음 사용하는 경우 먼저 간단한 애플리케이션 생성 주제를 확인할 AWS SDK for .NET수 있습니다. SDK에 대한 소개를 제공합니다.

AWS 코드 예제 리포지토리 및 GitHub의 awslabs 리포지토리에서 더 많은 코드 예제를 찾을 수 있습니다.

시작하기 전에 환경을 설정하고 프로젝트를 구성했는지 확인합니다. 또한 SDK 사용의 정보를 검토합니다.

주제

- 에 대한 가이드 코드 예제 AWS SDK for .NET
- AWS Lambda 컴퓨팅 서비스에 사용
- 에 대한 상위 수준 라이브러리 및 프레임워크 AWS SDK for .NET
- 기타 AWS 서비스 및 구성 지원

에 대한 가이드 코드 예제 AWS SDK for .NET

다음 섹션에는 코드 예제가 포함되어 있으며 예제에 대한 지침을 제공합니다. 이를 통해를 사용하여 AWS 서비스를 AWS SDK for .NET 사용하는 방법을 배울 수 있습니다. 추가 코드 예제는 코드 예제 섹션을 참조하세요.

> **Note**
>
> 의 V3와 관련된 코드 예제 AWS SDK for .NET 도 사용할 수 있습니다. 이를 찾으려면 AWS SDK for .NET (V3) 개발자 안내서의 지침이 포함된 코드 예제와 코드 예제를 참조하세요. 그러나 SDK의 V3-specific 코드 예제를 사용하는 경우의 정보에 따라 조정해야 할 수 있습니다버전 4로 마이그레이션. V4

를 처음 사용하는 경우 먼저 간단한 애플리케이션 생성 주제를 확인할 AWS SDK for .NET수 있습니다. SDK에 대한 소개를 제공합니다.

시작하기 전에 환경을 설정하고 프로젝트를 구성했는지 확인합니다. 또한 SDK 사용의 정보를 검토합니다.

주제

- 를 CloudFormation 사용하여 액세스 AWS SDK for .NET
- Amazon Cognito로 사용자 인증
- Amazon DynamoDB NoSQL 데이터베이스 사용
- Amazon EC2 작업
- 를 사용하여 AWS Identity and Access Management (IAM)에 액세스 AWS SDK for .NET
- Amazon Simple Storage Service 인터넷 스토리지 사용
- Amazon Simple Notification Service를 사용하여 클라우드에서 알림 전송
- Amazon SQS를 사용한 메시징

를 CloudFormation 사용하여 액세스 AWS SDK for .NET

는 AWS 인프라 배포를 예측 가능하고 반복적으로 생성하고 프로비저닝AWS CloudFormation하는를 AWS SDK for .NET 지원합니다.

API

는 CloudFormation 클라이언트를 위한 APIs AWS SDK for .NET 제공합니다. APIs 사용하면 템플릿 및 스택과 같은 CloudFormation 기능을 사용할 수 있습니다. 이 섹션에는 이러한 API로 작업할 때 따를 수 있는 패턴을 보여주는 몇 가지 예제가 포함되어 있습니다. 전체 API 세트를 보려면 AWS SDK for .NET API 참조를 참조하세요(그리고 "Amazon.CloudFormation"으로 스크롤하세요).

AWS CloudFormation APIs는 AWSSDK.CloudFormation 패키지에서 제공됩니다.

사전 조건

시작하기 전에 환경을 설정하고 프로젝트를 구성했는지 확인합니다. 또한 SDK 사용의 정보를 검토합니다.

주제

주제

- 를 사용하여 AWS 리소스 나열 AWS CloudFormation

를 사용하여 AWS 리소스 나열 AWS CloudFormation

이 예제에서는를 사용하여 CloudFormation 스택의 리소스를 나열 AWS SDK for .NET 하는 방법을 보여줍니다. 이 예제에서는 하위 수준 API를 사용합니다. 애플리케이션에 인수가 없지만 단순히 사용자 보안 인증으로 액세스할 수 있는 모든 스택에 대한 정보를 수집한 다음 해당 스택에 대한 정보를 표시합니다.

SDK 레퍼런스

NuGet 패키지:

- AWSSDK.CloudFormation

프로그래밍 요소:

- 네임스페이스 Amazon.CloudFormation

  클래스 AmazonCloudFormationClient
- 네임스페이스 Amazon.CloudFormation.Model

  클래스 ICloudFormationPaginatorFactory.DescribeStacks

  클래스 DescribeStackResourcesRequest

  클래스 DescribeStackResourcesResponse

  클래스 Stack

  클래스 StackResource

  클래스 Tag

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

namespace CloudFormationActions;
```

```
public static class HelloCloudFormation
{
    public static IAmazonCloudFormation _amazonCloudFormation;

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

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

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

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

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

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

```
                }

                // The resources of each stack
                DescribeStackResourcesResponse responseDescribeResources =
                    await _amazonCloudFormation.DescribeStackResourcesAsync(
                        new DescribeStackResourcesRequest
                        {
                            StackName = stack.StackName
                        });
                if (responseDescribeResources.StackResources.Count > 0)
                {
                    Console.WriteLine("  Resources:");
                    foreach (StackResource resource in responseDescribeResources
                                 .StackResources)
                        Console.WriteLine(
                            $"    {resource.LogicalResourceId}:
{resource.ResourceStatus}");
                }
            }

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

            return false;
        }
```

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

            return false;
        }
    }
```

Amazon Cognito로 사용자 인증

> **Note**
>
> 이 주제의 정보는 .NET Framework 및 AWS SDK for .NET 버전 3.3 이하를 기반으로 하는 프로젝트에만 해당됩니다.

Amazon Cognito 자격 증명을 사용하면 사용자를 위한 고유한 자격 증명을 생성하고 Amazon S3 또는 Amazon DynamoDB와 같은 AWS 리소스에 대한 보안 액세스를 위해 사용자를 인증할 수 있습니다. Amazon Cognito Identity는 Amazon, Facebook, Twitter/Digits, Google 또는 OpenID Connect 호환 공급자와 같은 퍼블릭 ID 공급자와 인증되지 않은 자격 증명을 지원합니다. 또한 Amazon Cognito는 개발자 인증 자격 증명을 지원하여 Amazon Cognito Sync를 사용하여 사용자 데이터를 동기화하고 AWS 리소스에 액세스하면서도 자체 백엔드 인증 프로세스를 사용하여 사용자를 등록하고 인증할 수 있게 합니다.

Amazon Cognito에 대한 자세한 내용은 Amazon Cognito 개발자 안내서를 참조하세요.

다음 코드 예시는 Amazon Cognito 자격 증명을 쉽게 사용하는 방법을 보여줍니다. 자격 증명 공급자 예에서는 사용자 자격 증명을 생성하고 인증하는 방법을 보여줍니다. CognitoAuthentication 확장 라이브러리 예제에서는 CognitoAuthentication 확장 라이브러리를 사용하여 Amazon Cognito 사용자 풀을 인증하는 방법을 보여줍니다.

주제

- Amazon Cognito 보안 인증 공급자
- Amazon CognitoAuthentication 확장 라이브러리 예

Amazon Cognito 보안 인증 공급자

> **Note**
>
> 이 주제의 정보는 .NET Framework 및 AWS SDK for .NET 버전 3.3 이하를 기반으로 하는 프로젝트에만 해당됩니다.

`Amazon.CognitoIdentity.CognitoAWSCredentials`AWSSDK.CognitoIdentity NuGet 패키지에 있는는 Amazon Cognito 및 AWS Security Token Service (AWS STS)를 사용하여 AWS 호출할 자격 증명을 검색하는 자격 증명 객체입니다.

`CognitoAWSCredentials` 설정의 첫 번째 단계는 "자격 증명 풀"을 생성하는 것입니다. (자격 증명 풀은 계정에 관련된 사용자 자격 증명 정보의 저장소입니다. 정보는 클라이언트 플랫폼, 디바이스 및 운영 체제 간에 가져올 수 있어 사용자가 스마트폰에서 앱을 사용하다가 태블릿으로 전환하면 앱 정보가 해당 사용자에게 계속 제공됩니다. Amazon Cognito 콘솔에서 새로운 자격 증명 풀을 생성할 수 있습니다. 콘솔을 사용 중인 경우, 콘솔에서 사용자에게 필요한 다음과 같은 다른 정보 또한 제공합니다.

- 계정 번호- A 12자리 숫자. 예: 123456789012 (사용자 계정에 고유한 숫자)
- 인증되지 않은 역할 ARN- 인증되지 않은 사용자가 맡을 역할 예를 들어, 이 역할은 데이터에 대한 읽기 전용 권한을 제공할 수 있습니다.
- 인증된 역할 ARN- 인증된 사용자가 맡을 역할 이 역할은 데이터에 대한 더 광범위한 권한을 제공할 수 있습니다.

CognitoAWSCredentials 설정

다음 코드 예제에서는 인증되지 않은 사용자로 Amazon S3를 호출하는 데 사용할 수 있도록 `CognitoAWSCredentials`를 설정하는 방법을 보여줍니다. 이렇게 하면 사용자를 인증하기 위해 필요한 최소한의 데이터만으로도 호출할 수 있습니다. 사용자 권한은 역할에 의해 제어되므로 필요한 액세스를 구성할 수 있습니다.

```
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();
}
```

인증되지 않은 사용자 AWS 로 사용

다음 코드 예제에서는 미인증 사용자 AWS 로 사용을 시작한 다음 Facebook을 통해 인증하고 자격 증명을 업데이트하여 Facebook 자격 증명을 사용하는 방법을 보여줍니다. 이 접근 방식을 사용하여 인증된 역할을 통해 인증된 사용자에게 다른 자격 증명을 부여할 수 있습니다. 예를 들어, 사용자가 익명으로 콘텐츠를 볼 수 있도록 허용하되 하나 이상의 구성된 공급자로 로그온한 경우에 게시할 수 있도록 허용하는 스마트폰 애플리케이션이 있을 수도 있습니다.

```
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();
}
```

`CognitoAWSCredentials` 객체는 AWS SDK for .NET의 일부인 `AmazonCognitoSyncClient`와 함께 사용하는 경우 훨씬 더 많은 기능을 제공합니다. `AmazonCognitoSyncClient`와 `CognitoAWSCredentials`를 모두 사용 중인 경우 `IdentityPoolId`로 호출할 때 `IdentityId` 및 `AmazonCognitoSyncClient` 속성을 지정하지 않아도 됩니다. 이러한 속성은 `CognitoAWSCredentials`에서 자동으로 채워집니다. 다음 코드 예에서는 이를 보여주고, 뿐만 아니

라 IdentityId에 대한 CognitoAWSCredentials가 변경될 때마다 이를 알려주는 이벤트도 보여 줍니다. IdentityId는 인증되지 않은 사용자에서 인증된 사용자로 변경될 때와 같은 일부 경우에 변경될 수 있습니다.

```
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 확장 라이브러리 예

> **Note**
>
> 이 주제의 정보는 .NET Framework 및 AWS SDK for .NET 버전 3.3 이하를 기반으로 하는 프로젝트에만 해당됩니다.

Amazon.Extensions.CognitoAuthentication NuGet 패키지에 있는 CognitoAuthentication 확장 라이브러리는 .NET Core 및 Xamarin 개발자를 위한 Amazon Cognito 사용자 풀의 인증 프로세스를 간소화합니다. 이 라이브러리는 사용자 인증 API 직접 호출을 생성하고 전송하기 위해 Amazon Cognito Identity Provider API를 기반으로 하여 구축됩니다.

CognitoAuthentication 확장 라이브러리 사용

Amazon Cognito에는 표준 인증 흐름에서 SRP(Secure Remote Password)를 통해 사용자 이름과 암호를 확인할 수 있는 기본 제공 `AuthFlow` 및 `ChallengeName` 값이 있습니다. 인증 흐름에 대한 자세한 내용은 Amazon Cognito 사용자 풀 인증 흐름을 참조하십시오.

다음 예제에는 다음과 같은 `using` 설명문이 필요합니다.

```
// 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;
```

기본 사용자 인증 사용

요청에 서명하지 않아도 되는 AnonymousAWSCredentials를 사용하여 AmazonCognitoIdentityProviderClient를 생성합니다. 리전을 공급할 필요가 없습니다. 리전이 제공되지 않은 경우 기본 코드가 `FallbackRegionFactory.GetRegionEndpoint()`를 호출합니다. `CognitoUserPool` 및 `CognitoUser` 객체를 생성합니다. 사용자 암호를 포함한 `StartWithSrpAuthAsync`로 `InitiateSrpAuthRequest` 메서드를 호출합니다.

```
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;
```

```
}
```

챌린지로 인증

NewPasswordRequired 및 Multi-Factor Authentication(MFA)과 같은 챌린지로 인증 흐름을 계속하는 것 또한 간단합니다. 요구 사항은 CognitoAuthentication 객체, SRP에 대한 사용자 암호, 및 다음 챌린지(사용자에게 정보를 입력하라는 메시지를 표시한 후에 확보됨)를 위해 필요한 정보 뿐입니다. 다음 코드에서는 챌린지 유형을 확인하고 인증 흐름 동안 MFA 및 NewPasswordRequired 챌린지에 대한 적절한 응답을 얻기 위한 하나의 방법을 보여줍니다.

전과 같이 기본 인증 요청을 사용하고 `await`를 `AuthFlowResponse` 합니다. 응답이 반환된 `AuthenticationResult` 객체를 통해 받은 루프인 경우. `ChallengeName` 유형이 `NEW_PASSWORD_REQUIRED`인 경우 `RespondToNewPasswordRequiredAsync` 메서드를 호출합니다.

```
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.");
    }

}
```

인증 후 AWS 리소스 사용

사용자가 CognitoAuthentication 라이브러리를 사용하여 인증되면 다음 단계는 사용자가 적절한 AWS 리소스에 액세스할 수 있도록 허용하는 것입니다. 이렇게 하려면 Amazon Cognito 연동 자격 증명 콘솔을 통해 자격 증명 풀을 생성해야 합니다. poolID 및 clientID를 사용하여 공급자로 생성한 Amazon Cognito 사용자 풀을 지정하면, Amazon Cognito 사용자 풀 사용자가 계정에 연결된 AWS 리소스에 액세스하도록 허용할 수 있습니다. 인증되지 않은 사용자와 인증된 사용자가 서로 다른 리소스에 액세스

할 수 있도록 서로 다른 역할을 지정할 수도 있습니다. 역할의 연결된 정책의 Action 필드에서 권한을 추가하거나 제거할 수 있는 IAM 콘솔에서 이러한 역할을 변경할 수 있습니다. 그런 다음 적절한 자격 증명 풀, 사용자 풀 및 Amazon Cognito 사용자 정보를 사용하여 다른 AWS 리소스를 호출할 수 있습니다. 다음 예제에서는 연결된 자격 증명 풀의 역할이 허용하는 다양한 Amazon S3 버킷에 액세스하여 SRP로 인증된 사용자를 보여줍니다.

```
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);
        }
    }
}
```

더 많은 인증 옵션

SRP, NewPasswordRequired 및 MFA 뿐만 아니라 CognitoAuthentication 확장 라이브러리도 다음을 위해 더 쉬운 인증 흐름을 제공합니다.

- 사용자 지정 - `StartWithCustomAuthAsync(InitiateCustomAuthRequest customRequest)`에 대한 호출로 시작합니다.
- RefreshToken - `StartWithRefreshTokenAuthAsync(InitiateRefreshTokenAuthRequest refreshTokenRequest)`에 대한 호출로 시작합니다.
- RefreshTokenSRP - `StartWithRefreshTokenAuthAsync(InitiateRefreshTokenAuthRequest refreshTokenRequest)`에 대한 호출로 시작합니다.
- AdminNoSRP - `StartWithAdminNoSrpAuthAsync(InitiateAdminNoSrpAuthRequest adminAuthRequest)`에 대한 호출로 시작합니다.

원하는 흐름에 따라 적절한 메서드를 호출합니다. 그런 다음 각 메서드 호출의 `AuthFlowResponse` 객체에 제시되어 있는 챌린지로 사용자에게 프롬프트 표시를 계속합니다. 또한 MFA 챌린지에 대한 `RespondToSmsMfaAuthAsync` 및 사용자 지정 챌린지에 대한 `RespondToCustomAuthAsync`와 같은 적절한 응답 메서드를 호출합니다.

Amazon DynamoDB NoSQL 데이터베이스 사용

> **Note**
>
> 이 항목의 프로그래밍 모델은 .NET Framework와 .NET(Core) 모두에 제공되지만 호출 규칙은 동기식이든 비동기식이든 서로 다릅니다.

는에서 제공하는 빠른 NoSQL 데이터베이스 서비스인 Amazon DynamoDB를 AWS SDK for .NET 지원합니다 AWS. SDK는 DynamoDB와 통신하기 위해 하위 레벨 모델, 문서 모델, 객체 지속성 모델이라는 세 가지 프로그래밍 모델을 제공합니다.

다음 정보는 이러한 모델과 해당 API를 소개하고 이러한 모델과 해당 API를 어떻게, 언제 사용하는지에 대한 예제를 제공하며 AWS SDK for .NET의 추가 DynamoDB 프로그래밍 리소스에 대한 링크를 제공합니다.

하위 수준 모델

하위 레벨 프로그래밍 모델은 DynamoDB 서비스에 대한 직접적인 호출을 래핑합니다. Amazon.DynamoDBv2 네임스페이스를 통해 이 모델에 액세스합니다.

세 모델 중에서 하위 수준 모델은 사용자가 코드 대부분을 작성해야 합니다. 예를 들면 .NET 데이터 형식을 동등한 DynamoDB 데이터 형식으로 전환해야 합니다. 그러나 이 모델을 사용하면 대부분의 기능에 액세스할 수 있습니다.

다음 예제에서는 하위 레벨 모델을 사용하여 DynamoDB에서 테이블을 생성하고 테이블을 수정하며 테이블에 항목을 삽입하는 방법을 보여줍니다.

표 생성

다음 예제에서는 `CreateTable` 클래스의 `AmazonDynamoDBClient` 메서드를 사용하여 테이블을 생성합니다. `CreateTable` 메서드에서는 필수 항목 속성 이름, 기본 키 정의, 처리 용량과 같은 특성이 저장된 `CreateTableRequest` 클래스의 인스턴스를 사용합니다. `CreateTable` 메서드는 `CreateTableResponse` 클래스의 인스턴스를 반환합니다.

```
// 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);
}
```

테이블이 수정할 준비가 되었는지 확인

테이블은 변경 또는 수정 전에 미리 그러한 작업을 할 준비가 되어 있어야 합니다. 다음 예제에서는 하위 레벨 모델을 사용하여 DynamoDB의 테이블이 준비되는지 확인하는 방법을 보여줍니다. 이 예제에서 확인할 대상 테이블은 `DescribeTable` 클래스의 `AmazonDynamoDBClient` 메서드를 통해 참조됩니다. 코드는 5초마다 테이블의 `TableStatus` 속성에 대한 값을 확인합니다. 상태가 `ACTIVE`로 설정되어 있으면 테이블은 수정할 준비가 된 것입니다.

```
// 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);
```

테이블에 항목 삽입

다음 예제에서는 하위 레벨 모델을 사용하여 DynamoDB의 테이블에 두 개의 항목을 삽입합니다. 각 항목은 PutItem 클래스의 인스턴스를 사용해 AmazonDynamoDBClient 클래스의 PutItemRequest 메서드를 통해 삽입됩니다. PutItemRequest 클래스의 두 인스턴스 각각은 일련의 항목 속성 값과 함께 항목이 삽입될 테이블의 이름을 사용합니다.

```
// 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);
```

문서 모델

문서 프로그래밍 모델을 사용하면 DynamoDB에서 데이터 작업을 더 쉽게 수행할 수 있습니다. 이 모델은 테이블과 테이블 내 항목에 접근하기 위한 목적으로 고안되었습니다. Amazon.DynamoDBv2.DocumentModel 네임스페이스를 통해 이 모델에 액세스합니다.

하위 레벨 프로그래밍 모델에 비해 문서 모델은 DynamoDB 데이터에 대해 더 쉽게 코딩할 수 있습니다. 예를 들어, 여러 가지 .NET 데이터 형식을 동등한 DynamoDB 데이터 형식으로 전환할 필요가 없습니다. 그러나 이 모델에서는 하위 수준 프로그래밍 모델과 같은 개수의 기능에 액세스하지는 못합니다. 예를 들면 이 모델을 사용하여 테이블의 항목을 생성, 검색, 업데이트 및 삭제할 수 있습니다. 그러나 테이블을 생성하려면 하위 수준 모델을 사용해야 합니다. 객체 지속성 모델에 비해 이 모델은 .NET 객체를 저장, 로드 및 쿼리할 코드를 더 많이 작성해야 합니다.

DynamoDB 문서 프로그래밍 모델에 대한 자세한 내용은 Amazon DynamoDB 개발자 안내서의 .NET: 문서 모델을 참조하세요.

다음 섹션에서는 원하는 DynamoDB 테이블의 표현을 생성하는 방법에 대한 정보와 문서 모델을 사용하여 테이블에 항목을 삽입하고 테이블에서 항목을 가져오는 방법에 대한 예를 제공합니다.

테이블 표현 생성

이 문서 모델을 사용하여 데이터 작업을 수행하려면 먼저 특정 테이블을 나타내는 `Table` 클래스의 인스턴스를 만들어야 합니다. 이렇게 하는 두 가지 기본 방법이 있습니다.

LoadTable 방법

첫 번째 메커니즘은 다음 예제와 마찬가지로 `Table` 클래스의 정적 `LoadTable` 메서드 중 하나를 사용하는 것입니다.

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

> **Note**
>
> 이 메커니즘은 작동하지만 특정 조건에서는 콜드 스타트 및 스레드 풀 동작으로 인해 추가 지연 시간이나 교착 상태가 발생할 수 있습니다. 이러한 동작에 대한 자세한 내용은 AWS SDK for .NET에 대한 향상된 DynamoDB 초기화 패턴 블로그 게시물을 참조하세요.

TableBuilder

대체 메커니즘인 `TableBuilder` 클래스는 AWSSDK.DynamoDBv2 NuGet 패키지의 버전 3.7.203에 도입되었습니다. 이 메커니즘은 특정 암시적 메서드 호출, 특히 `DescribeTable` 메서드를 제거하여 위에서 언급한 동작을 해결할 수 있습니다. 이 메커니즘은 다음 예제와 비슷한 방식으로 사용됩니다.

```
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();
```

이 대체 메커니즘에 대한 자세한 내용은 AWS SDK for .NET에 대한 향상된 DynamoDB 초기화 패턴 블로그 게시물을 참조하세요.

테이블에 항목 삽입

다음 예제에서는 `PutItemAsync` 클래스의 `Table` 메서드를 통해 응답이 Reply 테이블에 삽입됩니다. `PutItemAsync` 메서드는 `Document` 클래스의 인스턴스를 취하고, `Document` 클래스는 단지 초기화된 속성을 모아놓은 것입니다.

```
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);
```

테이블에서 항목 가져오기

다음 예제에서는 GetItemAsync 클래스의 Table 메서드를 통해 응답을 가져옵니다. 가져올 응답을 결정하기 위해 GetItemAsync 메서드는 대상 응답의 해시 및 범위 프라이머리 키를 사용합니다.

```
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"]);
```

앞서 본 예제에서는 WriteLine 메서드를 위해 테이블 값을 문자열로 묵시적으로 변환합니다. DynamoDBEntry 클래스의 다양한 "As[type]" 메서드를 사용하여 명시적인 변환을 할 수 있습니다. 예를 들면 다음과 같이 AsGuid() 메서드를 통해 Id에 대한 값을 Primitive 데이터 형식에서 GUID로 묵시적으로 변환할 수 있습니다.

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

객체 지속성 모델

객체 지속성 프로그래밍 모델은 특히 DynamoDB에서 .NET 객체를 저장, 로드 및 쿼리하기 위해 설계되었습니다. Amazon.DynamoDBv2.DataModel 네임스페이스를 통해 이 모델에 액세스합니다.

세 가지 모델 중에서 객체 지속성 모델은 DynamoDB 데이터를 저장, 로드 또는 쿼리할 때마다 가장 쉽게 코딩할 수 있습니다. 예를 들면 DynamoDB 데이터 형식을 직접 작업할 수 있습니다. 그러나 이 모델에서는 DynamoDB에서 .NET 객체를 저장, 로드 및 쿼리하는 작업에만 액세스할 수 있습니다. 예를 들면 이 모델을 사용하여 테이블의 항목을 생성, 검색, 업데이트 및 삭제할 수 있습니다. 그러나 먼저 하위 수준 모델을 사용하여 테이블을 생성한 후 이 모델을 사용하여 .NET 클래스를 테이블로 매핑해야 합니다.

DynamoDB 객체 지속성 프로그래밍 모델에 대한 자세한 내용은 Amazon DynamoDB 개발자 안내서의 .NET: 객체 지속성 모델을 참조하세요.

다음 예제에서는 DynamoDB 항목을 나타내는 .NET 클래스를 정의하고, 항목을 DynamoDB 테이블에 삽입할 .NET 클래스의 인스턴스를 사용하며, .NET 클래스의 인스턴스를 사용하여 테이블에서 항목을 가져오는 방법을 보여줍니다.

테이블의 항목을 나타내는 .NET 클래스 정의

다음 클래스 정의의 예제에서 `DynamoDBTable` 속성은 테이블 이름을 지정하고, `DynamoDBHashKey` 및 `DynamoDBRangeKey` 속성은 테이블의 해시 및 범위 프라이머리 키를 만듭니다. `DynamoDBGlobalSecondaryIndexHashKey` 속성은 특정 작성자의 답변에 대한 쿼리를 구성할 수 있도록 정의됩니다.

```
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; }
}
```

객체 지속성 모델을 위한 컨텍스트 생성

DynamoDB에 대한 객체 지속성 프로그래밍 모델을 사용하려면 컨텍스트를 생성해야 합니다. 이 컨텍스트는 DynamoDB에 대한 연결을 제공하고, 테이블 액세스, 다양한 작업 수행, 쿼리 실행을 가능하게 합니다.

기본 컨텍스트

다음 예제에서는 가장 기본적인 컨텍스트를 만드는 방법을 보여줍니다.

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

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

DisableFetchingTableMetadata 속성이 포함된 컨텍스트

다음 예제는 `DescribeTable` 메서드에 대한 암시적 호출을 방지하기 위해 `DynamoDBContextConfig` 클래스의 `DisableFetchingTableMetadata` 속성을 추가로 설정하는 방법을 보여줍니다.

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

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

첫 번째 예제와 같이 `DisableFetchingTableMetadata` 속성을 `false`(기본값)로 설정하면 `Reply` 클래스에서 테이블 항목의 키 및 인덱스 구조를 설명하는 속성을 생략할 수 있습니다. 대신 `DescribeTable` 메서드에 대한 암시적 호출을 통해 이러한 속성을 유추합니다. `DisableFetchingTableMetadata`를 `true`로 설정하면 두 번째 예제에서 볼 수 있듯이, `SaveAsync` 및 `QueryAsync`와 같은 객체 지속성 모델의 메서드는 `Reply` 클래스에 정의된 속성에 전적으로 의존합니다. 이 경우 `DescribeTable` 메서드에 대한 호출은 발생하지 않습니다.

> **Note**
>
> 특정 조건에서 `DescribeTable` 메서드에 대한 호출은 콜드 스타트 및 스레드 풀 동작으로 인해 추가 지연 시간이나 교착 상태가 발생할 수 있습니다. 이러한 이유로 해당 메서드에 대한 호출을 피하는 것이 유리한 경우도 있습니다.
> 이러한 동작에 대한 자세한 내용은 AWS SDK for .NET에 대한 향상된 DynamoDB 초기화 패턴 블로그 게시물을 참조하세요.

.NET 클래스의 인스턴스를 사용하여 테이블에 항목을 삽입

이 예제에서는 해당 항목을 나타내는 .NET 클래스의 초기화 인스턴스를 취하는 `SaveAsync` 클래스의 `DynamoDBContext` 메서드를 통해 항목이 삽입됩니다.

```
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"
});
```

.NET 클래스의 인스턴스를 사용하여 테이블에서 항목을 가져오기

이 예제에서는 `DynamoDBContext` 클래스의 `QueryAsync` 메서드를 사용하여 "Author1"의 모든 레코드를 찾는 쿼리를 만듭니다. 그런 다음 쿼리의 `GetNextSetAsync` 메서드를 통해 항목을 검색합니다.

```
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);
}
```

객체 지속성 모델에 대한 추가 정보

위에 표시된 예제와 설명에는 `DisableFetchingTableMetadata`라는 `DynamoDBContext` 클래스의 속성이 포함되는 경우가 있습니다. AWSSDK.DynamoDBv2 NuGet 패키지 버전 3.7.203에 도입된 이 속성을 사용하면 콜드 스타트 및 스레드 풀 동작으로 인해 추가 지연 시간이나 교착 상태가 발생할 수 있는 특정 조건을 피할 수 있습니다. 자세한 내용은 AWS SDK for .NET에 대한 향상된 DynamoDB 초기화 패턴 블로그 게시물을 참조하세요.

다음은 이 속성에 대한 몇 가지 추가 정보입니다.

- .NET Framework를 사용하는 경우 `app.config` 또는 `web.config` 파일에서 이 속성을 전역적으로 설정할 수 있습니다.
- 다음 예제와 같이 `AWSConfigsDynamoDB` 클래스를 사용하여 이 속성을 전역적으로 설정할 수 있습니다.

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

var client = new AmazonDynamoDBClient();
```

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

- 경우에 따라 DynamoDB 속성을 .NET 클래스에 추가할 수 없습니다(예: 클래스가 종속성으로 정의된 경우). 이러한 경우에도 `DisableFetchingTableMetadata` 속성을 계속 활용할 수 있습니다. 이렇게 하려면 `DisableFetchingTableMetadata` 속성과 함께 `TableBuilder` 클래스를 사용합니다. 또한 `TableBuilder` 클래스는 AWSSDK.DynamoDBv2 NuGet 패키지의 버전 3.7.203에 도입되었습니다.

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

추가 정보

AWS SDK for .NET 를 사용하여 DynamoDB 프로그래밍, 정보 및 예제

- DynamoDB API
- DynamoDB 시리즈 시작
- DynamoDB 시리즈 - 문서 모델

- DynamoDB 시리즈 - 변환 스키마
- DynamoDB 시리즈 - 객체 지속성 모델
- DynamoDB 시리즈 - 표현식
- Amazon DynamoDB 및에서 표현식 사용 AWS SDK for .NET
- Amazon DynamoDB의 JSON 지원

하위 레벨 모델 정보 및 예제

- AWS SDK for .NET 하위 수준 API를 사용하여 테이블 작업
- AWS SDK for .NET 하위 수준 API를 사용하여 항목 작업
- AWS SDK for .NET 하위 수준 API를 사용하여 테이블 쿼리
- AWS SDK for .NET 하위 수준 API를 사용하여 테이블 스캔
- AWS SDK for .NET 하위 수준 API를 사용하여 로컬 보조 인덱스 작업
- AWS SDK for .NET 하위 수준 API를 사용하여 글로벌 보조 인덱스 작업

문서 모델, 정보 및 예제

- DynamoDB 데이터 형식
- DynamoDBEntry
- .NET: 문서 모델

객체 지속성 모델 정보 및 예제

- .NET: 객체 지속성 모델

기타 유용한 정보

- .NET Aspire를 통한 Amazon DynamoDB 로컬 개발에 대한 자세한 내용은 .NET Aspire AWS 와 통합 섹션을 참조하세요.

주제

- Amazon DynamoDB 및에서 표현식 사용 AWS SDK for .NET
- Amazon DynamoDB의 JSON 지원

Amazon DynamoDB 및에서 표현식 사용 AWS SDK for .NET

> **Note**
>
> 이 주제의 정보는 .NET Framework 및 AWS SDK for .NET 버전 3.3 이하를 기반으로 하는 프로젝트에만 해당됩니다.

다음 코드 예제에서는를 사용하여 표현식으로 DynamoDB AWS SDK for .NET 를 프로그래밍하는 방법을 보여줍니다. 표현식은 DynamoDB 테이블의 항목에서 읽을 속성을 나타냅니다. 또한 항목을 쓸 때도 표현식을 사용하여 충족해야 할 조건(조건부 업데이트라고도 함)을 나타내고 속성을 업데이트하는 방식을 나타낼 수 있습니다. 일부 업데이트 예에서는 속성을 새 값으로 바꾸거나 새 데이터를 목록이나 맵에 추가합니다. 자세한 내용은 표현식을 사용하여 항목 읽기 및 쓰기를 참조하십시오.

주제

- 샘플 데이터
- 표현식 및 항목의 기본 키를 사용한 단일 항목 조회
- 표현식 및 테이블의 기본 키를 사용한 다수 항목 조회
- 표현식 및 기타 항목 속성을 사용한 다수 항목 조회
- 항목 출력
- 표현식을 사용한 항목 생성 및 교체
- 표현식을 사용한 항목 업데이트
- 표현식을 사용한 항목 삭제
- 추가 정보

샘플 데이터

이 주제의 코드 예제는 `ProductCatalog`라는 DynamoDB 테이블에 있는 다음 두 가지 예제 항목에 의존합니다. 이 항목들은 가상의 자전거 점포 카탈로그에 있는 제품 항목에 대한 정보를 알려줍니다. 이러한 항목은 사례 연구: ProductCatalog 항목에서 제공하는 예제를 기반으로 합니다. `BOOL`, `L`, `M`, `N`, `NS`, `S`, `SS`과 같은 데이터 형식 서술자는 JSON 데이터 형식의 서술자와 일치합니다.

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

표현식 및 항목의 기본 키를 사용한 단일 항목 조회

다음 예제에는 Amazon.DynamoDBv2.AmazonDynamoDBClient.GetItem의 Id가 있는 항목을 조회하여 출력하는 데 사용할 205 메서드와 일련의 표현식이 포함되어 있습니다. 항목 속성 중에서 Id, Title, Description, Color, RelatedItems, Pictures, ProductReviews 속성만 반환됩니다.

```
// 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);
```

앞서 다룬 예제에서 ProjectionExpression 속성은 반환될 속성을 지정합니다. ExpressionAttributeNames 속성에서는 #pr 속성을 나타낼 자리표시자 ProductReviews과 #ri 속성을 나타낼 자리표시자 RelatedItems를 지정합니다. PrintItem에 대한 호출은 항목 인쇄의 설명과 같이 사용자 지정 함수를 참조합니다.

표현식 및 테이블의 기본 키를 사용한 다수 항목 조회

다음 예제에는 Amazon.DynamoDBv2.AmazonDynamoDBClient.Query의 값이 Id보다 큰 경우에 한해 301의 Price가 있는 항목을 조회하여 출력하는 데 사용할 150 메서드와 일련의 표현식이 포함되어 있습니다. 항목의 속성 중에서 반환되는 속성은 Id와 Title, 그리고 ThreeStar의 모든 ProductReviews 속성뿐입니다.

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

앞서 다룬 예제에서 ProjectionExpression 속성은 반환될 속성을 지정합니다. ExpressionAttributeNames 속성은 ProductReviews 속성을 나타내는 자리 표시자 #pr 및 Price 속성을 나타내는 자리 표시자 #p를 지정합니다. #pr.ThreeStar는 ThreeStar 속성만 반환하도록 지정합니다. ExpressionAttributeValues 속성에서는 :val이라는 값을 나타낼 자리표시자 150을 지정합니다. FilterExpression 속성은 #p(Price)가 :val(150)보다 커야 한다는 조건을 지정합니다. PrintItem에 대한 호출은 항목 인쇄의 설명과 같이 사용자 지정 함수를 참조합니다.

표현식 및 기타 항목 속성을 사용한 다수 항목 조회

다음 예제에는 Amazon.DynamoDBv2.AmazonDynamoDBClient.Scan의 ProductCategory가 있는 항목을 모두 조회하여 출력하는 데 사용할 Bike 메서드와 일련의 표현식이 포함되어 있습니다. 항목의 속성 중에서 반환되는 속성은 Id와 Title, 그리고 ProductReviews의 모든 속성뿐입니다.

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

앞서 다룬 예제에서 ProjectionExpression 속성은 반환될 속성을 지정합니다. ExpressionAttributeNames 속성에서는 #pr 속성을 나타낼 자리표시자 ProductReviews과 #pc 속성을 나타낼 자리표시자 ProductCategory를 지정합니다. ExpressionAttributeValues 속성에서는 :catg이라는 값을 나타낼 자리표시자 Bike을 지정합니다. FilterExpression 속성에서는 #pc(ProductCategory)가 :catg(Bike)와 같아야 한다는 조건을 지정합니다. PrintItem에 대한 호출은 항목 인쇄의 설명과 같이 사용자 지정 함수를 참조합니다.

항목 출력

다음 예제에서는 항목의 속성과 값을 출력하는 방법을 보여줍니다. 이 예제는 표현식 및 항목의 기본 키를 사용하여 단일 항목 가져오기, 표현식 및 테이블의 기본 키를 사용하여 다중 항목 가져오기, 표현식 및 기타 항목 속성을 사용하여 다중 항목 가져오기 방법을 보여주는 앞의 예제에서 사용됩니다.

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

앞서 다룬 예제에서는 속성을 출력할 정확한 형식을 결정하기 위해 평가할 수 있는 몇 가지 데이터 형식별 속성이 각 속성 값에 있습니다. 이러한 속성에는 JSON 데이터 형식에 해당하는 B, BOOL, BS, L, M, N, NS, NULL, S 및 SS가 포함됩니다. B, N, NULL, S와 같은 속성의 경우에는 상응하는 속성이 null이 아니라면 그 속성은 상응하는 null 아닌 데이터 형식에 대한 것입니다. BS, L, M, NS, SS와 같

은 속성의 경우에는 Count 값이 0보다 크다면 그 속성은 상응하는 0 아닌 값 데이터 형식에 대한 것입니다. 그 속성의 데이터 형식별 속성 전체가 null이거나 Count 값이 0인 경우에 그 속성은 BOOL 데이터 형식에 상응하는 것입니다.

표현식을 사용한 항목 생성 및 교체

다음 예제에는 Amazon.DynamoDBv2.AmazonDynamoDBClient.PutItem의 Title이 있는 항목을 업데이트하는 데 사용할 18-Bicycle 301 메서드와 일련의 표현식이 포함되어 있습니다. 항목이 아직 없으면 새 항목이 추가됩니다.

```
// 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);
```

앞서 다룬 예제에서 ExpressionAttributeNames 속성은 #title 속성을 나타낼 자리표시자 Title을 지정합니다. ExpressionAttributeValues 속성에서는 :product이라는 값을 나타낼 자리표시자 18-Bicycle 301을 지정합니다. ConditionExpression 속성에서는 #title(Title)가 :product(18-Bicycle 301)와 같아야 한다는 조건을 지정합니다. CreateItemData에 대한 호출은 다음과 같은 사용자 지정 함수를 참조합니다.

```
// 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;
}
```

앞서 다룬 예제에서 샘플 데이터가 있는 예제 항목은 호출자에게 반환됩니다. 일련의 속성과 해당 값은 JSON 데이터 형식에 해당하는 BOOL, L, M, N, NS, S 및 SS 등의 데이터 유형을 사용하여 구성됩니다.

표현식을 사용한 항목 업데이트

다음 예제에는 Amazon.DynamoDBv2.AmazonDynamoDBClient.UpdateItem의 Title가 있는 항목에 대해 18" Girl's Bike을 Id로 변경하는 데 사용할 301 메서드와 일련의 표현식이 포함되어 있습니다.

```
// 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);
```

앞서 다룬 예제에서 ExpressionAttributeNames 속성은 #title 속성을 나타낼 자리표시자 Title을 지정합니다. ExpressionAttributeValues 속성에서는 :newproduct이라는 값을 나타낼 자리표시자 18" Girl's Bike을 지정합니다. UpdateExpression 속성에서는 #title(Title)을 :newproduct(18" Girl's Bike)로 변경하도록 지정합니다.

표현식을 사용한 항목 삭제

다음 예제에는 항목의 Title이 18-Bicycle 301인 경우에 한해 Id가 301인 항목을 삭제하는 데 사용할 Amazon.DynamoDBv2.AmazonDynamoDBClient.DeleteItem 메서드와 일련의 표현식이 포함되어 있습니다.

```
// 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);
```

앞서 다룬 예제에서 `ExpressionAttributeNames` 속성은 `#title` 속성을 나타낼 자리표시자 `Title`을 지정합니다. `ExpressionAttributeValues` 속성에서는 `:product`이라는 값을 나타낼 자리표시자 `18-Bicycle 301`을 지정합니다. `ConditionExpression` 속성에서는 `#title`(`Title`)이 `:product`(`18-Bicycle 301`)와 같아야 한다는 조건을 지정합니다.

추가 정보

자세한 내용 및 코드 예제는 다음을 참조하십시오.

- DynamoDB 시리즈 - 표현식
- 프로젝션 식을 사용하여 항목 속성 액세스
- 속성 이름 및 값에 자리 표시자 사용
- 조건식을 사용하여한 조건 지정
- 업데이트 식을 사용하여 항목 및 속성 수정
- AWS SDK for .NET 하위 수준 API를 사용하여 항목 작업
- AWS SDK for .NET 하위 수준 API를 사용하여 테이블 쿼리
- AWS SDK for .NET 하위 수준 API를 사용하여 테이블 스캔
- AWS SDK for .NET 하위 수준 API를 사용하여 로컬 보조 인덱스 작업

- AWS SDK for .NET 하위 수준 API를 사용하여 글로벌 보조 인덱스 작업

Amazon DynamoDB의 JSON 지원

> **Note**
>
> 이 주제의 정보는 .NET Framework 및 AWS SDK for .NET 버전 3.3 이하를 기반으로 하는 프로젝트에만 해당됩니다.

는 Amazon DynamoDB로 작업할 때 JSON 데이터를 AWS SDK for .NET 지원합니다. 따라서 DynamoDB 테이블에서 JSON 형식 데이터를 더 쉽게 가져올 수 있으며 JSON 문서를 테이블에 더 쉽게 삽입할 수 있습니다.

주제

- DynamoDB 테이블에서 JSON 형식으로 데이터 가져오기
- DynamoDB 테이블에 JSON 형식 데이터 삽입
- JSON으로 DynamoDB 데이터 형식 변환
- 추가 정보

DynamoDB 테이블에서 JSON 형식으로 데이터 가져오기

다음 예제에서는 DynamoDB 테이블에서 JSON 형식으로 데이터를 가져오는 방법을 보여줍니다.

```
// 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
//   }
```

앞의 예제에서는 Document 클래스의 ToJson 메서드가 테이블의 항목을 JSON 형식 문자열로 변환합니다. 해당 항목은 Table 클래스의 GetItem 메서드를 통해 가져옵니다. 가져올 항목을 결정하기 위해 이 예제에서는 GetItem 메서드가 대상 항목의 해시 및 범위 기본 키를 사용합니다. 항목을 가져올 테이블을 결정하기 위해 Table 클래스의 LoadTable 메서드는 DynamoDB에서 AmazonDynamoDBClient 클래스의 인스턴스와 대상 테이블의 이름을 사용합니다.

DynamoDB 테이블에 JSON 형식 데이터 삽입

다음 예제에서는 JSON 형식을 사용하여 DynamoDB 테이블에 항목을 삽입하는 방법을 보여줍니다.

```
// 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);
```

앞의 예제에서는 FromJson 클래스의 Document 메서드가 JSON 형식 문자열을 항목으로 변환합니다. 해당 항목은 이 항목을 포함하는 PutItem 클래스의 인스턴스를 사용하는 Table 클래스의 Document 메서드를 통해 테이블에 삽입됩니다. 항목을 삽입할 테이블을 결정하기 위해 Table 클래스의 LoadTable 메서드가 호출되어 DynamoDB에서 AmazonDynamoDBClient 클래스의 인스턴스와 대상 테이블의 이름을 지정합니다.

JSON으로 DynamoDB 데이터 형식 변환

Document 클래스의 ToJson 메서드를 호출한 다음 그 결과로 얻은 JSON 데이터에서 FromJson 메서드를 호출하여 JSON 데이터를 Document 클래스의 인스턴스로 다시 변환할 때마다 일부 DynamoDB 데이터 형식은 예상대로 변환되지 않습니다. 구체적으로 설명하면 다음과 같습니다.

- DynamoDB 세트(SS, NS 및 BS 유형)는 JSON 어레이로 변환됩니다.

- DynamoDB 이진수 스칼라 및 세트(B 및 BS 형식)는 base64로 인코딩된 JSON 문자열 또는 문자열 목록으로 변환됩니다.

  이 시나리오에서는 `DecodeBase64Attributes` 클래스의 `Document` 메서드를 호출하여 base64로 인코딩된 JSON 데이터를 정확한 이진수 표시로 바꿔야 합니다. 다음 예제에서는 `Document`라는 `Picture` 클래스의 인스턴스에 있는 base64로 인코딩된 이진수 스칼라 항목 속성을 정확한 이진수 표시로 바꿉니다. 또한 이 예제에서는 다음과 같이 `Document`라는 `RelatedPictures` 클래스의 동일한 인스턴스에 있는 base64로 인코딩된 이진수 세트 항목 속성에 대해 동일한 작업을 합니다.

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

추가 정보

를 사용하여 DynamoDB로 JSON을 프로그래밍하는 방법에 대한 자세한 내용과 예제는 다음을 AWS SDK for .NET참조하세요.

- DynamoDB JSON 지원
- Amazon DynamoDB 업데이트 - JSON, 확장 프리 티어, 유연한 조정, 대규모 항목

Amazon EC2 작업

는 크기 조정 가능한 컴퓨팅 용량을 제공하는 웹 서비스인 Amazon EC2를 AWS SDK for .NET 지원합니다. 이 컴퓨팅 용량을 사용하여 소프트웨어 시스템을 구축하고 호스팅할 수 있습니다.

API

는 Amazon EC2 클라이언트용 APIs AWS SDK for .NET 제공합니다. API를 사용하면 보안 그룹 및 키 페어와 같은 EC2 기능을 사용할 수 있습니다. 또한 API를 사용하여 Amazon EC2 인스턴스를 제어할 수 있습니다. 이 섹션에는 이러한 API로 작업할 때 따를 수 있는 패턴을 보여주는 몇 가지 예제가 포함되어 있습니다. 전체 API 세트를 보려면 AWS SDK for .NET API 참조를 참조하세요(그리고 "Amazon.EC2"으로 스크롤하세요).

Amazon EC2 API는 AWSSDK.EC2 NuGet 패키지를 통해 제공됩니다.

사전 조건

시작하기 전에 환경을 설정하고 프로젝트를 구성했는지 확인합니다. 또한 SDK 사용의 정보를 검토합니다.

예시 관련 정보

이 섹션의 예시에서는 Amazon EC2 클라이언트 작업 방법과 Amazon EC2 인스턴스를 관리하는 방법을 보여줍니다.

EC2 스팟 인스턴스 자습서에서는 Amazon EC2 스팟 인스턴스를 요청하는 방법을 보여줍니다. 스팟 인스턴스를 사용하면 온디맨드 가격보다 저렴한 비용으로 미사용 EC2 용량에 액세스할 수 있습니다.

주제

- Amazon EC2 에서 보안 그룹 작업
- Amazon EC2 키 페어로 작업
- Amazon EC2 리전 및 가용 영역 표시
- Amazon EC2 인스턴스 작업
- Amazon EC2 스팟 인스턴스 자습서

Amazon EC2 에서 보안 그룹 작업

Amazon EC2에서 보안 그룹은 하나 이상의 EC2 인스턴스에 대한 네트워크 트래픽을 제어하는 가상 방화벽 역할을 합니다. 기본적으로 EC2는 인바운드 트래픽을 허용하지 않는 보안 그룹과 인스턴스를 연결합니다. EC2 인스턴스가 특정 트래픽을 받아들이도록 허용하는 보안 그룹을 생성할 수 있습니다. 예를 들어 EC2 Windows 인스턴스에 연결해야 하는 경우 보안 그룹이 RDP 트래픽을 허용하도록 구성해야 합니다.

보안 그룹에 대한 자세한 내용은 Amazon EC2 사용 설명서의 Amazon EC2 보안 그룹을 참조하세요.

> **Warning**
>
> EC2-Classic은 2022년 8월 15일에 사용 중지되었습니다. EC2-Classic에서 VPC로 마이그레이션하는 것이 좋습니다. 자세한 내용은 블로그 게시물 EC2-Classic 네트워킹은 사용 중지 중입니다 - 준비 방법은 다음과 같습니다를 참조하세요.

API 및 사전 조건에 대한 자세한 내용은 상위 섹션(Amazon EC2 작업)을 참조하세요.

주제

- 보안 그룹 열거

- 보안 그룹 생성
- 보안 그룹 업데이트

보안 그룹 열거

이 예제에서는를 사용하여 보안 그룹을 열거 AWS SDK for .NET 하는 방법을 보여줍니다. Amazon Virtual Private Cloud ID를 제공하는 경우 애플리케이션은 해당 VPC의 보안 그룹을 열거합니다. 그렇지 않으면 애플리케이션은 사용 가능한 모든 보안 그룹 목록만 표시합니다.

다음 섹션에서는 이 예제의 코드 조각을 제공합니다. 예제의 전체 코드는 그 뒤에 표시되며, 그대로 빌드하고 실행할 수 있습니다.

주제

- 보안 그룹 열거
- 전체 코드
- 추가 고려 사항

보안 그룹 열거

다음 코드 조각은 보안 그룹을 열거합니다. 모든 그룹을 열거하거나 특정 VPC의 그룹이 있는 경우 해당 그룹을 열거합니다.

이 주제의 끝 부분에 있는 예제에서는 사용 중인 이 코드 조각을 보여줍니다.

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

전체 코드

이 섹션에는 이 예제에 대한 관련 참조와 전체 코드가 나와 있습니다.

SDK 레퍼런스

NuGet 패키지:

- AWSSDK.EC2

프로그래밍 요소:

- 네임스페이스 Amazon.EC2

  클래스 AmazonEC2Client
- 네임스페이스 Amazon.EC2.Model

  클래스 DescribeSecurityGroupsRequest

  클래스 DescribeSecurityGroupsResponse

  클래스 Filter

  클래스 SecurityGroup

코드

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

추가 고려 사항

- VPC의 경우 이름-값 쌍의 Name 부분이 "vpc-id"로 설정된 상태로 필터가 구성된다는 점에 유의하세요. 이 이름은 DescribeSecurityGroupsRequest 클래스의 Filters 속성 설명에서 가져온 것입니다.

- 전체 보안 그룹 목록을 가져오려면 파라미터 없이 DescribeSecurityGroupsAsync를 사용할 수도 있습니다.

- Amazon EC2 콘솔에서 보안 그룹 목록을 확인하여 결과를 확인할 수 있습니다.

보안 그룹 생성

이 예제에서는를 사용하여 보안 그룹을 AWS SDK for .NET 생성하는 방법을 보여줍니다. 기존 VPC의 ID를 제공하여 VPC의 EC2에 대한 보안 그룹을 생성할 수 있습니다. 이러한 ID를 제공하지 않으면 AWS 계정이 이를 지원하는 경우 새 보안 그룹은 EC2-Classic용가 됩니다.

VPC ID를 제공하지 않고 AWS 계정이 EC2-Classic을 지원하지 않는 경우 새 보안 그룹은 계정의 기본 VPC에 속합니다.

> **Warning**
>
> EC2-Classic은 2022년 8월 15일에 사용 중지되었습니다. EC2-Classic에서 VPC로 마이그레이션하는 것이 좋습니다. 자세한 내용은 블로그 게시물 EC2-Classic 네트워킹은 사용 중지 중입니다 - 준비 방법은 다음과 같습니다를 참조하세요.

다음 섹션에서는 이 예제의 코드 조각을 제공합니다. 예제의 전체 코드는 그 뒤에 표시되며, 그대로 빌드하고 실행할 수 있습니다.

주제

- 기존 보안 그룹 찾기
- 보안 그룹 생성
- 전체 코드

기존 보안 그룹 찾기

다음 코드 조각은 지정된 VPC에서 지정된 이름을 가진 기존 보안 그룹을 검색합니다.

이 주제의 끝 부분에 있는 예제에서는 사용 중인 이 코드 조각을 보여줍니다.

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

보안 그룹 생성

다음 코드 조각은 해당 이름을 가진 그룹이 지정된 VPC에 없는 경우 새 보안 그룹을 생성합니다. VPC가 제공되지 않고 같은 이름을 가진 그룹이 하나 이상 존재할 경우 코드 조각은 단순히 그룹 목록을 반환합니다.

이 주제의 끝 부분에 있는 예제에서는 사용 중인 이 코드 조각을 보여줍니다.

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

전체 코드

이 섹션에는 이 예제에 대한 관련 참조와 전체 코드가 나와 있습니다.

SDK 레퍼런스

NuGet 패키지:

- AWSSDK.EC2

프로그래밍 요소:

- 네임스페이스 Amazon.EC2

  클래스 AmazonEC2Client
- 네임스페이스 Amazon.EC2.Model

  클래스 CreateSecurityGroupRequest

  클래스 CreateSecurityGroupResponse

클래스 DescribeSecurityGroupsRequest

클래스 DescribeSecurityGroupsResponse

필터 Filter

클래스 SecurityGroup

코드

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

namespace EC2CreateSecGroup
{
  // = = =
```

```
        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);
```

```
    }
  }

}
```

보안 그룹 업데이트

이 예제에서는 AWS SDK for .NET 를 사용하여 보안 그룹에 규칙을 추가하는 방법을 보여줍니다. 특히 이 예제에서는 지정된 TCP 포트에서 인바운드 트래픽을 허용하는 규칙을 추가합니다. 이 규칙은 예를 들어 EC2 인스턴스에 대한 원격 연결에 사용할 수 있습니다. 애플리케이션은 기존 보안 그룹의 ID, CIDR 형식의 IP 주소(또는 주소 범위), 그리고 선택적으로 TCP 포트 번호를 사용합니다. 그런 다음 지정된 보안 그룹에 인바운드 규칙을 추가합니다.

> **Note**
>
> 이 예제를 사용하려면 CIDR 형식의 IP 주소(또는 주소 범위)가 필요합니다. 로컬 컴퓨터의 IP 주소를 얻는 방법은 이 주제의 끝 부분에 있는 추가 고려 사항을 참조하세요.

다음 섹션에서는 이 예제의 코드 조각을 제공합니다. 예제의 전체 코드는 그 뒤에 표시되며, 그대로 빌드하고 실행할 수 있습니다.

주제

- 인바운드 규칙 추가
- 전체 코드
- 추가 고려 사항

인바운드 규칙 추가

다음 코드 조각은 특정 IP 주소(또는 범위) 및 TCP 포트에 대한 보안 그룹에 인바운드 규칙을 추가합니다.

이 주제의 끝 부분에 있는 예제에서는 사용 중인 이 코드 조각을 보여줍니다.

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

전체 코드

이 섹션에는 이 예제에 대한 관련 참조와 전체 코드가 나와 있습니다.

SDK 레퍼런스

NuGet 패키지:

- AWSSDK.EC2

프로그래밍 요소:

- 네임스페이스 Amazon.EC2

  클래스 AmazonEC2Client
- 네임스페이스 Amazon.EC2.Model

  클래스 AuthorizeSecurityGroupIngressRequest

  클래스 AuthorizeSecurityGroupIngressResponse

  클래스 IpPermission

  클래스 IpRange

코드

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

namespace EC2AddRuleForRDP
{
  // = = =
```

```
        new AmazonEC2Client(), groupID, ipAddress, int.Parse(portStr));
    }


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

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


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


  // = = =
```

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

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

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

```
      return parsedArgs;
    }

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

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

}
```

추가 고려 사항

- 포트 번호를 제공하지 않는 경우 애플리케이션은 기본적으로 포트 3389로 설정됩니다. 이 포트는 Windows RDP용 포트로, Windows를 실행하는 EC2 인스턴스에 연결할 수 있습니다. Linux를 실행하는 EC2 인스턴스를 시작하는 경우에는 그 대신에 TCP 포트 22(SSH)를 사용하세요.

- 참고로 이 예제에서는 `IpProtocol`이 “tcp”로 설정되어 있습니다. `IpProtocol`의 값은 IpPermission 클래스의 `IpProtocol` 속성에 대한 설명에서 찾을 수 있습니다.

- 이 예를 사용할 때 로컬 컴퓨터의 IP 주소가 필요할 수 있습니다. 주소를 확인할 수 있는 몇 가지 방법은 다음과 같습니다.
  - EC2 인스턴스에 연결할 로컬 컴퓨터에 정적 퍼블릭 IP 주소가 있는 경우 서비스를 사용하여 해당 주소를 가져올 수 있습니다. 이러한 서비스 중 하나가 http://checkip.amazonaws.com/입니다. 인바운드 트래픽 권한 부여에 대한 자세한 내용은 Amazon EC2 사용 설명서의 보안 그룹에 규칙 추가 및 다양한 사용 사례에 대한 보안 그룹 규칙을 참조하세요.
  - 로컬 컴퓨터의 IP 주소를 얻는 또 다른 방법은 Amazon EC2 콘솔을 사용하는 것입니다.

    보안 그룹 중 하나를 선택하고 인바운드 규칙 탭을 선택한 다음 인바운드 규칙 편집을 선택합니다. 인바운드 규칙에서 소스 열의 드롭다운 메뉴를 열고 내 IP를 선택하면 로컬 컴퓨터의 IP 주소가 CIDR 형식으로 표시됩니다. 작업을 취소해야 합니다.

- Amazon EC2 콘솔에서 보안 그룹 목록을 검사하여 이 예제의 결과를 확인할 수 있습니다.

Amazon EC2 키 페어로 작업

Amazon EC2는 퍼블릭 키 암호화 기법을 사용하여 로그인 정보를 암호화 및 해독합니다. 퍼블릭 키 암호화 기법에서는 퍼블릭 키를 사용하여 데이터를 암호화한 후 수신자가 개인 키를 사용하여 해당 데이터를 해독합니다. 퍼블릭 키와 프라이빗 키를 키 페어라고 합니다. EC2 인스턴스에 로그인하려면 시작 시 키 페어를 지정한 다음 연결할 때 키 페어의 프라이빗 키를 제공해야 합니다.

EC2 인스턴스를 시작할 때 해당 인스턴스에 대해 키 페어를 생성하거나 다른 인스턴스를 시작할 때 이미 사용한 키 페어를 사용할 수 있습니다. Amazon EC2 키 페어에 대한 자세한 내용은 Amazon EC2 사용 설명서의 Amazon EC2 키 페어 작업을 참조하세요.

API 및 사전 조건에 대한 자세한 내용은 상위 섹션(Amazon EC2 작업)을 참조하세요.

주제

- 키 페어 생성 및 표시
- 키 페어 삭제

키 페어 생성 및 표시

이 예제에서는를 사용하여 키 페어를 AWS SDK for .NET 생성하는 방법을 보여줍니다. 애플리케이션은 새 키 페어의 이름과 PEM 파일(“.pem” 확장명)의 이름을 사용합니다. 키 페어를 생성하고 PEM 파일에 프라이빗 키를 쓴 다음 사용 가능한 모든 키 페어를 표시합니다. 명령줄 인수를 제공하지 않으면 애플리케이션은 사용 가능한 모든 키 페어만 표시합니다.

다음 섹션에서는 이 예제의 코드 조각을 제공합니다. 예제의 전체 코드는 그 뒤에 표시되며, 그대로 빌드하고 실행할 수 있습니다.

주제

- 키 페어 생성
- 사용 가능한 키 페어 표시
- 전체 코드
- 추가 고려 사항

키 페어 생성

다음 코드 조각은 키 페어를 생성한 다음 지정된 PEM 파일에 프라이빗 키를 저장합니다.

이 주제의 끝 부분에 있는 예제에서는 사용 중인 이 코드 조각을 보여줍니다.

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

사용 가능한 키 페어 표시

다음 코드 조각은 사용 가능한 키 페어의 목록을 표시합니다.

이 주제의 끝 부분에 있는 예제에서는 사용 중인 이 코드 조각을 보여줍니다.

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

전체 코드

이 섹션에는 이 예제에 대한 관련 참조와 전체 코드가 나와 있습니다.

SDK 레퍼런스

NuGet 패키지:

- AWSSDK.EC2

프로그래밍 요소:

- 네임스페이스 Amazon.EC2

  클래스 AmazonEC2Client
- 네임스페이스 Amazon.EC2.Model

  클래스 CreateKeyPairRequest

  클래스 CreateKeyPairResponse

  클래스 DescribeKeyPairsResponse

  클래스 KeyPairInfo

코드

```
using System;
using System.Threading.Tasks;
using System.IO;
using Amazon.EC2;
```

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

namespace EC2CreateKeyPair
{
  // = = =
```

```
    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.");
    }
 }


 // = = =
```

```
        // 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);
    }
  }

}
```

추가 고려 사항

- 예제를 실행한 후 Amazon EC2 콘솔에서 새 키 페어를 확인할 수 있습니다.
- 나중에 프라이빗 키를 검색할 수 없으므로 키 페어를 생성할 때는 반환되는 프라이빗 키를 저장해야 합니다.

키 페어 삭제

이 예제에서는를 사용하여 키 페어 AWS SDK for .NET 를 삭제하는 방법을 보여줍니다. 이 애플리케이션은 키 페어 이름을 사용합니다. 키 페어를 삭제한 다음 사용 가능한 모든 키 페어를 표시합니다. 명령줄 인수를 제공하지 않으면 애플리케이션은 사용 가능한 모든 키 페어만 표시합니다.

다음 섹션에서는 이 예제의 코드 조각을 제공합니다. 예제의 전체 코드는 그 뒤에 표시되며, 그대로 빌드하고 실행할 수 있습니다.

주제

- 키 페어 삭제
- 사용 가능한 키 페어 표시
- 전체 코드

키 페어 삭제

다음 코드 조각은 키 페어를 삭제합니다.

이 주제의 끝 부분에 있는 예제에서는 사용 중인 이 코드 조각을 보여줍니다.

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

사용 가능한 키 페어 표시

다음 코드 조각은 사용 가능한 키 페어의 목록을 표시합니다.

이 주제의 끝 부분에 있는 예제에서는 사용 중인 이 코드 조각을 보여줍니다.

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

전체 코드

이 섹션에는 이 예제에 대한 관련 참조와 전체 코드가 나와 있습니다.

SDK 레퍼런스

NuGet 패키지:

- AWSSDK.EC2

프로그래밍 요소:

- 네임스페이스 Amazon.EC2

  클래스 AmazonEC2Client
- 네임스페이스 Amazon.EC2.Model

  클래스 DeleteKeyPairRequest

  클래스 DescribeKeyPairsResponse

  클래스 KeyPairInfo

코드

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

Amazon EC2 리전 및 가용 영역 표시

Amazon EC2는 전 세계의 여러 곳에서 호스팅되고 있습니다. 해당 위치는 리전 및 가용 영역으로 구성됩니다. 각 리전은 지리적 개별 영역이며, 가용 영역이라고 알려진 여러 개의 격리된 위치가 있습니다.

리전 및 가용 영역에 대한 자세한 내용은 Amazon EC2 사용 설명서의 리전 및 가용 영역을 참조하세요.

이 예제에서는 AWS SDK for .NET 를 사용하여 EC2 클라이언트와 관련된 리전 및 가용 영역에 대한 세부 정보를 가져오는 방법을 보여줍니다. 애플리케이션은 EC2 클라이언트가 사용할 수 있는 리전 및 가용 영역 목록을 표시합니다.

SDK 레퍼런스

NuGet 패키지:

- AWSSDK.EC2

프로그래밍 요소:

- 네임스페이스 Amazon.EC2

  클래스 AmazonEC2Client
- 네임스페이스 Amazon.EC2.Model

  클래스 DescribeAvailabilityZonesResponse

  클래스 DescribeRegionsResponse

  클래스 AvailabilityZone

  클래스 Region

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

namespace EC2RegionsAndZones
{
  class Program
  {
    static async Task Main(string[] args)
    {
      Console.WriteLine(
        "Finding the Regions and Availability Zones available to an EC2 client...");

      // Create the EC2 client
      var ec2Client = new AmazonEC2Client();

      // Display the Regions and Availability Zones
      await DescribeRegions(ec2Client);
      await DescribeAvailabilityZones(ec2Client);
    }


    //
    // Method to display Regions
    private static async Task DescribeRegions(IAmazonEC2 ec2Client)
    {
      Console.WriteLine("\nRegions that are enabled for the EC2 client:");
      DescribeRegionsResponse response = await ec2Client.DescribeRegionsAsync();
      foreach (Region region in response.Regions)
        Console.WriteLine(region.RegionName);
    }


    //
    // Method to display Availability Zones
    private static async Task DescribeAvailabilityZones(IAmazonEC2 ec2Client)
    {
      Console.WriteLine("\nAvailability Zones for the EC2 client's region:");
      DescribeAvailabilityZonesResponse response =
        await ec2Client.DescribeAvailabilityZonesAsync();
      foreach (AvailabilityZone az in response.AvailabilityZones)
        Console.WriteLine(az.ZoneName);
```

```
    }
  }
}
```

Amazon EC2 인스턴스 작업

AWS SDK for .NET 를 사용하여 생성, 시작 및 종료와 같은 작업을 통해 Amazon EC2 인스턴스를 제어할 수 있습니다. 이 섹션의 주제에서는 이를 수행하는 방법의 몇 가지 예제를 제공합니다. EC2 인스턴스에 대한 자세한 내용은 Amazon EC2 사용 설명서의 Amazon EC2 인스턴스를 참조하세요.

API 및 사전 조건에 대한 자세한 내용은 상위 섹션(Amazon EC2 작업)을 참조하세요.

주제

- Amazon EC2 인스턴스 시작
- Amazon EC2 인스턴스 종료

Amazon EC2 인스턴스 시작

이 예제에서는 AWS SDK for .NET 를 사용하여 동일한 Amazon Machine Image(AMI)에서 하나 이상의 동일하게 구성된 Amazon EC2 인스턴스를 시작하는 방법을 보여줍니다. 애플리케이션은 사용자가 제공한 여러 입력을 사용하여 EC2 인스턴스를 시작한 다음 “Pending” 상태가 아닐 때까지 인스턴스를 모니터링합니다.

(선택 사항) 인스턴스에 연결에 설명된 대로 EC2 인스턴스가 실행될 때 EC2 인스턴스에 원격으로 연결할 수 있습니다.

> **Warning**
>
> EC2-Classic은 2022년 8월 15일에 사용 중지되었습니다. EC2-Classic에서 VPC로 마이그레이션하는 것이 좋습니다. 자세한 내용은 블로그 게시물 EC2-Classic 네트워킹은 사용 중지 중입니다 - 준비 방법은 다음과 같습니다를 참조하세요.

다음 섹션에서는 이 예제의 코드 조각과 기타 정보를 제공합니다. 예제의 전체 코드는 코드 조각 뒤에 표시되며, 그대로 빌드하고 실행할 수 있습니다.

주제

- 필요한 내용 수집
- 인스턴스 시작

- 인스턴스 모니터링
- 전체 코드
- 추가 고려 사항
- (선택 사항) 인스턴스에 연결
- 정리

필요한 내용 수집

EC2 인스턴스를 시작하려면 몇 가지 사항이 필요합니다.

- 인스턴스가 시작될 VPC입니다. Windows 인스턴스이고 RDP를 통해 연결하려는 경우 VPC에 인터넷 게이트웨이가 연결되어 있어야 할 가능성이 높으며 라우팅 테이블의 인터넷 게이트웨이에 대한 항목도 있어야 합니다. 자세한 내용은 Amazon VPC 사용 설명서의 인터넷 게이트웨이를 참조하세요.

- 인스턴스가 시작될 VPC의 기존 서브넷 ID입니다. Amazon VPC 콘솔에 로그인하는 것도 이를 찾거나 생성하는 쉬운 방법이지만, CreateSubnetAsync 및 DescribeSubnetsAsync 메서드를 사용하여 프로그래밍 방식으로 가져올 수도 있습니다.

  > **Note**
  >
  > 이 파라미터를 제공하지 않으면 계정의 기본 VPC에서 새 인스턴스가 시작됩니다.

- 인스턴스가 시작될 VPC에 속하는 기존 보안 그룹의 ID입니다. 자세한 내용은 Amazon EC2 에서 보안 그룹 작업 단원을 참조하십시오.

- 새 인스턴스에 연결하려면 앞서 언급한 보안 그룹에 포트 22에서의 SSH 트래픽(Linux 인스턴스) 또는 포트 3389에서의 RDP 트래픽(Windows 인스턴스)을 허용하는 적절한 인바운드 규칙이 있어야 합니다. 이를 수행하는 방법은 해당 주제의 끝 부분에 있는 추가 고려 사항을 포함하여 보안 그룹 업데이트을 참조하세요.

- 인스턴스를 생성하는 데 사용되는 Amazon Machine Image(AMI)입니다. AMI에 대한 자세한 내용은 Amazon EC2 사용 설명서의 Amazon Machine Image(AMI)를 참조하세요. 특히 AMI 찾기 및 공유 AMI를 참조하세요.

- 새 인스턴스에 연결하는 데 사용되는 기존 EC2 키 페어의 이름입니다. 자세한 내용은 Amazon EC2 키 페어로 작업 단원을 참조하십시오.

- 앞서 언급한 EC2 키 페어의 프라이빗 키가 포함된 PEM 파일의 이름입니다. PEM 파일은 인스턴스에 원격으로 연결할 때 사용됩니다.

인스턴스 시작

다음 코드 조각은 EC2 인스턴스를 시작합니다.

이 주제의 끝 부분에 있는 예제에서는 사용 중인 이 코드 조각을 보여줍니다.

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

인스턴스 모니터링

다음 코드 조각은 “Pending” 상태가 아닐 때까지 인스턴스를 모니터링합니다.

이 주제의 끝 부분에 있는 예제에서는 사용 중인 이 코드 조각을 보여줍니다.

`Instance.State.Code` 속성의 유효한 값은 InstanceState 클래스를 참조하세요.

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

전체 코드

이 섹션에는 이 예제에 대한 관련 참조와 전체 코드가 나와 있습니다.

SDK 레퍼런스

NuGet 패키지:

- AWSSDK.EC2

프로그래밍 요소:

- 네임스페이스 Amazon.EC2

  클래스 AmazonEC2Client

  클래스 InstanceType
- 네임스페이스 Amazon.EC2.Model

  클래스 DescribeInstancesRequest

  클래스 DescribeInstancesResponse

  클래스 Instance

  클래스 InstanceNetworkInterfaceSpecification

  클래스 RunInstancesRequest

  클래스 RunInstancesResponse

코드

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

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

namespace EC2LaunchInstance
{
  // = = =
```

```
      // Launch the instances and wait for them to start running
      var instanceIds = await LaunchInstances(ec2Client, request);
      await CheckState(ec2Client, instanceIds);
    }


    //
    // Method to put together the properties needed to launch the instance.
    private static RunInstancesRequest GetRequestData(
      string groupID, string ami, string keyPairName, string subnetID)
    {
      // Common properties
      var groupIDs = new List<string>() { groupID };
      var request = new RunInstancesRequest()
      {
        // The first three of these would be additional command-line arguments or
similar.
        InstanceType = InstanceType.T1Micro,
        MinCount = 1,
        MaxCount = 1,
        ImageId = ami,
        KeyName = keyPairName
      };

      // Properties specifically for EC2 in a VPC.
      if(!string.IsNullOrEmpty(subnetID))
      {
        request.NetworkInterfaces =
          new List<InstanceNetworkInterfaceSpecification>() {
            new InstanceNetworkInterfaceSpecification() {
              DeviceIndex = 0,
              SubnetId = subnetID,
              Groups = groupIDs,
              AssociatePublicIpAddress = true
            }
          };
      }

      // Properties specifically for EC2-Classic
      else
      {
        request.SecurityGroupIds = groupIDs;
      }
```

```
      return request;
    }


    //
    // Method to launch the instances
    // Returns a list with the launched instance IDs
    private static async Task<List<string>> LaunchInstances(
      IAmazonEC2 ec2Client, RunInstancesRequest requestLaunch)
    {
      var instanceIds = new List<string>();
      RunInstancesResponse responseLaunch =
        await ec2Client.RunInstancesAsync(requestLaunch);

      Console.WriteLine("\nNew instances have been created.");
      foreach (Instance item in responseLaunch.Reservation.Instances)
      {
        instanceIds.Add(item.InstanceId);
        Console.WriteLine($"  New instance: {item.InstanceId}");
      }

      return instanceIds;
    }


    //
    // Method to wait until the instances are running (or at least not pending)
    private static async Task CheckState(IAmazonEC2 ec2Client, List<string>
instanceIds)
    {
      Console.WriteLine(
        "\nWaiting for the instances to start." +
        "\nPress any key to stop waiting. (Response might be slightly delayed.)");

      int numberRunning;
      DescribeInstancesResponse responseDescribe;
      var requestDescribe = new DescribeInstancesRequest{
        InstanceIds = instanceIds};

      // Check every couple of seconds
      int wait = 2000;
      while(true)
      {
```

```
        // Get and check the status for each of the instances to see if it's past
pending.
        // Once all instances are past pending, break out.
        // (For this example, we are assuming that there is only one reservation.)
        Console.Write(".");
        numberRunning = 0;
        responseDescribe = await ec2Client.DescribeInstancesAsync(requestDescribe);
        foreach(Instance i in responseDescribe.Reservations[0].Instances)
        {
          // Check the lower byte of State.Code property
          // Code == 0 is the pending state
          if((i.State.Code & 255) > 0) numberRunning++;
        }
        if(numberRunning == responseDescribe.Reservations[0].Instances.Count)
          break;

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

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


    //
    // Command-line help
    private static void PrintHelp()
    {
      Console.WriteLine(
        "\nUsage: EC2LaunchInstance -g <group-id> -a <ami-id> -k <keypair-name> [-s
<subnet-id>]" +
        "\n  -g, --group-id: The ID of the security group." +
        "\n  -a, --ami-id: The ID of an Amazon Machine Image." +
```

```
        "\n  -k, --keypair-name - The name of a key pair." +
        "\n  -s, --subnet-id: The ID of a subnet. Required only for EC2 in a VPC.");
    }
  }


  // = = =
```

```
        // 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);
    }
  }

}
```

추가 고려 사항

- EC2 인스턴스의 상태를 확인할 때 DescribeInstancesRequest 객체의 `Filter` 속성에 필터를 추가할 수 있습니다. 이 기술을 사용하면 요청을 특정 인스턴스(예: 특정 사용자 지정 태그가 있는 인스턴스)로 제한할 수 있습니다.

- 간략하게 나타내기 위해 일부 속성에는 일반적인 값이 지정되었습니다. 대신 프로그래밍 방식으로 또는 사용자 입력을 통해 이러한 속성 중 일부 또는 전체를 결정할 수 있습니다.

- RunInstancesRequest 객체의 `MinCount` 및 `MaxCount` 속성에 사용할 수 있는 값은 대상 가용 영역과 해당 인스턴스 유형에 허용되는 최대 인스턴스 수에 따라 결정됩니다. 자세한 내용은 Amazon EC2 일반 FAQ의 Amazon EC2에서 실행할 수 있는 인스턴스 수는 몇 개입니까를 참조하세요.

- 이 예제와 다른 인스턴스 유형을 사용하려는 경우 선택할 수 있는 여러 인스턴스 유형이 있습니다. 자세한 내용은 Amazon EC2 사용 설명서의 Amazon EC2 인스턴스 유형을 참조하세요. 인스턴스 유형 세부 정보 및 인스턴스 유형 탐색기도 참조하세요.

- 인스턴스를 시작할 때 IAM 역할을 인스턴스에 연결할 수도 있습니다. 이렇게 하려면 `Name` 속성이 IAM 역할의 이름으로 설정된 IamInstanceProfileSpecification 객체를 생성해야 합니다. 그런 다음 해당 객체를 RunInstancesRequest 객체의 `IamInstanceProfile` 속성에 추가합니다.

  > **Note**
  >
  > IAM 역할이 연결된 EC2 인스턴스를 시작하려면 IAM 사용자의 구성에 특정 권한이 포함되어야 합니다. 필요한 권한에 대한 자세한 내용은 Amazon EC2 사용 설명서의 IAM 역할을 인스턴스에 전달할 수 있는 사용자 권한 부여를 참조하세요.

(선택 사항) 인스턴스에 연결

인스턴스가 실행된 후에는 적절한 원격 클라이언트를 사용하여 인스턴스에 원격으로 연결할 수 있습니다. Linux 및 Windows 인스턴스 모두에 대해 인스턴스의 퍼블릭 IP 주소 또는 퍼블릭 DNS 이름이 필요합니다. 또한 다음 항목이 필요합니다.

Linux 인스턴스의 경우

SSH 클라이언트를 사용하여 Linux 인스턴스에 연결할 수 있습니다. 보안 그룹 업데이트에 설명된 대로 인스턴스를 시작할 때 사용한 보안 그룹이 포트 22에서의 SSH 트래픽을 허용하는지 확인합니다.

또한 인스턴스를 시작하는 데 사용한 키 페어의 비공개 부분, 즉 PEM 파일도 필요합니다.

자세한 내용은 Amazon EC2 사용 설명서의 Linux 인스턴스에 연결을 참조하세요.

Windows 인스턴스의 경우

RDP 클라이언트를 사용하여 인스턴스에 연결할 수 있습니다. 보안 그룹 업데이트에 설명된 대로 인스턴스를 시작할 때 사용한 보안 그룹이 포트 3389에서의 RDP 트래픽을 허용하는지 확인합니다.

또한 관리자 암호가 필요합니다. 다음 예제 코드를 사용하여 이를 얻을 수 있습니다. 이 코드에는 인스턴스 ID와 인스턴스 시작에 사용된 키 페어의 비공개 부분, 즉 PEM 파일이 필요합니다.

자세한 내용은 Amazon EC2 사용 설명서의 Windows 인스턴스에 연결을 참조하세요.

> **Warning**
>
> 이 예제 코드는 인스턴스의 일반 텍스트 관리자 암호를 반환합니다.

SDK 레퍼런스

NuGet 패키지:

- AWSSDK.EC2

프로그래밍 요소:

- 네임스페이스 Amazon.EC2

  클래스 AmazonEC2Client
- 네임스페이스 Amazon.EC2.Model

  클래스 GetPasswordDataRequest

  클래스 GetPasswordDataResponse

코드

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

namespace EC2GetWindowsPassword
{
  // = = =
```

```
    //
    // Method to get the administrator password of a Windows EC2 instance
    private static async Task<string> GetPassword(
      IAmazonEC2 ec2Client, string instanceID, string pemFilename)
    {
      string password = string.Empty;
      GetPasswordDataResponse response =
        await ec2Client.GetPasswordDataAsync(new GetPasswordDataRequest{
          InstanceId = instanceID});
      if(response.PasswordData != null)
      {
        password = response.GetDecryptedPassword(File.ReadAllText(pemFilename));
      }
      else
      {
        Console.WriteLine($"\nThe password is not available for instance
{instanceID}.");
        Console.WriteLine($"If this is a Windows instance, the password might not be
ready.");
      }
      return password;
    }


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

  // = = =
```

```
    //
    // 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);
    }
  }

}
```

정리

EC2 인스턴스가 더 이상 필요하지 않은 경우 Amazon EC2 인스턴스 종료의 설명에 따라 인스턴스를 삭제하십시오.

Amazon EC2 인스턴스 종료

하나 이상의 Amazon EC2 인스턴스가 더 이상 필요하지 않으면 인스턴스를 종료할 수 있습니다.

이 예제에서는를 사용하여 EC2 인스턴스 AWS SDK for .NET 를 종료하는 방법을 보여줍니다. 인스턴스 ID를 입력으로 사용합니다.

SDK 레퍼런스

NuGet 패키지:

- AWSSDK.EC2

프로그래밍 요소:

- 네임스페이스 Amazon.EC2

  클래스 AmazonEC2Client
- 네임스페이스 Amazon.EC2.Model

  클래스 TerminateInstancesRequest

  클래스 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);
      }
    }
  }
}
```

예제를 실행한 후에는 Amazon EC2 콘솔에 로그인하여 EC2 인스턴스가 종료되었는지 확인하는 것이 좋습니다.

Amazon EC2 스팟 인스턴스 자습서

이 자습서에서는를 사용하여 Amazon EC2 스팟 인스턴스 AWS SDK for .NET 를 관리하는 방법을 보여줍니다.

개요

스팟 인스턴스를 사용하면 온디맨드 가격보다 저렴한 비용으로 미사용 Amazon EC2 용량을 요청할 수 있습니다. 이렇게 하면 중단될 수 있는 애플리케이션에 대한 EC2 비용을 크게 낮출 수 있습니다.

다음은 스팟 인스턴스 요청 및 사용 방법의 전반적인 요약입니다.

1. 지불하고자 하는 최고 가격을 지정하는 스팟 인스턴스 요청을 생성합니다.
2. 요청이 처리되면 다른 Amazon EC2 인스턴스와 마찬가지로 인스턴스를 실행합니다.
3. 원하는 기간만큼 인스턴스를 실행한 다음 종료합니다. 단, 스팟 가격이 변경되어 인스턴스가 자동으로 종료되는 경우는 예외입니다.
4. 더 이상 필요하지 않은 스팟 인스턴스 요청을 정리하여 스팟 인스턴스가 더 이상 생성되지 않도록 합니다.

여기까지 스팟 인스턴스의 대략적 개요였습니다. 스팟 인스턴스를 더 잘 이해하려면 Amazon EC2 사용 설명서의 스팟 인스턴스를 참조하세요.

이 자습서 소개

이 자습서를 따르면 AWS SDK for .NET 를 사용하여 다음을 수행할 수 있습니다.

- 스팟 인스턴스 요청 생성
- 스팟 인스턴스 요청이 이행된 시점 확인
- 스팟 인스턴스 요청 취소
- 연결된 인스턴스 종료

다음 섹션에서는 이 예제의 코드 조각과 기타 정보를 제공합니다. 예제의 전체 코드는 코드 조각 뒤에 표시되며, 그대로 빌드하고 실행할 수 있습니다.

주제

- 사전 조건
- 필요한 내용 수집
- 스팟 인스턴스 요청 생성
- 스팟 인스턴스 요청 상태 확인
- 스팟 인스턴스 요청 정리
- 스팟 인스턴스 정리
- 전체 코드
- 추가 고려 사항

사전 조건

API 및 사전 조건에 대한 자세한 내용은 상위 섹션(Amazon EC2 작업)을 참조하세요.

필요한 내용 수집

스팟 인스턴스 요청을 생성하려면 몇 가지 사항이 필요합니다.

- 인스턴스 수와 인스턴스 유형입니다. 선택할 수 있는 여러 인스턴스 유형이 있습니다. 자세한 내용은 Amazon EC2 사용 설명서의 Amazon EC2 인스턴스 유형을 참조하세요. 인스턴스 유형 세부 정보 및 인스턴스 유형 탐색기도 참조하세요.

이 자습서에서 기본 수는 1입니다.

- 인스턴스를 생성하는 데 사용되는 Amazon Machine Image(AMI)입니다. AMI에 대한 자세한 내용은 Amazon EC2 사용 설명서의 Amazon Machine Image(AMI)를 참조하세요. 특히 AMI 찾기 및 공유 AMI를 참조하세요.

- 인스턴스 시간당 지불하려는 최고 가격입니다. Amazon EC2 요금 페이지에서 모든 인스턴스 유형(온디맨드 인스턴스 및 스팟 인스턴스 모두)의 가격을 확인할 수 있습니다. 이 자습서의 기본 가격은 나중에 설명합니다.

- 인스턴스에 원격으로 연결하려는 경우 적절한 구성 및 리소스를 갖춘 보안 그룹을 사용합니다. 이에 대한 설명은 Amazon EC2 에서 보안 그룹 작업에 나와 있으며 필요한 정보를 수집하고 인스턴스에 연결하는 방법에 대한 자세한 내용은 Amazon EC2 인스턴스 시작에 나와 있습니다. 쉽게 설명하기 위해 이 자습서에서는 모든 새 AWS 계정에 있는 default라는 보안 그룹을 사용합니다.

스팟 인스턴스를 요청하기 위한 접근 방식에는 여러 가지가 있습니다. 다음은 일반적인 전략입니다.

- 온디맨드 요금보다 저렴한 요금으로 요청하세요.
- 결과 계산 값을 기준으로 요청하세요.
- 가능한 한 빨리 컴퓨팅 용량을 획득하도록 요청하세요.

다음 설명은 Amazon EC2 사용 설명서의 스팟 인스턴스 요금 기록을 참조합니다.

온디맨드 이하로 비용 줄이기

실행하는 데 몇 시간 또는 며칠이 걸리는 일괄 처리 작업이 있습니다. 그러나 시작 및 종료 시점은 유연하게 선택할 수 있습니다. 그 작업을 온디맨드 인스턴스보다 적은 비용으로 완료할 수 있는지 확인하고 싶다면

Amazon EC2 콘솔 또는 Amazon EC2 API를 사용하여 인스턴스 유형에 대한 스팟 가격 기록을 살펴보십시오. 특정 가용 영역에서 원하는 인스턴스 유형의 가격 내역을 분석하였다면 요청에 사용할 수 있는 다른 방법이 두 가지 있습니다.

- 일회적인 스팟 인스턴스 요청이 이행되고 해당 작업을 완료하기에 충분한 연속적 컴퓨팅 시간 동안 실행될 가능성이 높을 것이라는 기대로 스팟 가격 범위의 상단(여전히 온디맨드 가격보다는 낮음)에서 요청을 지정합니다.

- 가격 범위의 하단에서 요청을 지정하고 영구 요청을 통해 시간이 지남에 따라 시작되는 여러 인스턴스를 결합하는 계획을 세울 수도 있습니다. 인스턴스는 합계 시간 동안 충분히 실행되어 훨씬 더 낮은 총 비용으로도 작업을 완료합니다.

결과의 가치에 대한 요금만 지불

실행할 데이터 처리 작업이 있는 경우 작업 당사자는 그 작업의 결과가 지니는 가치를 충분히 이해하여 컴퓨팅 비용 측면에서 그 결과의 가치가 얼마나 되는지 알 수 있습니다.

해당 인스턴스 유형의 스팟 가격 내역을 분석한 후 컴퓨팅 시간의 비용이 해당 작업의 결과가 지니는 가치보다 더 많지 않은 가격을 선택합니다. 영구 요청을 생성하여 스팟 가격이 요청 가격까지 또는 그 이하로 변동하는 과정에서 간헐적으로 실행되도록 합니다.

신속하게 컴퓨팅 용량 획득

온디맨드 인스턴스를 통해서는 사용할 수 없는 추가 용량이 예기치 않게 단기간 동안 필요한 경우가 있습니다. 이때는 해당 인스턴스 유형의 스팟 가격 내역을 분석한 후 과거의 최고 가격보다 높은 가격을 선택함으로써 요청이 신속하게 이행되고 완료될 때까지 컴퓨팅이 지속될 수 있는 가능성을 크게 높입니다.

필요한 사항을 수집하고 전략을 선택했으면 스팟 인스턴스를 요청할 준비가 된 것입니다. 이 자습서에서는 기본 스팟 인스턴스 최고 가격이 온디맨드 가격(이 자습서의 경우 0.003 USD)과 동일하게 설정됩니다. 이 방법으로 가격을 설정하면 요청이 이행될 가능성이 극대화됩니다.

스팟 인스턴스 요청 생성

다음 코드 조각은 이전에 수집한 요소를 사용하여 스팟 인스턴스 요청을 생성하는 방법을 보여줍니다.

이 주제의 끝 부분에 있는 예제에서는 사용 중인 이 코드 조각을 보여줍니다.

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

이 메서드에서 반환되는 중요한 값은 반환된 SpotInstanceRequest 객체의 `SpotInstanceRequestId` 멤버에 포함된 스팟 인스턴스 요청 ID입니다.

> **Note**
>
> 시작된 모든 스팟 인스턴스에 대해 요금이 부과됩니다. 불필요한 비용을 피하려면 모든 요청을 취소하고 인스턴스를 종료해야 합니다.

스팟 인스턴스 요청 상태 확인

다음 코드 조각은 스팟 인스턴스 요청에 대한 정보를 가져오는 방법을 보여줍니다. 이 정보를 사용하여 스팟 인스턴스 요청이 이행될 때까지 계속 기다릴지 여부와 같은 코드의 특정 결정을 내릴 수 있습니다.

이 주제의 끝 부분에 있는 예제에서는 사용 중인 이 코드 조각을 보여줍니다.

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

이 메서드는 인스턴스 ID, 상태 및 상태 코드와 같은 스팟 인스턴스 요청에 대한 정보를 반환합니다. 스팟 인스턴스 요청의 상태 코드에 대한 자세한 내용은 Amazon EC2 사용 설명서의 스팟 요청 상태를 참조하세요.

스팟 인스턴스 요청 정리

스팟 인스턴스를 더 이상 요청할 필요가 없는 경우, 미해결 요청을 모두 취소하여 해당 요청이 다시 처리되지 않도록 하는 것이 중요합니다. 다음 코드 조각은 스팟 인스턴스 요청을 취소하는 방법을 보여줍니다.

이 주제의 끝 부분에 있는 예제에서는 사용 중인 이 코드 조각을 보여줍니다.

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

스팟 인스턴스 정리

불필요한 비용을 피하려면 스팟 인스턴스 요청으로 시작된 인스턴스를 모두 종료하는 것이 중요합니다. 스팟 인스턴스를 취소하는 것만으로는 인스턴스가 종료되지 않는데, 이는 인스턴스에 대해 계속해서 비용이 부과됨을 뜻합니다. 다음 코드 조각은 활성 스팟 인스턴스의 인스턴스 식별자를 획득한 후 인스턴스를 종료하는 방법을 보여줍니다.

이 주제의 끝 부분에 있는 예제에서는 사용 중인 이 코드 조각을 보여줍니다.

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

전체 코드

다음 코드 예제는 앞에서 설명한 메서드를 호출하여 스팟 인스턴스 요청을 생성 및 취소하고 스팟 인스턴스를 종료합니다.

SDK 레퍼런스

NuGet 패키지:

- AWSSDK.EC2

프로그래밍 요소:

- 네임스페이스 Amazon.EC2

  클래스 AmazonEC2Client

  클래스 InstanceType

- 네임스페이스 Amazon.EC2.Model

  클래스 CancelSpotInstanceRequestsRequest

클래스 DescribeSpotInstanceRequestsRequest

클래스 DescribeSpotInstanceRequestsResponse

클래스 InstanceStateChange

클래스 LaunchSpecification

클래스 RequestSpotInstancesRequest

클래스 RequestSpotInstancesResponse

클래스 SpotInstanceRequest

클래스 TerminateInstancesRequest

클래스 TerminateInstancesResponse

코드

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

추가 고려 사항

- 자습서를 실행한 후에는 Amazon EC2 콘솔에 로그인하여 스팟 인스턴스 요청이 취소되고 스팟 인스턴스가 종료되었는지 확인하는 것이 좋습니다.

를 사용하여 AWS Identity and Access Management (IAM)에 액세스 AWS SDK for .NET

는 AWS 고객이에서 사용자 및 사용자 권한을 관리할 수 있는 웹 서비스AWS Identity and Access Management인를 AWS SDK for .NET 지원합니다 AWS.

AWS Identity and Access Management (IAM) 사용자는 사용자가 생성하는 엔터티입니다 AWS. 개체는와 상호 작용하는 사람 또는 애플리케이션을 나타냅니다 AWS. IAM 사용자에 대한 자세한 내용은 IAM 사용 설명서의 IAM 사용자 및 IAM 및 STS 제한을 참조하세요.

IAM 정책을 생성하여 사용자에게 권한을 부여합니다. 정책에는 사용자가 수행할 수 있는 작업과 작업이 적용되는 리소스를 나열한 정책 문서가 포함됩니다. IAM 정책에 대한 자세한 내용은 IAM 사용 설명서에서 정책 및 권한을 참조하세요.

> **Warning**
>
> 보안 위험을 방지하려면 목적별 소프트웨어를 개발하거나 실제 데이터로 작업할 때 IAM 사용자를 인증에 사용하지 마세요. 대신 AWS IAM Identity Center과 같은 보안 인증 공급자를 통한 페더레이션을 사용하십시오.

API

는 IAM 클라이언트용 APIs AWS SDK for .NET 제공합니다. API를 사용하면 사용자, 역할, 액세스 키와 같은 IAM 기능을 사용할 수 있습니다.

이 섹션에는 이러한 API로 작업할 때 따를 수 있는 패턴을 보여주는 몇 가지 예제가 포함되어 있습니다. 전체 API 세트를 보려면 AWS SDK for .NET API 참조를 참조하세요(그리고 "Amazon.IdentityManagement"로 스크롤하세요).

이 섹션에는 보안 인증을 보다 쉽게 관리할 수 있도록 Amazon EC2 인스턴스에 IAM 역할을 연결하는 방법을 보여주는 예제도 포함되어 있습니다.

IAM API는 AWSSDK.IdentityManagement NuGet 패키지에서 제공됩니다.

사전 조건

시작하기 전에 환경을 설정하고 프로젝트를 구성했는지 확인합니다. 또한 SDK 사용의 정보를 검토합니다.

주제

주제

- JSON에서 IAM 관리형 정책 생성
- IAM 관리형 정책의 정책 문서 표시
- IAM 역할을 사용하여 액세스 권한 부여

JSON에서 IAM 관리형 정책 생성

이 예제에서는 AWS SDK for .NET 를 사용하여 JSON으로 지정된 정책 문서에서 IAM 관리형 정책을 생성하는 방법을 보여줍니다. 애플리케이션은 IAM 클라이언트 객체를 생성하고 파일에서 정책 문서를 읽은 다음 정책을 생성합니다.

> **Note**
>
> JSON으로 된 예제 정책 문서는 이 주제 끝에 있는 추가 고려 사항을 참조하세요.

다음 섹션에서는 이 예제의 코드 조각을 제공합니다. 예제의 전체 코드는 그 뒤에 표시되며, 그대로 빌드하고 실행할 수 있습니다.

주제

- 정책을 생성합니다.
- 전체 코드
- 추가 고려 사항

정책을 생성합니다.

다음 코드 조각은 지정된 이름과 정책 문서로 IAM 관리형 정책을 생성합니다.

이 주제의 끝 부분에 있는 예제에서는 사용 중인 이 코드 조각을 보여줍니다.

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

전체 코드

이 섹션에는 이 예제에 대한 관련 참조와 전체 코드가 나와 있습니다.

SDK 레퍼런스

NuGet 패키지:

- AWSSDK.IdentityManagement

프로그래밍 요소:

- 네임스페이스 Amazon.IdentityManagement

  클래스 AmazonIdentityManagementServiceClient
- 네임스페이스 Amazon.IdentityManagement.Model

  클래스 CreatePolicyRequest

  클래스 CreatePolicyResponse

코드

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Threading.Tasks;
using Amazon.IdentityManagement;
using Amazon.IdentityManagement.Model;

namespace IamCreatePolicyFromJson
{
  // = = =
```

```
      string policyFilename =
        CommandLine.GetArgument(parsedArgs, null, "-j", "--json-filename");
      if(   string.IsNullOrEmpty(policyName)
         || (string.IsNullOrEmpty(policyFilename) || !
policyFilename.EndsWith(".json")))
        CommandLine.ErrorExit(
          "\nOne or more of the required arguments is missing or incorrect." +
          "\nRun the command with no arguments to see help.");

      // Create an IAM service client
      var iamClient = new AmazonIdentityManagementServiceClient();

      // Create the new policy
      var response = await CreateManagedPolicy(iamClient, policyName, policyFilename);
      Console.WriteLine($"\nPolicy {response.Policy.PolicyName} has been created.");
      Console.WriteLine($"  Arn: {response.Policy.Arn}");
    }


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


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

```
 // = = =
```

```
      }

      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);
    }
  }

}
```

추가 고려 사항

- 다음은 JSON 파일에 복사하여 이 애플리케이션의 입력으로 사용할 수 있는 예제 정책 문서입니다.

  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": "*"
    }
  ]
}
```

- IAM 콘솔을 살펴보면 정책이 생성되었는지 확인할 수 있습니다. 정책 필터 드롭다운 목록에서 고객 관리형을 선택합니다. 필요 없는 정책을 삭제합니다.

- 정책 생성에 대한 자세한 내용은 IAM 사용 설명서의 IAM 정책 생성 및 IAM JSON 정책 참조를 참조하세요.

IAM 관리형 정책의 정책 문서 표시

이 예제에서는를 사용하여 정책 문서를 AWS SDK for .NET 표시하는 방법을 보여줍니다. 애플리케이션은 IAM 클라이언트 객체를 생성하고, 지정된 IAM 관리형 정책의 기본 버전을 찾은 다음, 정책 문서를 JSON으로 표시합니다.

다음 섹션에서는 이 예제의 코드 조각을 제공합니다. 예제의 전체 코드는 그 뒤에 표시되며, 그대로 빌드하고 실행할 수 있습니다.

주제

- 기본 버전 찾기

- 정책 문서 표시
- 전체 코드

기본 버전 찾기

다음 코드 조각은 해당 IAM 정책의 기본 버전을 찾습니다.

이 주제의 끝 부분에 있는 예제에서는 사용 중인 이 코드 조각을 보여줍니다.

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

정책 문서 표시

다음 코드 조각은 해당 IAM 정책의 정책 문서를 JSON으로 표시합니다.

이 주제의 끝 부분에 있는 예제에서는 사용 중인 이 코드 조각을 보여줍니다.

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

전체 코드

이 섹션에는 이 예제에 대한 관련 참조와 전체 코드가 나와 있습니다.

SDK 레퍼런스

NuGet 패키지:

- AWSSDK.IdentityManagement

프로그래밍 요소:

- 네임스페이스 Amazon.IdentityManagement

  클래스 AmazonIdentityManagementServiceClient
- 네임스페이스 Amazon.IdentityManagement.Model

  클래스 GetPolicyVersionRequest

  클래스 GetPolicyVersionResponse

  클래스 ListPolicyVersionsRequest

  클래스 ListPolicyVersionsResponse

  클래스 PolicyVersion

코드

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

IAM 역할을 사용하여 액세스 권한 부여

이 자습서에서는 AWS SDK for .NET 를 사용하여 Amazon EC2 인스턴스에서 IAM 역할을 활성화하는 방법을 보여줍니다.

개요

에 대한 모든 요청은에서 발급한 자격 증명을 사용하여 암호화 방식으로 서명해야 AWS 합니다 AWS. 따라서 Amazon EC2 인스턴스에서 실행되는 애플리케이션에 대한 자격 증명을 관리할 전략이 필요합니다. 이 자격 증명을 안전하게 배포, 저장, 교체해야 할 뿐 아니라 애플리케이션에 접근할 수 있는 상태로 유지해야 합니다.

IAM 역할을 사용하면 이러한 자격 증명을 효과적으로 관리할 수 있습니다. IAM 역할을 생성하고 애플리케이션에 필요한 권한으로 구성한 다음 해당 역할을 EC2 인스턴스에 연결합니다. IAM 역할 사용의 이점에 대한 자세한 내용은 Amazon EC2 사용 설명서의 Amazon EC2에 대한 IAM 역할을 참조하세요. 또한 IAM 사용 설명서의 IAM 역할에 대한 정보도 참조하세요.

를 사용하여 빌드된 애플리케이션의 경우 애플리케이션이 AWS 서비스에 대한 클라이언트 객체를 구성할 AWS SDK for .NET때 객체는 여러 잠재적 소스에서 자격 증명을 검색합니다. 검색 순서는 보안 인증 정보 및 프로파일 확인에 나와 있습니다.

클라이언트 객체가 다른 소스에서 보안 인증을 찾을 수 없는 경우, IAM 역할에 구성된 것과 동일한 권한이 있고 EC2 인스턴스의 메타데이터에 있는 임시 보안 인증을 검색합니다. 이러한 자격 증명은 클라이언트 객체 AWS 에서를 호출하는 데 사용됩니다.

이 자습서 소개

이 자습서를 따르면 AWS SDK for .NET (및 기타 도구)를 사용하여 IAM 역할이 연결된 Amazon EC2 인스턴스를 시작한 다음 IAM 역할의 권한을 사용하여 인스턴스에서 애플리케이션을 볼 수 있습니다.

주제

- 샘플 Amazon S3 애플리케이션 생성
- IAM 역할 생성
- EC2 인스턴스 시작 및 IAM 역할 연결
- EC2 인스턴스에 연결
- EC2 인스턴스에서 샘플 애플리케이션 실행
- 정리

샘플 Amazon S3 애플리케이션 생성

이 샘플 애플리케이션은 Amazon S3에서 객체를 검색합니다. 애플리케이션을 실행하려면 다음이 필요합니다.

- 텍스트 파일이 포함된 Amazon S3 버킷입니다.
- AWS 버킷에 액세스할 수 있는 개발 시스템의 자격 증명입니다.

Amazon S3 버킷 생성 및 객체 업로드 방법에 대한 자세한 내용은 Amazon Simple Storage Service 사용 설명서를 참조하세요. 자격 AWS 증명에 대한 자세한 내용은 섹션을 참조하세요를 AWS SDK for .NET 사용하여 인증 AWS.

다음 코드를 사용하여 .NET Core 프로젝트를 생성합니다. 그런 다음 개발 머신에서 애플리케이션을 테스트합니다.

> **Note**
>
> 개발 머신에 .NET Core 런타임이 설치되어 있으므로 애플리케이션을 게시하지 않고도 애플리케이션을 실행할 수 있습니다. 이 자습서의 후반부에서 EC2 인스턴스를 생성할 때 인스턴스에 .NET Core 런타임을 설치하도록 선택할 수 있습니다. 이렇게 하면 비슷한 환경에서 더 작은 파일을 전송할 수 있습니다.
> 하지만 인스턴스에 .NET Core 런타임을 설치하지 않도록 선택할 수도 있습니다. 이 작업 과정을 선택하는 경우 애플리케이션을 인스턴스로 전송할 때 모든 종속성이 포함되도록 애플리케이션을 게시해야 합니다.

SDK 레퍼런스

NuGet 패키지:

- AWSSDK.S3

프로그래밍 요소:

- 네임스페이스 Amazon.S3

  클래스 AmazonS3Client
- 네임스페이스 Amazon.S3.Model

클래스 GetObjectResponse

코드

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Threading.Tasks;
using Amazon.S3;
using Amazon.S3.Model;

namespace S3GetTextItem
{
  // = = =
```

```
      // Create the S3 client object and get the file object from the bucket.
      var response = await GetObject(new AmazonS3Client(), bucket, item);

      // Write the contents of the file object to the given output file.
      var reader = new StreamReader(response.ResponseStream);
      string contents = reader.ReadToEnd();
      using (var s = new FileStream(outFile, FileMode.Create))
      using (var writer = new StreamWriter(s))
        writer.WriteLine(contents);
    }


    //
    // Method to get an object from an S3 bucket.
    private static async Task<GetObjectResponse> GetObject(
      IAmazonS3 s3Client, string bucket, string item)
    {
        Console.WriteLine($"Retrieving {item} from bucket {bucket}.");
        return await s3Client.GetObjectAsync(bucket, item);
    }


    //
    // Command-line help
    private static void PrintHelp()
    {
      Console.WriteLine(
        "\nUsage: S3GetTextItem -b <bucket-name> -t <text-object> -o <output-filename>"
+
        "\n  -b, --bucket-name: The name of the S3 bucket." +
        "\n  -t, --text-object: The name of the text object in the bucket." +
        "\n  -o, --output-filename: The name of the file to write the text to.");
    }
  }


  // = = =
```

```
    // 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);
    }
  }

}
```

원하는 경우 개발 머신에서 사용하는 보안 인증을 일시적으로 제거하여 애플리케이션이 어떻게 반응하는지 확인할 수 있습니다. (하지만 작업을 마치면 보안 인증을 복원해야 합니다.)

IAM 역할 생성

Amazon S3에 액세스할 수 있는 적절한 권한이 있는 IAM 역할을 생성합니다.

1. IAM 콘솔을 엽니다.
2. 탐색 창에서 역할을 선택한 후 역할 생성을 선택합니다.
3. AWS 서비스를 선택하고, EC2를 찾아 선택하고, 다음: 권한을 선택합니다.
4. 권한 정책 연결에서 AmazonS3ReadOnlyAccess를 찾아 선택합니다. 원하는 경우 정책을 검토한 후 다음: 태그를 선택합니다.
5. 원하는 경우 태그를 추가한 후 다음: 검토를 선택합니다.

6. 역할 이름 및 설명을 입력한 후 역할 생성을 선택합니다. EC2 인스턴스를 시작할 때 필요하므로 이 이름을 기억해 두십시오.

EC2 인스턴스 시작 및 IAM 역할 연결

앞에서 생성한 IAM 역할로 EC2 인스턴스를 시작합니다. 다음과 같은 방법으로 수행할 수 있습니다.

- EC2 콘솔 사용

  EC2 콘솔을 사용하여 인스턴스를 실행하려면 Amazon EC2 사용 설명서의 새로운 인스턴스 시작 마법사를 사용하여 인스턴스 시작을 참조하세요.

  시작 페이지를 살펴볼 때 IAM 인스턴스 프로파일에서 이전에 생성한 IAM 역할을 지정할 수 있도록 최소한 고급 세부 정보 창을 확장해야 합니다.

- 사용 AWS SDK for .NET

  이에 대한 자세한 내용은 해당 주제의 끝 부분에 있는 추가 고려 사항을 포함하여 Amazon EC2 인스턴스 시작을 참조하세요.

IAM 역할이 연결된 EC2 인스턴스를 시작하려면 IAM 사용자의 구성에 특정 권한이 포함되어야 합니다. 필요한 권한에 대한 자세한 내용은 Amazon EC2 사용 설명서의 IAM 역할을 인스턴스에 전달할 수 있는 사용자 권한 부여를 참조하세요.

EC2 인스턴스에 연결

EC2 인스턴스에 연결하면 샘플 애플리케이션을 전송한 다음 애플리케이션을 실행할 수 있습니다. 인스턴스를 시작하는 데 사용한 키 페어의 비공개 부분이 포함된 파일, 즉 PEM 파일이 필요합니다.

인스턴스에 연결하는 방법에 대한 자세한 내용은 Amazon EC2 사용 설명서의 Linux 인스턴스에 연결 또는 Windows 인스턴스에 연결을 참조하세요. 연결할 때는 개발 머신에서 인스턴스로 파일을 전송할 수 있는 방식으로 연결합니다.

Windows에서 Visual Studio를 사용하는 경우, Visual Studio용 도구 키트를 사용하여 인스턴스에 연결할 수도 있습니다. 자세한 내용은 AWS Toolkit for Visual Studio 사용 설명서의 Amazon EC2 인스턴스에 연결을 참조하세요.

EC2 인스턴스에서 샘플 애플리케이션 실행

1. 애플리케이션 파일을 로컬 드라이브에서 인스턴스로 복사합니다.

전송하는 파일은 애플리케이션을 빌드한 방법과 인스턴스에 .NET Core 런타임이 설치되어 있는지 여부에 따라 달라집니다. 인스턴스로 파일을 전송하는 방법에 대한 자세한 내용은 Amazon EC2 사용 설명서의 Linux 인스턴스에 연결(해당 하위 섹션 참조) 또는 Windows 인스턴스로 파일 전송을 참조하세요.

2. 애플리케이션을 시작하고 개발 머신에서와 동일한 결과로 실행되는지 확인합니다.
3. 애플리케이션이 IAM 역할에서 제공하는 보안 인증을 사용하는지 확인합니다.

   a. Amazon EC2 콘솔을 엽니다.
   b. 인스턴스를 선택하고 작업, 인스턴스 설정, IAM 역할 연결/바꾸기를 통해 IAM 역할을 분리합니다.
   c. 애플리케이션을 다시 실행하여 권한 부여 오류가 반환되는지 확인합니다.

정리

이 자습서를 마치고 생성한 EC2 인스턴스를 더 이상 사용하지 않으려면 인스턴스를 종료하여 원치 않는 비용이 발생하지 않도록 하십시오. Amazon EC2 콘솔에서 또는 Amazon EC2 인스턴스 종료에 설명된 대로 프로그래밍 방식으로 이 작업을 수행할 수 있습니다. 필요에 따라 이 자습서를 위해 생성한 다른 리소스도 삭제할 수 있습니다. 여기에는 IAM 역할, EC2 키 페어 및 PEM 파일, 보안 그룹 등이 포함될 수 있습니다.

Amazon Simple Storage Service 인터넷 스토리지 사용

는 인터넷용 스토리지인 Amazon S3를 AWS SDK for .NET 지원합니다. 이 서비스는 개발자가 더 쉽게 웹 규모 컴퓨팅 작업을 수행할 수 있도록 설계되었습니다.

API

는 Amazon S3 클라이언트에 대한 APIs AWS SDK for .NET 제공합니다. API를 사용하면 버킷 및 항목과 같은 Amazon S3 리소스를 사용할 수 있습니다. Amazon S3용 전체 API 세트를 보려면 다음을 참조하세요.

- AWS SDK for .NET API 참조(및 "Amazon.S3"로 스크롤).
- Amazon.Extensions.S3.Encryption 문서

Amazon S3 API는 다음과 같은 NuGet 패키지에서 제공됩니다.

- AWSSDK.S3

- Amazon.Extensions.S3.Encryption

사전 조건

시작하기 전에 환경을 설정하고 프로젝트를 구성했는지 확인합니다. 또한 SDK 사용의 정보를 검토합니다.

이 문서의 예제

이 문서의 다음 주제에서는를 사용하여 Amazon S3로 작업 AWS SDK for .NET 하는 방법을 보여줍니다.

- S3 암호화에 KMS 키 사용

다른 문서에 있는 예제

Amazon S3 개발자 안내서에 대한 다음 링크는를 사용하여 Amazon S3로 작업하는 방법에 대한 추가 예제 AWS SDK for .NET 를 제공합니다.

> **Note**
>
> 이러한 예제와 추가 프로그래밍 고려 사항은 .NET Framework를 AWS SDK for .NET 사용하는 버전 3에 대해 생성되었지만 .NET Core를 AWS SDK for .NET 사용하는의 이후 버전에서도 실행 가능합니다. 코드를 약간 조정해야 하는 경우가 있습니다.

Amazon S3 프로그래밍 예시

- ACL 관리
- 버킷 생성
- 객체 업로드
- 상위 수준 API(Amazon.S3.Transfer.TransferUtility)를 사용한 멀티파트 업로드
- 하위 레벨 API를 통한 멀티파트 업로드
- 객체 나열
- 키 나열
- 객체 가져오기

- 객체 복사
- 멀티파트 업로드 API를 사용한 객체 복사
- 객체 삭제
- 여러 객체 삭제
- 객체 복원
- 알림용 버킷 구성
- 객체 수명 주기 관리
- 미리 서명된 객체 URL 생성
- 웹 사이트 관리
- Cross-Origin 리소스 공유(CORS) 활성화

추가 프로그래밍 고려 사항

- Amazon S3 프로그래밍에 AWS SDK for .NET 사용
- IAM 사용자 임시 자격 증명을 사용하여 요청하기
- 연합된 사용자의 임시 자격 증명을 사용하여 요청하기
- 서버 측 암호화 지정
- 고객 제공 암호화 키를 사용한 서버 측 암호화 지정

에서 Amazon S3 암호화에 AWS KMS 키 사용 AWS SDK for .NET

이 예제에서는 AWS Key Management Service 키를 사용하여 Amazon S3 객체를 암호화하는 방법을 보여줍니다. 애플리케이션은 고객 마스터 키(CMK)를 생성하고 이를 사용하여 클라이언트측 암호화를 위한 AmazonS3EncryptionClientV2 객체를 생성합니다. 애플리케이션은 해당 클라이언트를 사용하여 기존 Amazon S3 버킷의 지정된 텍스트 파일로부터 암호화된 객체를 생성합니다. 그런 다음 객체를 복호화하고 콘텐츠를 표시합니다.

> **Warning**
>
> `AmazonS3EncryptionClient`라는 유사한 클래스는 더 이상 사용되지 않으며 `AmazonS3EncryptionClientV2` 클래스보다 보안성이 떨어집니다. S3 암호화 클라이언트 마이그레이션(V1에서 V2로)를 사용하는 기존 코드를 마이그레이션하려면 `AmazonS3EncryptionClient`을 참조하세요.

주제

- 암호화 자료 생성
- Amazon S3 객체 생성 및 암호화
- 전체 코드
- 추가 고려 사항

암호화 자료 생성

다음 코드 조각은 KMS 키 ID가 포함된 `EncryptionMaterials` 객체를 생성합니다.

이 주제의 끝 부분에 있는 예제에서는 사용 중인 이 코드 조각을 보여줍니다.

```
     // 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);
```

Amazon S3 객체 생성 및 암호화

다음 코드 조각은 앞서 생성한 암호화 자료를 사용하여 `AmazonS3EncryptionClientV2` 객체를 생성합니다. 그런 다음 클라이언트를 사용하여 새 Amazon S3 객체를 생성하고 암호화합니다.

이 주제의 끝 부분에 있는 예제에서는 사용 중인 이 코드 조각을 보여줍니다.

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

전체 코드

이 섹션에는 이 예제에 대한 관련 참조와 전체 코드가 나와 있습니다.

SDK 레퍼런스

NuGet 패키지:

- Amazon.Extensions.S3.Encryption

프로그래밍 요소:

- 네임스페이스 Amazon.Extensions.S3.Encryption

  클래스 AmazonS3EncryptionClientV2

  클래스 AmazonS3CryptoConfigurationV2

  클래스 CryptoStorageMode

  클래스 EncryptionMaterialsV2
- 네임스페이스 Amazon.Extensions.S3.Encryption.Primitives

  클래스 KmsType
- 네임스페이스 Amazon.S3.Model

  클래스 GetObjectRequest

클래스 GetObjectResponse

클래스 PutObjectRequest

- 네임스페이스 Amazon.KeyManagementService

클래스 AmazonKeyManagementServiceClient

- 네임스페이스 Amazon.KeyManagementService.Model

클래스 CreateKeyRequest

클래스 CreateKeyResponse

코드

```
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
{
  // = = =
```

```
      }

      // Get the application arguments from the parsed list
      string bucketName =
        CommandLine.GetArgument(parsedArgs, null, "-b", "--bucket-name");
      string fileName =
        CommandLine.GetArgument(parsedArgs, null, "-f", "--file-name");
      string itemName =
        CommandLine.GetArgument(parsedArgs, null, "-i", "--item-name");
      if(string.IsNullOrEmpty(bucketName) || (string.IsNullOrEmpty(fileName)))
        CommandLine.ErrorExit(
          "\nOne or more of the required arguments is missing or incorrect." +
          "\nRun the command with no arguments to see help.");
      if(!File.Exists(fileName))
        CommandLine.ErrorExit($"\nThe given file {fileName} doesn't exist.");
      if(string.IsNullOrEmpty(itemName))
        itemName = Path.GetFileName(fileName);

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

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


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

```
      {
        StorageMode = CryptoStorageMode.ObjectMetadata
      };
      var s3EncClient = new AmazonS3EncryptionClientV2(config, materials);

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

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


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

  }

  // = = =
```

```
    //
    // 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);
    }
  }

}
```

추가 고려 사항

- 이 예제의 결과를 확인할 수 있습니다. 이렇게 하려면 Amazon S3 콘솔로 이동하여 애플리케이션에 제공한 버킷을 엽니다. 그런 다음 새 객체를 찾아 다운로드한 다음 텍스트 편집기에서 엽니다.

- AmazonS3EncryptionClientV2 클래스는 표준 AmazonS3Client 클래스와 동일한 인터페이스를 구현합니다. 이렇게 하면 코드를 AmazonS3EncryptionClientV2 클래스로 쉽게 포팅하여 클라이언트에서 암호화와 복호화가 자동으로 투명하게 수행되도록 할 수 있습니다.

- 키를 마스터 AWS KMS 키로 사용할 때의 한 가지 장점은 자체 마스터 키를 저장하고 관리할 필요가 없다는 것입니다.이 작업은에서 수행합니다 AWS. 두 번째 장점은의 AmazonS3EncryptionClientV2 클래스가의 AmazonS3EncryptionClientV2 클래스와 상호 운용 가능하다 AWS SDK for .NET 는 것입니다 AWS SDK for Java. 즉,를 사용하여 암호화 AWS

SDK for Java 하고를 사용하여 복호화할 수 있으며 AWS SDK for .NET그 반대의 경우도 마찬가지입니다.

> **Note**
>
> 의 `AmazonS3EncryptionClientV2` 클래스는 메타데이터 모드에서 실행되는 경우에만 KMS 마스터 키를 AWS SDK for .NET 지원합니다. 의 `AmazonS3EncryptionClientV2` 클래스에 대한 명령 파일 모드 AWS SDK for .NET 는의 `AmazonS3EncryptionClientV2` 클래스와 호환되지 않습니다 AWS SDK for Java.

- `AmazonS3EncryptionClientV2` 클래스를 사용한 클라이언트 측 암호화 및 봉투 암호화 작동 방식에 대한 자세한 내용은 AWS SDK for .NET 및 Amazon S3를 사용한 클라이언트 측 데이터 암호화를 참조하세요.

Amazon Simple Notification Service를 사용하여 클라우드에서 알림 전송

> **Note**
>
> 이 주제의 정보는 .NET Framework 및 AWS SDK for .NET 버전 3.3 이하를 기반으로 하는 프로젝트에만 해당됩니다.

는 애플리케이션, 최종 사용자 및 디바이스가 클라우드에서 즉시 알림을 보낼 수 있는 웹 서비스인 Amazon Simple Notification Service(Amazon SNS)를 AWS SDK for .NET 지원합니다. 자세한 내용은 Amazon SNS를 참조하세요.

Amazon SNS 주제 나열

다음 예제는 Amazon SNS 주제, 각 주제에 대한 구독, 각 주제에 대한 속성을 나열하는 방법을 보여줍니다. 이 예제에서는 기본 AmazonSimpleNotificationServiceClient를 사용합니다.

```
// 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));
```

Amazon SNS 주제로 메시지 전송

다음 예제는 메시지를 Amazon SNS 주제로 전송하는 방법을 보여줍니다. 이 예제에서는 하나의 인수(Amazon SNS 주제의 ARN)가 필요합니다.

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

GitHub의 명령줄에서 예제를 빌드하고 실행하는 방법에 대한 정보를 비롯한 전체 예제를 참조하십시오.

전화 번호로 SMS 메시지 전송

다음 예제는 SMS 메시지를 전화 번호로 전송하는 방법을 보여줍니다. 이 예제에서는 하나의 인수(전화 번호)가 필요합니다. 전화 번호는 주석에 설명된 두 가지 형식 중 하나여야 합니다.

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

GitHub의 명령줄에서 예제를 빌드하고 실행하는 방법에 대한 정보를 비롯한 전체 예제를 참조하십시오.

Amazon SQS를 사용한 메시징

는 시스템의 구성 요소 간에 메시지 또는 워크플로를 처리하는 메시지 대기열 서비스인 Amazon Simple Queue Service(Amazon SQS)를 AWS SDK for .NET 지원합니다.

Amazon SQS 대기열은 마이크로서비스, 분산 시스템, 서버리스 애플리케이션과 같은 소프트웨어 구성 요소 간에 메시지를 전송, 저장 및 수신할 수 있게 해주는 메커니즘을 제공합니다. 이를 통해 이러한 구성 요소를 분리할 수 있으므로 고유의 메시징 시스템을 설계하고 운영할 필요가 없습니다. Amazon SQS에서 대기열 및 메시지가 작동하는 방식에 대한 자세한 내용은 Amazon Simple Queue Service 개발자 안내서의 Amazon SQS 자습서 및 기본 Amazon SQS 아키텍처를 참조하세요.

> **Important**
>
> Amazon SQS는 대기열의 분산 특성 때문에 메시지를 전송한 순서대로 수신하도록 보장할 수 없습니다. 메시지 순서를 유지해야 하는 경우 Amazon SQS FIFO 대기열을 사용합니다.

API

는 Amazon SQS 클라이언트에 대한 APIs AWS SDK for .NET 제공합니다. API를 사용하면 대기열 및 메시지와 같은 Amazon SQS 기능을 사용할 수 있습니다. 이 섹션에는 이러한 API로 작업할 때 따를 수 있는 패턴을 보여주는 몇 가지 예제가 포함되어 있습니다. 전체 API 세트를 보려면 AWS SDK for .NET API 참조를 참조하세요(그리고 "Amazon.SQS"로 스크롤하세요).

Amazon SQS API는 AWSSDK.SQS NuGet 패키지를 통해 제공됩니다.

사전 조건

시작하기 전에 환경을 설정하고 프로젝트를 구성했는지 확인합니다. 또한 SDK 사용의 정보를 검토합니다.

주제

주제

- Amazon SQS 대기열 생성
- Amazon SQS 대기열 업데이트
- Amazon SQS 대기열 삭제
- Amazon SQS 메시지 전송
- Amazon SQS 메시지 수신

Amazon SQS 대기열 생성

이 예제에서는를 사용하여 Amazon SQS 대기열을 AWS SDK for .NET 생성하는 방법을 보여줍니다. ARN을 제공하지 않으면 애플리케이션에서 DLQ(Dead Letter Queue)를 생성합니다. 그런 다음 DLQ(Dead Letter Queue)(사용자가 제공한 대기열 또는 생성한 대기열)가 포함된 표준 메시지 대기열을 생성합니다.

명령줄 인수를 제공하지 않으면 애플리케이션은 모든 기존 대기열에 대한 정보만 표시합니다.

다음 섹션에서는 이 예제의 코드 조각을 제공합니다. 예제의 전체 코드는 그 뒤에 표시되며, 그대로 빌드하고 실행할 수 있습니다.

주제

- 기존 대기열 표시
- 대기열 생성
- 대기열의 ARN 가져오기
- 전체 코드
- 추가 고려 사항

기존 대기열 표시

다음 코드 조각은 SQS 클라이언트 리전의 기존 대기열 목록과 각 대기열의 속성을 보여줍니다.

이 주제의 끝 부분에 있는 예제에서는 사용 중인 이 코드 조각을 보여줍니다.

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

대기열 생성

다음 조각은 대기열을 생성합니다. 코드 조각에는 DLQ(Dead Letter Queue) 사용이 포함되지만 대기열에 반드시 DLQ(Dead Letter Queue)가 필요한 것은 아닙니다.

이 주제의 끝 부분에 있는 예제에서는 사용 중인 이 코드 조각을 보여줍니다.

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

대기열의 ARN 가져오기

다음 코드 조각은 지정된 대기열 URL로 식별되는 대기열의 ARN을 가져옵니다.

이 주제의 끝 부분에 있는 예제에서는 사용 중인 이 코드 조각을 보여줍니다.

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

전체 코드

이 섹션에는 이 예제에 대한 관련 참조와 전체 코드가 나와 있습니다.

SDK 레퍼런스

NuGet 패키지:

- AWSSDK.SQS

프로그래밍 요소:

- 네임스페이스 Amazon.SQS

  클래스 AmazonSQSClient

  클래스 QueueAttributeName
- 네임스페이스 Amazon.SQS.Model

  클래스 CreateQueueRequest

  클래스 CreateQueueResponse

  클래스 GetQueueAttributesResponse

  클래스 ListQueuesResponse

코드

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

```
using System.Collections.Generic;
using Amazon.SQS;
using Amazon.SQS.Model;

namespace SQSCreateQueue
{
  // = = =
```

```
      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}.");
    }
  }


  // = = =
```

```
    // 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);
    }
  }

}
```

추가 고려 사항

- 대기열 이름은 영문자, 숫자, 하이픈, 밑줄로 구성되어야 합니다.
- 대기열 이름과 대기열 URL은 대소문자를 구분합니다
- 대기열 URL이 필요한데 대기열 이름만 있는 경우에는 `AmazonSQSClient.GetQueueUrlAsync` 방법 중 하나를 사용합니다.
- 설정할 수 있는 다양한 대기열 속성에 대한 자세한 내용은 AWS SDK for .NET API 참조의 CreateQueueRequest 또는 Amazon Simple Queue Service API 참조의 SetQueueAttributes를 참조하세요.

- 이 예제는 생성한 대기열에 있는 모든 메시지에 대해 긴 폴링을 지정합니다. `ReceiveMessageWaitTimeSeconds` 속성을 사용하여 이를 수행합니다.

  AmazonSQSClient 클래스의 `ReceiveMessageAsync` 메서드를 호출하는 동안 긴 폴링을 지정할 수도 있습니다. 자세한 내용은 Amazon SQS 메시지 수신 단원을 참조하십시오.

  짧은 폴링과 긴 폴링에 대한 자세한 내용은 Amazon Simple Queue Service 개발자 안내서의 짧은 폴링 및 긴 폴링을 참조하세요.

- 배달 못한 편지 대기열은 성공적으로 처리되지 않은 메시지를 다른 (소스) 대기열에서 보낼 수 있는 대기열입니다. 자세한 정보는 Amazon Simple Queue Service 개발자 안내서의 Amazon SQS DLQ(Dead Letter Queue)를 참조하세요.

- Amazon SQS 콘솔에서도 대기열 목록과 이 예제의 결과를 확인할 수 있습니다.

Amazon SQS 대기열 업데이트

이 예제에서는를 사용하여 Amazon SQS 대기열 AWS SDK for .NET 을 업데이트하는 방법을 보여줍니다. 일부 점검 후 애플리케이션은 지정된 속성을 지정된 값으로 업데이트한 다음 대기열의 모든 속성을 표시합니다.

명령줄 인수에 대기열 URL만 포함된 경우 애플리케이션에서는 단순히 대기열의 모든 속성을 표시합니다.

다음 섹션에서는 이 예제의 코드 조각을 제공합니다. 예제의 전체 코드는 그 뒤에 표시되며, 그대로 빌드하고 실행할 수 있습니다.

주제

- 대기열 속성 표시
- 속성 이름 검증
- 대기열 속성 업데이트
- 전체 코드
- 추가 고려 사항

대기열 속성 표시

다음 코드 조각은 지정된 대기열 URL로 식별되는 대기열의 속성을 보여줍니다.

이 주제의 끝 부분에 있는 예제에서는 사용 중인 이 코드 조각을 보여줍니다.

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

속성 이름 검증

다음 코드 조각은 업데이트 중인 속성의 이름을 검증합니다.

이 주제의 끝 부분에 있는 예제에서는 사용 중인 이 코드 조각을 보여줍니다.

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

대기열 속성 업데이트

다음 코드 조각은 지정된 대기열 URL로 식별되는 대기열의 속성을 업데이트합니다.

이 주제의 끝 부분에 있는 예제에서는 사용 중인 이 코드 조각을 보여줍니다.

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

전체 코드

이 섹션에는 이 예제에 대한 관련 참조와 전체 코드가 나와 있습니다.

SDK 레퍼런스

NuGet 패키지:

- AWSSDK.SQS

프로그래밍 요소:

- 네임스페이스 Amazon.SQS

  클래스 AmazonSQSClient

  클래스 QueueAttributeName
- 네임스페이스 Amazon.SQS.Model

  클래스 GetQueueAttributesResponse

코드

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

namespace SQSUpdateQueue
{
  // = = =
```

```
    private const int MaxArgs = 3;
    private const int InvalidArgCount = 2;

    static async Task Main(string[] args)
    {
      // Parse the command line and show help if necessary
      var parsedArgs = CommandLine.Parse(args);
      if(parsedArgs.Count == 0)
      {
        PrintHelp();
        return;
      }
      if((parsedArgs.Count > MaxArgs) || (parsedArgs.Count == InvalidArgCount))
        CommandLine.ErrorExit("\nThe number of command-line arguments is incorrect." +
          "\nRun the command with no arguments to see help.");

      // Get the application arguments from the parsed list
      var qUrl = CommandLine.GetArgument(parsedArgs, null, "-q");
      var attribute = CommandLine.GetArgument(parsedArgs, null, "-a");
      var value = CommandLine.GetArgument(parsedArgs, null, "-v", "--value");

      if(string.IsNullOrEmpty(qUrl))
        CommandLine.ErrorExit("\nYou must supply at least a queue URL." +
          "\nRun the command with no arguments to see help.");

      // Create the Amazon SQS client
      var sqsClient = new AmazonSQSClient();

      // In the case of one command-line argument, just show the attributes for the
queue
      if(parsedArgs.Count == 1)
        await ShowAllAttributes(sqsClient, qUrl);

      // Otherwise, attempt to update the given queue attribute with the given value
      else
      {
        // Check to see if the attribute is valid
        if(ValidAttribute(attribute))
        {
          // Perform the update and then show all the attributes of the queue
          await UpdateAttribute(sqsClient, qUrl, attribute, value);
          await ShowAllAttributes(sqsClient, qUrl);
        }
        else
```

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


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


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


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

```
    //
    // Command-line help
    private static void PrintHelp()
    {
      Console.WriteLine("\nUsage: SQSUpdateQueue -q queue_url [-a attribute -v
value]");
      Console.WriteLine("  -q: The URL of the queue you want to update.");
      Console.WriteLine("  -a: The name of the attribute to update.");
      Console.WriteLine("  -v, --value: The value to assign to the attribute.");
    }
  }


  // = = =
```

```
          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);
```

```
    }
  }

}
```

추가 고려 사항

- `RedrivePolicy` 속성을 업데이트하려면 운영 체제에 맞게 전체 값을 인용하고 키/값 페어의 따옴표는 이스케이프 처리해야 합니다.

  예를 들어 Windows에서는 값이 다음과 비슷한 방식으로 구성됩니다.

  ```
  "{\"deadLetterTargetArn\":\"DEAD_LETTER-QUEUE-ARN\",\"maxReceiveCount\":\"10\"}"
  ```

Amazon SQS 대기열 삭제

이 예제에서는를 사용하여 Amazon SQS 대기열 AWS SDK for .NET 을 삭제하는 방법을 보여줍니다. 애플리케이션은 대기열을 삭제하고 대기열이 없어질 때까지 지정된 시간까지 기다린 다음 나머지 대기열의 목록을 표시합니다.

명령줄 인수를 제공하지 않으면 애플리케이션은 단순히 기존 대기열의 목록을 표시합니다.

다음 섹션에서는 이 예제의 코드 조각을 제공합니다. 예제의 전체 코드는 그 뒤에 표시되며, 그대로 빌드하고 실행할 수 있습니다.

주제

- 대기열 삭제
- 대기열이 없어질 때까지 대기
- 기존 대기열 목록 표시
- 전체 코드
- 추가 고려 사항

대기열 삭제

다음 코드 조각은 지정된 대기열 URL로 식별되는 대기열을 삭제합니다.

이 주제의 끝 부분에 있는 예제에서는 사용 중인 이 코드 조각을 보여줍니다.

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

대기열이 없어질 때까지 대기

다음 코드 조각은 삭제 프로세스가 완료될 때까지 기다립니다. 이 과정은 60초 정도 걸릴 수 있습니다.

이 주제의 끝 부분에 있는 예제에서는 사용 중인 이 코드 조각을 보여줍니다.

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

기존 대기열 목록 표시

다음 코드 조각은 SQS 클라이언트 리전의 기존 대기열 목록을 보여줍니다.

이 주제의 끝 부분에 있는 예제에서는 사용 중인 이 코드 조각을 보여줍니다.

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

전체 코드

이 섹션에는 이 예제에 대한 관련 참조와 전체 코드가 나와 있습니다.

SDK 레퍼런스

NuGet 패키지:

- AWSSDK.SQS

프로그래밍 요소:

- 네임스페이스 Amazon.SQS

  클래스 AmazonSQSClient
- 네임스페이스 Amazon.SQS.Model

  클래스 ListQueuesResponse

코드

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

```
namespace SQSDeleteQueue
{
  // = = =
```

```
    }


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

추가 고려 사항

- `DeleteQueueAsync` API 직접 호출은 삭제하려는 대기열이 DLQ(Dead Letter Queue)로 사용되고 있는지 확인하지 않습니다. 좀 더 정교한 절차를 통해 이를 확인할 수 있습니다.

- Amazon SQS 콘솔에서도 대기열 목록과 이 예제의 결과를 확인할 수 있습니다.

Amazon SQS 메시지 전송

이 예제에서는 AWS SDK for .NET 를 사용하여 프로그래밍 방식으로 또는 Amazon SQS 콘솔을 사용하여 생성할 수 있는 Amazon SQS 대기열로 메시지를 전송하는 방법을 보여줍니다. 애플리케이션은 대기열에 메시지 하나를 전송한 다음 메시지 배치를 전송합니다. 그러면 애플리케이션은 사용자 입력을 기다립니다. 사용자 입력은 대기열에 보낼 추가 메시지나 애플리케이션 종료 요청일 수 있습니다.

이 예제와 메시지 수신에 대한 다음 예제를 함께 사용하여 Amazon SQS의 메시지 흐름을 확인할 수 있습니다.

다음 섹션에서는 이 예제의 코드 조각을 제공합니다. 예제의 전체 코드는 그 뒤에 표시되며, 그대로 빌드하고 실행할 수 있습니다.

주제

- 메시지 전송
- 메시지 배치 전송
- 대기열에서 모든 메시지 삭제
- 전체 코드
- 추가 고려 사항

메시지 전송

다음 코드 조각은 지정된 대기열 URL로 식별되는 대기열에 메시지를 전송합니다.

이 주제의 끝 부분에 있는 예제에서는 사용 중인 이 코드 조각을 보여줍니다.

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

메시지 배치 전송

다음 코드 조각은 지정된 대기열 URL로 식별되는 대기열에 메시지 배치를 전송합니다.

이 주제의 끝 부분에 있는 예제에서는 사용 중인 이 코드 조각을 보여줍니다.

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

대기열에서 모든 메시지 삭제

다음 코드 조각은 지정된 대기열 URL로 식별되는 대기열에서 모든 메시지를 삭제합니다. 이를 대기열 비우기라고도 합니다.

이 주제의 끝 부분에 있는 예제에서는 사용 중인 이 코드 조각을 보여줍니다.

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

전체 코드

이 섹션에는 이 예제에 대한 관련 참조와 전체 코드가 나와 있습니다.

SDK 레퍼런스

NuGet 패키지:

- AWSSDK.SQS

프로그래밍 요소:

- 네임스페이스 Amazon.SQS

  클래스 AmazonSQSClient
- 네임스페이스 Amazon.SQS.Model

  클래스 PurgeQueueResponse

  클래스 SendMessageBatchResponse

  클래스 SendMessageResponse

  클래스 SendMessageBatchRequestEntry

  클래스 SendMessageBatchResultEntry

코드

```
using System;
using System.Collections.Generic;
```

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

namespace SQSSendMessages
{
  // = = =
```

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

추가 고려 사항

- 허용되는 문자를 비롯한 다양한 메시지 제한 사항에 대한 자세한 내용은 Amazon Simple Queue Service 개발자 안내서의 메시지 관련 할당량을 참조하세요.

- 메시지는 삭제되거나 대기열이 제거될 때까지 대기열에 남아 있습니다. 애플리케이션이 메시지를 수신한 경우 해당 메시지가 대기열에 남아 있더라도 대기열에 표시되지 않습니다. 제한 시간 초과에 대한 자세한 내용은 Amazon SQS 제한 시간 초과를 참조하세요.

- 메시지 본문 외에도 메시지에 속성을 추가할 수 있습니다. 자세한 내용은 메시지 메타데이터를 참조하세요.

Amazon SQS 메시지 수신

이 예제에서는를 사용하여 프로그래밍 방식으로 또는 Amazon SQS 콘솔을 사용하여 생성할 수 있는 Amazon SQS 대기열에서 메시지를 AWS SDK for .NET 수신하는 방법을 보여줍니다. 애플리케이션은 대기열에서 단일 메시지를 읽고 메시지를 처리한 다음(이 경우 콘솔에 메시지 본문을 표시) 대기열에서 메시지를 삭제합니다. 애플리케이션은 사용자가 키보드로 키를 입력할 때까지 이 단계를 반복합니다.

이 예제와 메시지 전송에 대한 이전 예제를 함께 사용하여 Amazon SQS의 메시지 흐름을 확인할 수 있습니다.

다음 섹션에서는 이 예제의 코드 조각을 제공합니다. 예제의 전체 코드는 그 뒤에 표시되며, 그대로 빌드하고 실행할 수 있습니다.

주제

- 메시지 수신
- 메시지 삭제
- 전체 코드
- 추가 고려 사항

메시지 수신

다음 코드 조각은 지정된 대기열 URL로 식별되는 대기열에서 메시지를 수신합니다.

이 주제의 끝 부분에 있는 예제에서는 사용 중인 이 코드 조각을 보여줍니다.

```
//
// 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.)
  });
}
```

메시지 삭제

다음 코드 조각은 지정된 대기열 URL로 식별되는 대기열에서 메시지를 삭제합니다.

이 주제의 끝 부분에 있는 예제에서는 사용 중인 이 코드 조각을 보여줍니다.

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

전체 코드

이 섹션에는 이 예제에 대한 관련 참조와 전체 코드가 나와 있습니다.

SDK 레퍼런스

NuGet 패키지:

- AWSSDK.SQS

프로그래밍 요소:

- 네임스페이스 Amazon.SQS

  클래스 AmazonSQSClient
- 네임스페이스 Amazon.SQS.Model

  클래스 ReceiveMessageRequest

  클래스 ReceiveMessageResponse

코드

```
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);
    }
  }
}
```

추가 고려 사항

- 긴 폴링을 지정하기 위해 이 예제에서는 `ReceiveMessageAsync` 메서드를 호출할 때마다 `WaitTimeSeconds` 속성을 사용합니다.

  대기열을 생성하거나 업데이트할 때 `ReceiveMessageWaitTimeSeconds` 속성을 사용하여 대기열의 모든 메시지에 대해 긴 폴링을 지정할 수도 있습니다.

짧은 폴링과 긴 폴링에 대한 자세한 내용은 Amazon Simple Queue Service 개발자 안내서의 짧은 폴링 및 긴 폴링을 참조하세요.

- 메시지 처리 중에 수신 핸들을 사용하여 메시지 제한 시간 초과를 변경할 수 있습니다. 이를 수행하는 방법에 대한 자세한 내용은 AmazonSQSClient 클래스의 `ChangeMessageVisibilityAsync` 메서드를 참조하세요.

- 조건 없이 `DeleteMessageAsync` 메서드를 호출하면 제한 시간 설정에 상관없이 대기열에서 메시지가 제거됩니다.

AWS Lambda 컴퓨팅 서비스에 사용

는 서버를 프로비저닝하거나 관리하지 않고도 코드를 실행할 수 AWS Lambda있도록 AWS SDK for .NET 지원합니다. 자세한 내용은 AWS Lambda 제품 페이지 및 AWS Lambda 개발자 안내서, 특히 C#을 사용한 작업 섹션을 참조하세요.

API

는에 대한 APIs AWS SDK for .NET 제공합니다 AWS Lambda. API를 사용하면 함수, 트리거, 이벤트와 같은 Lambda 기능을 사용할 수 있습니다. 전체 API 세트를 보려면 AWS SDK for .NET API 참조의 Lambda를 참조하세요.

Lambda API는 NuGet 패키지를 통해 제공됩니다.

사전 조건

시작하기 전에 환경을 설정하고 프로젝트를 구성했는지 확인합니다. 또한 SDK 사용의 정보를 검토합니다.

추가 정보

.NET Aspire를 AWS Lambda 통해를 사용하여 개발하는 방법에 대한 .NET Aspire AWS 와 통합 자세한 내용은 섹션을 참조하세요.

주제

주제

- 주석을 사용하여 AWS Lambda 함수 작성

주석을 사용하여 AWS Lambda 함수 작성

Lambda 함수를 작성할 때 다른 작업 중에서도 대량의 핸들러 코드를 작성하고 AWS CloudFormation 템플릿을 업데이트해야 하는 경우가 있습니다. Lambda 주석은 .NET 6 Lambda 함수에 대한 이러한 부담을 덜어주는 프레임워크로, C #에서 Lambda 작성 경험이 더욱 자연스럽게 느껴지도록 합니다.

Lambda 주석 프레임워크를 사용할 때의 이점을 예로 들어 두 개의 숫자를 추가하는 다음 코드 조각을 고려해 보세요.

Lambda 주석 없음

```
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" } }
```

```
        };
    }
}
```

Lambda 주석 포함

```
public class Functions
{
    [LambdaFunction]
    [RestApi("/plus/{x}/{y}")]
    public int Plus(int x, int y)
    {
        return x + y;
    }
}
```

예제에서 볼 수 있듯이 Lambda 주석을 사용하면 특정 보일러 플레이트 코드가 필요하지 않습니다.

프레임워크 사용 방법 및 추가 정보에 대한 자세한 내용은 다음 리소스를 참조하세요.

- Lambda 주석의 API 및 속성에 대한 설명서는 GitHub README를 참조하세요.
- Lambda 주석에 대한 블로그 게시물.
- `Amazon.Lambda.Annotations` NuGet 패키지.
- GitHub의 사진 자산 관리 프로젝트 구체적으로는 프로젝트 README의 PamApiAnnotations 폴더와 Lambda 주석에 대한 참조를 참조하세요.

> **Note**
>
> 앞의 예제는의 V3에만 해당됩니다 AWS SDK for .NET. SDK의 V4(최신 버전)에서 예제를 사용하는 경우의 정보에 따라 조정해야 할 수 있습니다버전 4로 마이그레이션.

에 대한 상위 수준 라이브러리 및 프레임워크 AWS SDK for .NET

다음 섹션에는 핵심 SDK 기능에 포함되지 않은 개요 수준 라이브러리 및 프레임워크에 대한 정보가 포함되어 있습니다. 이러한 라이브러리와 프레임워크는 핵심 SDK 기능을 사용하여 특정 작업을 용이하게 하는 기능을 만듭니다.

를 처음 사용하는 경우 먼저 간단한 애플리케이션 생성 주제를 확인할 AWS SDK for .NET수 있습니다. SDK에 대한 소개를 제공합니다.

시작하기 전에 환경을 설정하고 프로젝트를 구성했는지 확인합니다. 또한 SDK 사용의 정보를 검토합니다.

주제

- AWS .NET용 메시지 처리 프레임워크
- 에서 .NET Aspire AWS 와 통합 AWS SDK for .NET

AWS .NET용 메시지 처리 프레임워크

.NET용 AWS 메시지 처리 프레임워크는 Amazon Simple Queue Service(SQS), Amazon Simple Notification Service(SNS) 및 Amazon EventBridge와 같은 AWS 서비스를 사용하는 .NET 메시지 처리 애플리케이션의 개발을 간소화하는 AWS네이티브 프레임워크입니다. 이 프레임워크는 개발자가 작성해야 하는 상용구 코드의 양을 줄여주므로 메시지를 게시하고 소비할 때 비즈니스 로직에 집중할 수 있습니다. 프레임워크가 개발을 간소화하는 방법에 대한 자세한 내용은 블로그 게시물 .NET용 AWS 메시지 처리 프레임워크 소개(미리 보기)를 참조하세요. 특히 첫 번째 부분에서는 하위 수준 API 호출을 사용하는 것과 프레임워크를 사용하는 것의 차이점을 보여주는 데모를 제공합니다.

메시지 처리 프레임워크는 다음과 같은 활동 및 기능을 지원합니다.

- SQS에 메시지를 보내고 SNS 및 EventBridge에 이벤트를 게시합니다.
- 일반적으로 백그라운드 서비스에서 사용되는 장기 실행 폴러를 사용하여 SQS에서 메시지를 수신하고 처리합니다. 여기에는 메시지를 처리하는 동안 다른 클라이언트가 메시지를 처리하지 못하도록 제한 시간 초과를 관리하는 것이 포함됩니다.
- AWS Lambda 함수의 메시지 처리.
- FIFO(first-in-first-out) SQS 대기열 및 SNS 주제.
- 로깅을 위한 OpenTelemetry.

이러한 활동 및 기능에 대한 자세한 내용은 블로그 게시물의 기능 섹션과 아래에 나열된 주제를 참조하세요.

시작하기 전에 환경을 설정하고 프로젝트를 구성했는지 확인합니다. 또한 SDK 사용의 정보를 검토합니다.

추가 리소스

- NuGet.org의 `AWS.Messaging` 패키지입니다.
- API 참조입니다.

- 의 GitHub 리포지토리에 있는 README 파일 https://github.com/aws/aws-dotnet-messaging/
- Microsoft의 .NET 종속성 주입.
- Microsoft의 .NET 일반 호스트.

주제

- .NET용 AWS 메시지 처리 프레임워크 시작하기
- .NET용 AWS 메시지 처리 프레임워크를 사용하여 메시지 게시
- .NET용 메시지 처리 프레임워크를 사용하여 AWS 메시지 사용
- .NET용 AWS 메시지 처리 프레임워크에서 FIFO 사용
- .NET용 AWS 메시지 처리 프레임워크에 대한 로깅 및 개방형 텔레메트리
- .NET용 AWS 메시지 처리 프레임워크 사용자 지정
- .NET용 AWS 메시지 처리 프레임워크의 보안

.NET용 AWS 메시지 처리 프레임워크 시작하기

시작하기 전에 환경을 설정하고 프로젝트를 구성했는지 확인합니다. 또한 SDK 사용의 정보를 검토합니다.

이 주제에서는 메시지 처리 프레임워크 사용을 시작하는 데 도움이 되는 정보를 제공합니다. 사전 조건 및 구성 정보 외에도 일반적인 시나리오를 구현하는 방법을 보여주는 자습서가 제공됩니다.

사전 조건 및 구성

- 애플리케이션에 제공하는 자격 증명에는 사용하는 메시징 서비스 및 작업에 대한 적절한 권한이 있어야 합니다. 자세한 내용은 해당 개발자 안내서의 SQS, SNS 및 EventBridge에 대한 보안 주제를 참조하세요. 또한 특정 권한을 설명하는 GitHub의 README 파일 부분을 참조하세요.
- .NET용 AWS 메시지 처리 프레임워크를 사용하려면 프로젝트에 `AWS.Messaging` NuGet 패키지를 추가해야 합니다. 예시:

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

- 프레임워크는 .NET의 종속성 주입(DI) 서비스 컨테이너와 통합됩니다. 애플리케이션을 시작하는 동안를 호출AddAWSMessageBus하여 프레임워크를 구성하여 DI 컨테이너에 추가할 수 있습니다.

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

자습서

이 자습서에서는 .NET용 AWS 메시지 처리 프레임워크를 사용하는 방법을 보여줍니다. API 엔드포인트에서 요청을 수신할 때 Amazon SQS 대기열로 메시지를 보내는 ASP.NET Core Minimal API와 이러한 메시지를 폴링하고 처리하는 장기 실행 콘솔 애플리케이션이라는 두 가지 애플리케이션을 생성합니다.

- 이 자습서의 지침은 .NET CLI를 선호하지만 .NET CLI 또는 Microsoft Visual Studio와 같은 교차 플랫폼 도구를 사용하여이 자습서를 수행할 수 있습니다. 도구에 대한 자세한 내용은 단원을 참조하십시오용 도구 체인 설치 및 구성 AWS SDK for .NET.
- 이 자습서에서는 자격 증명에 `[default]` 프로필을 사용하고 있다고 가정합니다. 또한 Amazon SQS 메시지를 보내고 받기 위한 적절한 권한과 함께 단기 자격 증명을 사용할 수 있다고 가정합니다. 자세한 내용은 를 AWS SDK for .NET 사용하여 인증 AWS 및 SQS 보안 주제를 참조하세요.

> **Note**
>
> 이 자습서를 실행하면 SQS 메시징 비용이 발생할 수 있습니다.

단계

- SQS 대기열 생성
- 게시 애플리케이션 생성 및 실행
- 처리 애플리케이션 생성 및 실행
- 정리

SQS 대기열 생성

이 자습서에서는 메시지를 보내고 받으려면 SQS 대기열이 필요합니다. AWS CLI 또는에 대해 다음 명령 중 하나를 사용하여 대기열을 생성할 수 있습니다 AWS Tools for PowerShell. 다음 프레임워크 구성에서 지정할 수 있도록 반환되는 대기열 URL을 기록해 둡니다.

AWS CLI

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

AWS Tools for PowerShell

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

게시 애플리케이션 생성 및 실행

다음 절차에 따라 게시 애플리케이션을 생성하고 실행합니다.

1. 명령 프롬프트 또는 터미널을 엽니다. .NET 프로젝트를 만들 수 있는 운영 체제 폴더를 찾거나 생성합니다.
2. 해당 폴더에서 다음 명령을 실행하여 .NET 프로젝트를 만듭니다.

   ```
   dotnet new webapi --name Publisher
   ```
3. 새 프로젝트의 폴더로 이동합니다. .NET용 AWS 메시지 처리 프레임워크에 대한 종속성을 추가합니다.

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

   > **Note**
   >
   > 인증 AWS IAM Identity Center 에를 사용하는 경우 `AWSSDK.SSO` 및 도 추가해야 합니다`AWSSDK.SSOOIDC`.
4. 의 코드를 다음 코드`Program.cs`로 바꿉니다.

   ```
   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. 다음 명령을 실행합니다. 그러면 API를 탐색하고 테스트할 수 있는 Swagger UI가 있는 브라우저 창이 열립니다.

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

6. /greeting 엔드포인트를 열고 시도를 선택합니다.
7. 메시지의 senderName 및 greeting 값을 지정하고 실행을 선택합니다. 그러면 API가 호출되어 SQS 메시지가 전송됩니다.

처리 애플리케이션 생성 및 실행

다음 절차에 따라 처리 애플리케이션을 생성하고 실행합니다.

1. 명령 프롬프트 또는 터미널을 엽니다. .NET 프로젝트를 만들 수 있는 운영 체제 폴더를 찾거나 생성합니다.
2. 해당 폴더에서 다음 명령을 실행하여 .NET 프로젝트를 만듭니다.

```
dotnet new console --name Handler
```

3. 새 프로젝트의 폴더로 이동합니다. .NET용 AWS 메시지 처리 프레임워크에 대한 종속성을 추가합니다. 또한 Microsoft.Extensions.Hosting 패키지를 추가하여 .NET 일반 호스트를 통해 프레임워크를 구성할 수 있습니다.

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

> **Note**
>
> 인증 AWS IAM Identity Center 에를 사용하는 경우 AWSSDK.SSO 및 도 추가해야 합니다AWSSDK.SSOOIDC.

4. 의 코드를 다음 코드Program.cs로 바꿉니다.

```
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. 다음 명령을 실행합니다. 이렇게 하면 장기 실행 폴러가 시작됩니다.

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

시작 직후 애플리케이션은이 자습서의 첫 부분에서 전송된 메시지를 수신하고 다음 메시지를 기록합니다.

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

6. 폴러를 중지Ctrl+C하려면를 누릅니다.

정리

AWS CLI 또는에 대해 다음 명령 중 하나를 사용하여 대기열 AWS Tools for PowerShell 을 삭제합니다.

AWS CLI

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

AWS Tools for PowerShell

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

.NET용 AWS 메시지 처리 프레임워크를 사용하여 메시지 게시

.NET용 AWS 메시지 처리 프레임워크는 하나 이상의 메시지 유형을 게시하거나, 하나 이상의 메시지 유형을 처리하거나, 동일한 애플리케이션에서 둘 다 수행할 수 있도록 지원합니다.

다음 코드는 다양한 메시지 유형을 다양한 AWS 서비스에 게시하는 애플리케이션의 구성을 보여줍니다.

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

시작 중에 프레임워크를 등록했으면 일반를 코드IMessagePublisher에 주입합니다. PublishAsync 메서드를 호출하여 위에 구성된 메시지 유형을 게시합니다. 일반 게시자는 유형에 따라 메시지를 라우팅할 대상을 결정합니다.

다음 예제에서 ASP.NET MVC 컨트롤러는 사용자로부터 ChatMessage 메시지와 OrderInfo 이벤트를 모두 수신한 다음 각각 Amazon SQS 및 Amazon SNS에 게시합니다. 두 메시지 유형 모두 위에 구성된 일반 게시자를 사용하여 게시할 수 있습니다.

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

적절한 처리 로직으로 메시지를 라우팅하기 위해 프레임워크는 메시지 유형 식별자라는 메타데이터를 사용합니다. 기본적으로 이는 어셈블리 이름을 포함하여 메시지의 .NET 유형의 전체 이름입니다. 메시지를 보내고 처리하는 경우 프로젝트 간에 메시지 객체의 정의를 공유하는 경우이 메커니즘이 잘 작동합니다. 그러나 메시지가 다른 네임스페이스에서 재정의되거나 다른 프레임워크 또는 프로그래밍 언어와 메시지를 교환하는 경우 메시지 유형 식별자를 재정의해야 할 수 있습니다.

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

서비스별 게시자

위에 표시된 예제에서는 구성된 메시지 유형에 따라 지원되는 모든 AWS 서비스에 게시할 수 `IMessagePublisher`있는 일반를 사용합니다. 또한 프레임워크는 Amazon SQS, Amazon SNS 및 Amazon EventBridge에 대한 서비스별 게시자를 제공합니다. 이러한 특정 게시자는 해당 서비스에만 적용되는 옵션을 노출하며, `ISQSPublisher`, `ISNSPublisher`및 유형을 사용하여 삽입할 수 있습니다`IEventBridgePublisher`.

예를 들어 SQS FIFO 대기열로 메시지를 전송할 때는 적절한 메시지 그룹 ID를 설정해야 합니다. 다음 코드는 `ChatMessage` 예제를 다시 보여 주지만 이제 `ISQSPublisher`를 사용하여 SQS별 옵션을 설정합니다.

```
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();
    }
}
```

SNS 및 EventBridge에 대해 `IEventBridgePublisher` 각각 `ISNSPublisher` 및를 사용하여 동일한 작업을 수행할 수 있습니다.

```
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>
});
```

기본적으로 지정된 유형의 메시지는 미리 구성된 대상으로 전송됩니다. 그러나 메시지별 게시자를 사용하여 단일 메시지의 대상을 재정의할 수 있습니다. 메시지를 게시하는 데 사용되는 기본 AWS SDK for .NET 클라이언트를 재정의할 수도 있습니다. 이는 대상에 따라 역할 또는 자격 증명을 변경해야 하는 다중 테넌트 애플리케이션에서 유용할 수 있습니다.

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

.NET용 메시지 처리 프레임워크를 사용하여 AWS 메시지 사용

.NET용 AWS 메시지 처리 프레임워크를 사용하면 프레임워크 또는 메시징 서비스 중 하나를 사용하여 게시된 메시지를 사용할 수 있습니다. 메시지는 다양한 방식으로 사용할 수 있으며, 그 중 일부는 아래에 설명되어 있습니다.

메시지 핸들러

메시지를 사용하려면 처리하려는 각 메시지 유형에 대한 `IMessageHandler` 인터페이스를 사용하여 메시지 핸들러를 구현합니다. 메시지 유형과 메시지 핸들러 간의 매핑은 프로젝트 시작 시 구성됩니다.

```
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();
```

다음 코드는 메시지에 대한 샘플 ChatMessage 메시지 핸들러를 보여줍니다.

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

외부에는 프레임워크에서 사용하는 메타데이터가 MessageEnvelope 포함되어 있습니다. message 속성은 메시지 유형입니다(이 경우 ChatMessage).

MessageProcessStatus.Success() 로 돌아가 메시지가 성공적으로 처리되었으며 프레임워크가 Amazon SQS 대기열에서 메시지를 삭제함을 나타낼 수 있습니다. MessageProcessStatus.Failed()를 반환할 때 메시지는 다시 처리하거나 구성된 경우 배달 못한 편지 대기열로 이동할 수 있는 대기열에 남아 있습니다.

장기 실행 프로세스에서 메시지 처리

SQS 대기열 URLAddSQSPoller로를 호출하여 대기열을 BackgroundService 지속적으로 폴링하고 메시지를 처리하는 장기 실행을 시작할 수 있습니다.

```
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();
```

SQS 메시지 폴러 구성

를 호출할 `SQSMessagePollerOptions` 때에서 SQS 메시지 폴러를 구성할 수 있습니다`AddSQSPoller`.

- `MaxNumberOfConcurrentMessages` - 동시에 처리할 대기열의 최대 메시지 수입니다. 기본값은 10입니다.
- `WaitTimeSeconds` - `ReceiveMessage` SQS 호출이 반환되기 전에 대기열에 메시지가 도착할 때까지 기다리는 기간(초)입니다. 메시지를 사용할 수 있는 경우 호출은 보다 빨리 반환됩니다`WaitTimeSeconds`. 기본값은 20입니다.

메시지 가시성 제한 시간 처리

SQS 메시지에는 표시 제한 시간이 있습니다. 한 소비자가 지정된 메시지를 처리하기 시작하면 대기열에 남아 있지만 두 번 이상 처리하지 않도록 다른 소비자에게 숨겨집니다. 메시지가 다시 표시되기 전에 처리 및 삭제되지 않으면 다른 소비자가 동일한 메시지를 처리하려고 할 수 있습니다.

프레임워크는 현재 처리 중인 메시지의 제한 시간 초과를 추적하고 연장하려고 시도합니다. 를 호출할 `SQSMessagePollerOptions` 때에서이 동작을 구성할 수 있습니다`AddSQSPoller`.

- `VisibilityTimeout` - 수신된 메시지가 후속 검색 요청에서 숨겨지는 초 단위 기간입니다. 기본값은 30입니다.
- `VisibilityTimeoutExtensionThreshold` - 메시지의 제한 시간 초과가 만료된 후이 몇 초 내에 있는 경우 프레임워크는 제한 시간 초과를 (`VisibilityTimeout`초 더) 연장합니다. 기본값은 5입니다.
- `VisibilityTimeoutExtensionHeartbeatInterval` - 프레임워크가 만료 후 몇 초 이내에 메시지를 확인한 다음 표시 제한 시간을 연장하는 `VisibilityTimeoutExtensionThreshold` 초 단위의 빈도입니다. 기본값은 1입니다.

다음 예제에서 프레임워크는 1초마다 처리 중인 메시지를 확인합니다. 이러한 메시지가 다시 표시된 후 5초 이내에 표시되는 경우 프레임워크는 각 메시지의 표시 제한 시간을 자동으로 30초 더 연장합니다.

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

AWS Lambda 함수의 메시지 처리

SQS와 Lambda의 통합과 함께 .NET용 AWS 메시지 처리 프레임워크를 사용할 수 있습니다. https://docs.aws.amazon.com/lambda/latest/dg/with-sqs.html 이는 `AWS.Messaging.Lambda` 패키지에서 제공합니다. 시작하려면 README를 참조하세요.

.NET용 AWS 메시지 처리 프레임워크에서 FIFO 사용

메시지 순서 지정 및 메시지 중복 제거가 중요한 사용 사례의 경우 .NET용 AWS 메시지 처리 프레임워크는 first-in-first-out(FIFO) Amazon SQS 대기열 및 Amazon SNS 주제를 지원합니다.

게시

메시지를 FIFO 대기열 또는 주제에 게시할 때는 메시지가 속한 그룹을 지정하는 메시지 그룹 ID를 설정해야 합니다. 그룹 내의 메시지는 순서대로 처리됩니다. SQS별 및 SNS별 메시지 게시자에서 이를 설정할 수 있습니다.

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

구독

FIFO 대기열의 메시지를 처리할 때 프레임워크는 각 `ReceiveMessages` 호출에 대해 수신된 순서대로 지정된 메시지 그룹 내의 메시지를 처리합니다. 로 끝나는 대기열로 구성된 경우 프레임워크는이 작업 모드로 자동으로 전환됩니다`.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();
```

.NET용 AWS 메시지 처리 프레임워크에 대한 로깅 및 개방형 텔레메트리

.NET용 AWS 메시지 처리 프레임워크는 OpenTelemetry가 프레임워크에서 게시하거나 처리하는 각 메시지에 대한 추적을 로깅하도록 계측됩니다. 이는 `AWS.Messaging.Telemetry.OpenTelemetry` 패키지에서 제공합니다. 시작하려면 README를 참조하세요.

> **Note**
>
> 로깅과 관련된 보안 정보는 섹션을 참조하세요.NET용 AWS 메시지 처리 프레임워크의 보안.

.NET용 AWS 메시지 처리 프레임워크 사용자 지정

.NET용 AWS 메시지 처리 프레임워크는 세 가지 "계층"으로 메시지를 빌드, 전송 및 처리합니다.

1. 가장 바깥쪽 계층에서 프레임워크는 서비스와 관련된 AWS네이티브 요청 또는 응답을 빌드합니다. 예를 들어 Amazon SQS를 사용하면 `SendMessage` 요청을 빌드하고 서비스에서 정의한 `Message` 객체와 함께 작동합니다.
2. SQS 요청 및 응답 내에서 프레임워크는 `MessageBody` 요소(또는 Amazon SNS`Message`의 경우 또는 Amazon EventBridge의 `Detail` 경우)를 JSON 형식의 CloudEvent로 설정합니다. 여기에는 메시지를 처리할 때 `MessageEnvelope` 객체에서 액세스할 수 있는 프레임워크에서 설정한 메타데이터가 포함됩니다.
3. 가장 안쪽 계층에서 CloudEvent JSON 객체 내부의 `data` 속성에는 메시지로 전송되거나 수신된 .NET 객체의 JSON 직렬화가 포함됩니다.

```
{
    "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>"
}
```

메시지 봉투를 구성하고 읽는 방법을 사용자 지정할 수 있습니다.

- `"id"`는 메시지를 고유하게 식별합니다. 기본적으로 새 GUID로 설정되지만 자체 GUID를 구현`IMessageIdGenerator`하고 이를 DI 컨테이너에 주입하여 재정의할 수 있습니다.
- `"type"`는 메시지가 핸들러로 라우팅되는 방식을 제어합니다. 기본적으로 메시지에 해당하는 .NET 유형의 전체 이름을 사용합니다. `AddSQSPublisher`, `AddSNSPublisher`또는를 통해 메시지 유형을 대상에 매핑할 때 `messageTypeIdentifier` 파라미터를 통해 이를 재정의할 수 있습니다`AddEventBridgePublisher`.
- `"source"`는 메시지를 전송한 시스템 또는 서버를 나타냅니다.

  - 이는에서 게시하는 경우 함수 이름 AWS Lambda, Amazon ECS에서 게시하는 경우 클러스터 이름 및 작업 ARN, Amazon EC2에서 게시하는 경우 인스턴스 ID, 그렇지 않으면 폴백 값 입니다`/aws/messaging`.
  - 를 통해 `AddMessageSource` 또는 `AddMessageSourceSuffix`에서 이를 재정의할 수 있습니다`MessageBusBuilder`.
- `"time"`를 UTC의 현재 DateTime으로 설정합니다. 이는 자체를 구현`IDateTimeHandler`하고 이를 DI 컨테이너에 주입하여 재정의할 수 있습니다.
- `"data"` 에는 메시지로 전송되거나 수신된 .NET 객체의 JSON 표현이 포함되어 있습니다.
  - `ConfigureSerializationOptions`의를 `MessageBusBuilder` 사용하면 메시지를 직렬화 및 역직렬화할 때 사용할를 구성할 `System.Text.Json.JsonSerializerOptions` 수 있습니다.
  - 프레임워크가 빌드되면 추가 속성을 주입하거나 메시지 봉투를 변환하려면 `AddSerializationCallback`의를 통해 해당 속성을 구현`ISerializationCallback`하고 등록할 수 있습니다`MessageBusBuilder`.

.NET용 AWS 메시지 처리 프레임워크의 보안

.NET용 AWS 메시지 처리 프레임워크는와 통신하기 AWS SDK for .NET 위해에 의존합니다 AWS. 의 보안에 대한 자세한 내용은 단원을 AWS SDK for .NET참조하십시오이 AWS 제품 또는 서비스에 대한 보안.

보안을 위해 프레임워크는 사용자가 보낸 데이터 메시지를 로깅하지 않습니다. 디버깅을 위해이 기능을 활성화하려면 다음과 같이 메시지 버스`EnableDataMessageLogging()`에서를 호출해야 합니다.

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

잠재적 보안 문제가 발견되면 보안 정책에서 보고 정보를 참조하세요.

에서 .NET Aspire AWS 와 통합 AWS SDK for .NET

.NET Aspire는 클라우드 지원 애플리케이션을 구축하는 새로운 방법입니다. 특히 분산 애플리케이션의 구성 요소를 실행, 연결 및 디버깅할 로컬 환경에 대한 오케스트레이션을 제공합니다. 클라우드 지원 애플리케이션의 내부 개발 루프를 개선하기 위해 .NET 애플리케이션을 AWS 리소스에 연결하기 위해 .NET Aspire와의 통합이 생성되었습니다. 이러한 통합은 Aspire.Hosting.AWS NuGet 패키지를 통해 사용할 수 있습니다.

다음과 같은 .NET Aspire 통합을 사용할 수 있습니다.

- 를 통해 AWS 리소스를 프로비저닝할 수 있는 기능입니다CloudFormation. 이 통합은 .NET Aspire AppHost 프로젝트 내에서 활용됩니다.

  자세한 내용은 블로그 게시물 Integrating AWS with .NET Aspire를 참조하세요.
- 를 설치, 구성 및 Amazon DynamoDB 로컬 AWS SDK for .NET 에 연결합니다. 이 통합은 .NET Aspire AppHost 프로젝트 내에서 활용됩니다.

  자세한 내용은 블로그 게시물 Integrating AWS with .NET Aspire를 참조하세요.
- AWS Lambda 함수에 대한 로컬 개발 환경을 활성화합니다. 이 통합은 .NET Aspire AppHost 프로젝트 내에서 활용됩니다.

  자세한 내용은 블로그 게시물 Building and Debugging .NET Lambda applications with .NET Aspire (Part 1) 및 Building and Debugging .NET Lambda applications with .NET Aspire (Part 2)를 참조하세요.

  > **Note**
  >
  > 이 시험판 설명서는 프리뷰 릴리즈의 기능에 관한 것입니다. 내용은 변경될 수 있습니다.

  이 기능은 미리 보기이므로 미리 보기 기능을 옵트인해야 합니다. 이 미리 보기 기능과 옵트인 방법에 대한 자세한 내용은 GitHub의 개발 트래커 문제를 참조하세요.

추가 정보

Aspire.Hosting.AWS에서 사용할 수 있는 통합을 사용하는 방법에 대한 추가 정보와 세부 정보는 다음 리소스를 참조하세요.

- 블로그 게시물 Integrating AWS with .NET Aspire.
- 블로그 게시물 Building and Debugging .NET Lambda applications with .NET Aspire(1부) 및 Building and Debugging .NET Lambda applications with .NET Aspire(2부)
- GitHub의 integrations-on-dotnet-aspire-for-aws 리포지토리입니다.
- Aspire.Hosting.AWS NuGet 패키지에 대한 세부 README입니다.

기타 AWS 서비스 및 구성 지원

는 이전 단원에서 설명한 서비스 외에도 AWS 서비스를 AWS SDK for .NET 지원합니다. 지원되는 모든 서비스의 API에 대한 자세한 내용은 AWS SDK for .NET API 참조를 참조하세요.

는 개별 AWS 서비스의 네임스페이스 외에도 다음 APIs AWS SDK for .NET 도 제공합니다.

영역	설명	리소스
AWS 지원	AWS 지원 사례 및 Trusted Advisor 기능에 대한 프로그래밍 방식 액세스.	Amazon.AWSSupport 및 Amazon.AWSSupport.Model을 참조하십시오.
일반	헬퍼 클래스 및 열거입니다.	Amazon 및 Amazon.Util을 참조하십시오.

SDK for .NET (v4) 코드 예제

이 주제의 코드 예제에서는 AWS SDK for .NET (v4)를와 함께 사용하는 방법을 보여줍니다 AWS.

기본 사항은 서비스 내에서 필수 작업을 수행하는 방법을 보여주는 코드 예제입니다.

작업은 대규모 프로그램에서 발췌한 코드이며 컨텍스트에 맞춰 실행해야 합니다. 작업은 개별 서비스 함수를 직접 호출하는 방법을 보여주며, 관련 시나리오의 컨텍스트에 맞는 작업을 볼 수 있습니다.

시나리오는 동일한 서비스 내에서 또는 다른 AWS 서비스와 결합된 상태에서 여러 함수를 직접적으로 호출하여 특정 태스크를 수행하는 방법을 보여주는 코드 예제입니다.

일부 서비스에는 서비스와 관련된 라이브러리 또는 함수를 활용하는 방법을 보여주는 추가 예제 범주가 포함되어 있습니다.

서비스

- SDK for .NET (v4)를 사용한 Aurora 예제
- SDK for .NET (v4)를 사용한 Auto Scaling 예제
- SDK for .NET (v4)를 사용한 Amazon Bedrock 예제
- SDK for .NET (v4)를 사용한 Amazon Bedrock 런타임 예제
- CloudFormation SDK for .NET (v4)를 사용한 예제
- SDK for .NET (v4)를 사용한 CloudWatch 예제
- SDK for .NET (v4)를 사용한 CloudWatch Logs 예제
- SDK for .NET (v4)를 사용한 Amazon Cognito 자격 증명 공급자 예제
- AWS Control Tower SDK for .NET (v4)를 사용한 예제
- SDK for .NET (v4)를 사용한 DynamoDB 예제
- SDK for .NET (v4)를 사용한 Amazon EC2 예제
- SDK for .NET (v4)를 사용한 Amazon ECS 예제
- AWS IoT SDK for .NET (v4)를 사용한 예제
- AWS IoT data SDK for .NET (v4)를 사용한 예제
- SDK for .NET (v4)를 사용한 Amazon Redshift 예제
- SDK for .NET (v4)를 사용한 Amazon S3 예제

SDK for .NET (v4)를 사용한 Aurora 예제

다음 코드 예제에서는 Aurora에서 AWS SDK for .NET (v4)를 사용하여 작업을 수행하고 일반적인 시나리오를 구현하는 방법을 보여줍니다.

기본 사항은 서비스 내에서 필수 작업을 수행하는 방법을 보여주는 코드 예제입니다.

작업은 대규모 프로그램에서 발췌한 코드이며 컨텍스트에 맞춰 실행해야 합니다. 작업은 개별 서비스 함수를 직접적으로 호출하는 방법을 보여주며 관련 시나리오의 컨텍스트에 맞는 작업을 볼 수 있습니다.

각 예시에는 전체 소스 코드에 대한 링크가 포함되어 있으며, 여기에서 컨텍스트에 맞춰 코드를 설정하고 실행하는 방법에 대한 지침을 찾을 수 있습니다.

주제

- 시작하기
- 기본 사항
- 작업

시작하기

Hello Aurora

다음 코드 예제에서는 Aurora를 사용하여 시작하는 방법을 보여줍니다.

SDK for .NET (v4)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

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

- API 세부 정보는 AWS SDK for .NET API 참조의 DescribeDBClusters를 참조하세요.

기본 사항

기본 사항 알아보기

다음 코드 예제에서는 다음과 같은 작업을 수행하는 방법을 보여줍니다.

- 사용자 지정 Aurora DB 클러스터 파라미터 그룹을 만들고 파라미터 값을 설정합니다.
- 파라미터 그룹을 사용하는 DB 클러스터를 생성합니다.
- 데이터베이스가 포함된 DB 인스턴스를 생성합니다.
- DB 클러스터의 스냅샷을 만든 다음, 리소스를 정리합니다.

SDK for .NET (v4)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

명령 프롬프트에서 대화형 시나리오를 실행합니다.

```
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;
    }
```

Aurora 작업을 관리하기 위해 시나리오가 직접적으로 호출하는 래퍼 메서드.

```
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;
    }
}
```

- API 세부 정보는 AWS SDK for .NET API 참조의 다음 항목을 참조하세요.
  - CreateDBCluster
  - CreateDBClusterParameterGroup
  - CreateDBClusterSnapshot
  - CreateDBInstance
  - DeleteDBCluster
  - DeleteDBClusterParameterGroup
  - DeleteDBInstance
  - DescribeDBClusterParameterGroups
  - DescribeDBClusterParameters
  - DescribeDBClusterSnapshots
  - DescribeDBClusters
  - DescribeDBEngineVersions
  - DescribeDBInstances
  - DescribeOrderableDBInstanceOptions
  - ModifyDBClusterParameterGroup

작업

`CreateDBCluster`

다음 코드 예시는 `CreateDBCluster`의 사용 방법을 보여 줍니다.

SDK for .NET (v4)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

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

- API 세부 정보는 AWS SDK for .NET API 참조의 CreateDBCluster를 참조하세요.

`CreateDBClusterParameterGroup`

다음 코드 예시는 `CreateDBClusterParameterGroup`의 사용 방법을 보여 줍니다.

SDK for .NET (v4)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

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

- API 세부 정보는 AWS SDK for .NET SDK for Rust API 참조의 CreateDBClusterParameterGroup를 참조하세요.

`CreateDBClusterSnapshot`

다음 코드 예시는 `CreateDBClusterSnapshot`의 사용 방법을 보여 줍니다.

SDK for .NET (v4)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

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

- API 세부 정보는 AWS SDK for .NET API 참조의 CreateDBClusterSnapshot을 참조하세요.

`CreateDBInstance`

다음 코드 예시는 `CreateDBInstance`의 사용 방법을 보여 줍니다.

SDK for .NET (v4)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

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

- API 세부 정보는 AWS SDK for .NET API 참조의 CreateDBInstance를 참조하세요.

DeleteDBCluster

다음 코드 예시는 DeleteDBCluster의 사용 방법을 보여 줍니다.

SDK for .NET (v4)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

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

- API 세부 정보는 AWS SDK for .NET API 참조의 DeleteDBCluster를 참조하세요.

DeleteDBClusterParameterGroup

다음 코드 예시는 DeleteDBClusterParameterGroup의 사용 방법을 보여 줍니다.

SDK for .NET (v4)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

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

- API 세부 정보는 AWS SDK for .NET API 참조의 DeleteDBClusterParameterGroup을 참조하세요.

`DeleteDBInstance`

다음 코드 예시는 `DeleteDBInstance`의 사용 방법을 보여 줍니다.

SDK for .NET (v4)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

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

- API 세부 정보는 AWS SDK for .NET API 참조의 DeleteDBInstance를 참조하세요.

`DescribeDBClusterParameterGroups`

다음 코드 예시는 `DescribeDBClusterParameterGroups`의 사용 방법을 보여 줍니다.

SDK for .NET (v4)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

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

- API 세부 정보는 AWS SDK for .NET API 참조의 DescribeDBClusterParameterGroups를 참조하세요.

`DescribeDBClusterParameters`

다음 코드 예시는 `DescribeDBClusterParameters`의 사용 방법을 보여 줍니다.

SDK for .NET (v4)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

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

- API 세부 정보는 AWS SDK for .NET API 참조의 DescribeDBClusterParameters를 참조하세요.

`DescribeDBClusterSnapshots`

다음 코드 예시는 `DescribeDBClusterSnapshots`의 사용 방법을 보여 줍니다.

SDK for .NET (v4)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

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

- API 세부 정보는 AWS SDK for .NET API 참조의 DescribeDBClusterSnapshots를 참조하세요.

`DescribeDBClusters`

다음 코드 예시는 `DescribeDBClusters`의 사용 방법을 보여 줍니다.

SDK for .NET (v4)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

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

- API 세부 정보는 AWS SDK for .NET API 참조의 DescribeDBClusters를 참조하세요.

`DescribeDBEngineVersions`

다음 코드 예시는 `DescribeDBEngineVersions`의 사용 방법을 보여 줍니다.

SDK for .NET (v4)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

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

- API 세부 정보는 AWS SDK for .NET API 참조의 DescribeDBEngineVersions을 참조하세요.

`DescribeDBInstances`

다음 코드 예시는 `DescribeDBInstances`의 사용 방법을 보여 줍니다.

SDK for .NET (v4)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

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

- API 세부 정보는 AWS SDK for .NET API 참조의 DescribeDBInstances를 참조하세요.

`DescribeOrderableDBInstanceOptions`

다음 코드 예시는 `DescribeOrderableDBInstanceOptions`의 사용 방법을 보여 줍니다.

SDK for .NET (v4)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

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

- API 세부 정보는 AWS SDK for .NET API 참조의 DescribeOrderableDBInstanceOptions를 참조하세요.

`ModifyDBClusterParameterGroup`

다음 코드 예시는 `ModifyDBClusterParameterGroup`의 사용 방법을 보여 줍니다.

SDK for .NET (v4)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

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

- API 세부 정보는 AWS SDK for .NET API 참조의 ModifyDBClusterParameterGroup을 참조하세요.

SDK for .NET (v4)를 사용한 Auto Scaling 예제

다음 코드 예제에서는 Auto Scaling과 함께 AWS SDK for .NET (v4)를 사용하여 작업을 수행하고 일반적인 시나리오를 구현하는 방법을 보여줍니다.

기본 사항은 서비스 내에서 필수 작업을 수행하는 방법을 보여주는 코드 예제입니다.

작업은 대규모 프로그램에서 발췌한 코드이며 컨텍스트에 맞춰 실행해야 합니다. 작업은 개별 서비스 함수를 직접적으로 호출하는 방법을 보여주며 관련 시나리오의 컨텍스트에 맞는 작업을 볼 수 있습니다.

각 예시에는 전체 소스 코드에 대한 링크가 포함되어 있으며, 여기에서 컨텍스트에 맞춰 코드를 설정하고 실행하는 방법에 대한 지침을 찾을 수 있습니다.

주제

- 시작하기
- 기본 사항
- 작업

시작하기

Auto Scaling 시작

다음 코드 예제에서는 Auto Scaling 사용을 시작하는 방법을 보여줍니다.

SDK for .NET (v4)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
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
        });

    }
}
```

- API 세부 정보는 AWS SDK for .NET API 참조의 DescribeAutoScalingGroups을 참조하세요.

기본 사항

기본 사항 알아보기

다음 코드 예제에서는 다음과 같은 작업을 수행하는 방법을 보여줍니다.

- 시작 템플릿과 가용 영역이 있는 Amazon EC2 Auto Scaling 그룹을 생성하고 실행 중인 인스턴스에 대한 정보를 가져옵니다.
- Amazon CloudWatch 지표 수집 활성화
- 그룹의 원하는 용량을 업데이트하고 인스턴스가 시작될 때까지 기다립니다.
- 그룹에서 인스턴스를 종료합니다.
- 사용자 요청 및 용량 변경에 따라 발생하는 조정 활동을 나열합니다.
- CloudWatch 지표에 대한 통계를 가져온 다음 리소스를 정리합니다.

SDK for .NET (v4)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
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();
    }
}
```

시작 템플릿과 지표를 관리하기 위해 시나리오가 호출하는 함수를 정의합니다. 이러한 함수는 Auto Scaling, Amazon EC2 및 CloudWatch 작업을 래핑합니다.

```
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;
    }

}
```

- API 세부 정보는 AWS SDK for .NET API 참조의 다음 주제를 참조하십시오.
  - CreateAutoScalingGroup
  - DeleteAutoScalingGroup
  - DescribeAutoScalingGroups
  - DescribeAutoScalingInstances
  - DescribeScalingActivities
  - DisableMetricsCollection
  - EnableMetricsCollection
  - SetDesiredCapacity
  - TerminateInstanceInAutoScalingGroup
  - UpdateAutoScalingGroup

작업

`CreateAutoScalingGroup`

다음 코드 예시는 `CreateAutoScalingGroup`의 사용 방법을 보여 줍니다.

SDK for .NET (v4)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

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

- API에 대한 자세한 설명은 AWS SDK for .NET API 참조 문서의 CreateAutoScalingGroup을 참조하세요.

`DescribeAutoScalingGroups`

다음 코드 예시는 `DescribeAutoScalingGroups`의 사용 방법을 보여 줍니다.

SDK for .NET (v4)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

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

- API에 대한 세부 정보는 AWS SDK for .NET API 참조의 DescribeAutoScalingGroups를 참조하세요.

`DescribeAutoScalingInstances`

다음 코드 예시는 `DescribeAutoScalingInstances`의 사용 방법을 보여 줍니다.

SDK for .NET (v4)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

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

- API에 대한 세부 정보는 AWS SDK for .NET API 참조의 DescribeAutoScalingInstances를 참조하세요.

`DescribeScalingActivities`

다음 코드 예시는 `DescribeScalingActivities`의 사용 방법을 보여 줍니다.

SDK for .NET (v4)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

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

- API에 대한 세부 정보는 AWS SDK for .NET API 참조의 DescribeScalingActivities를 참조하세요.

`DisableMetricsCollection`

다음 코드 예시는 `DisableMetricsCollection`의 사용 방법을 보여 줍니다.

SDK for .NET (v4)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

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

- API에 대한 세부 정보는 AWS SDK for .NET API 참조의 DisableMetricsCollection을 참조하세요.

`EnableMetricsCollection`

다음 코드 예시는 `EnableMetricsCollection`의 사용 방법을 보여 줍니다.

SDK for .NET (v4)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

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

- API에 대한 세부 정보는 AWS SDK for .NET API 참조의 EnableMetricsCollection을 참조하세요.

`SetDesiredCapacity`

다음 코드 예시는 `SetDesiredCapacity`의 사용 방법을 보여 줍니다.

SDK for .NET (v4)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

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

- API에 대한 세부 정보는 AWS SDK for .NET API 참조의 SetDesiredCapacity를 참조하세요.

`TerminateInstanceInAutoScalingGroup`

다음 코드 예시는 `TerminateInstanceInAutoScalingGroup`의 사용 방법을 보여 줍니다.

SDK for .NET (v4)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

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

- API에 대한 세부 정보는 AWS SDK for .NET API 참조의 TerminateInstanceInAutoScalingGroup을 참조하세요.

`UpdateAutoScalingGroup`

다음 코드 예시는 `UpdateAutoScalingGroup`의 사용 방법을 보여 줍니다.

SDK for .NET (v4)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

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

- API에 대한 자세한 설명은 AWS SDK for .NET API 참조 문서의 UpdateAutoScalingGroup을 참조하세요.

SDK for .NET (v4)를 사용한 Amazon Bedrock 예제

다음 코드 예제에서는 Amazon Bedrock에서 AWS SDK for .NET (v4)를 사용하여 작업을 수행하고 일반적인 시나리오를 구현하는 방법을 보여줍니다.

작업은 대규모 프로그램에서 발췌한 코드이며 컨텍스트에 맞춰 실행해야 합니다. 작업은 개별 서비스 함수를 직접적으로 호출하는 방법을 보여주며 관련 시나리오의 컨텍스트에 맞는 작업을 볼 수 있습니다.

각 예시에는 전체 소스 코드에 대한 링크가 포함되어 있으며, 여기에서 컨텍스트에 맞춰 코드를 설정하고 실행하는 방법에 대한 지침을 찾을 수 있습니다.

주제

- 시작하기
- 작업

시작하기

Amazon Bedrock 시작

다음 코드 예제에서는 Amazon Bedrock 사용을 시작하는 방법을 보여줍니다.

SDK for .NET (v4)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

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

- API 세부 정보는 AWS SDK for .NET API 참조의 ListFoundationModels를 참조하세요.

작업

`ListFoundationModels`

다음 코드 예시는 `ListFoundationModels`의 사용 방법을 보여 줍니다.

SDK for .NET (v4)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

사용 가능한 Bedrock 기본 모델을 나열합니다.

```
    /// <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);
        }
    }
```

- API 세부 정보는 AWS SDK for .NET API 참조의 ListFoundationModels를 참조하세요.

SDK for .NET (v4)를 사용한 Amazon Bedrock 런타임 예제

다음 코드 예제에서는 Amazon Bedrock 런타임과 함께 AWS SDK for .NET (v4)를 사용하여 작업을 수행하고 일반적인 시나리오를 구현하는 방법을 보여줍니다.

각 예시에는 전체 소스 코드에 대한 링크가 포함되어 있으며, 여기에서 컨텍스트에 맞춰 코드를 설정하고 실행하는 방법에 대한 지침을 찾을 수 있습니다.

주제

- Anthropic Claude
- Cohere Command
- Meta Llama
- Mistral AI

Anthropic Claude

Converse

다음 코드 예제에서는 Bedrock의 Converse API를 사용하여 Anthropic Claude에 텍스트 메시지를 보내는 방법을 보여줍니다.

SDK for .NET (v4)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

Bedrock의 Converse API를 사용하여 Anthropic Claude에 텍스트 메시지를 보냅니다.

```
// 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;
}
```

- API 세부 정보는 AWS SDK for .NET API 참조의 Converse를 참조하세요.

ConverseStream

다음 코드 예제에서는 Bedrock의 Converse API를 사용하여 Anthropic Claude에 텍스트 메시지를 보내고 응답 스트림을 실시간으로 처리하는 방법을 보여줍니다.

SDK for .NET (v4)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

Bedrock의 Converse API를 사용하여 Anthropic Claude에 텍스트 메시지를 보내고 응답 스트림을 실시간으로 처리합니다.

```
// 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;
}
```

- API 세부 정보는 AWS SDK for .NET API 참조의 ConverseStream을 참조하세요.

InvokeModel

다음 코드 예제에서는 Invoke Model API를 사용하여 Anthropic Claude에 텍스트 메시지를 보내는 방법을 보여줍니다.

SDK for .NET (v4)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

Invoke Model API를 사용하여 텍스트 메시지를 보냅니다.

```
// 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;
}
```

- API 세부 정보는 AWS SDK for .NET API 참조의 InvokeModel을 참조하세요.

Cohere Command

Converse

다음 코드 예제에서는 Bedrock의 Converse API를 사용하여 Cohere Command에 텍스트 메시지를 보내는 방법을 보여줍니다.

SDK for .NET (v4)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

Bedrock의 Converse API를 사용하여 Cohere Command로 텍스트 메시지를 보냅니다.

```
// 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;
}
```

- API 세부 정보는 AWS SDK for .NET API 참조의 Converse를 참조하세요.

ConverseStream

다음 코드 예제에서는 Bedrock의 Converse API를 사용하여 Cohere Command에 텍스트 메시지를 보내고 응답 스트림을 실시간으로 처리하는 방법을 보여줍니다.

SDK for .NET (v4)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

Bedrock의 Converse API를 사용하여 Cohere Command에 텍스트 메시지를 보내고 응답 스트림을 실시간으로 처리합니다.

```
// 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;
}
```

- API 세부 정보는 AWS SDK for .NET API 참조의 ConverseStream을 참조하세요.

Meta Llama

Converse

다음 코드 예제에서는 Bedrock의 Converse API를 사용하여 Meta Llama에 텍스트 메시지를 보내는 방법을 보여줍니다.

SDK for .NET (v4)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

Bedrock의 Converse API를 사용하여 Meta Llama에 텍스트 메시지를 보냅니다.

```
// 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;
}
```

- API 세부 정보는 AWS SDK for .NET API 참조의 Converse를 참조하세요.

ConverseStream

다음 코드 예제에서는 Bedrock의 Converse API를 사용하여 Meta Llama에 텍스트 메시지를 보내고 응답 스트림을 실시간으로 처리하는 방법을 보여줍니다.

SDK for .NET (v4)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

Bedrock의 Converse API를 사용하여 Meta Llama에 텍스트 메시지를 보내고 응답 스트림을 실시간으로 처리합니다.

```
// 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;
}
```

- API 세부 정보는 AWS SDK for .NET API 참조의 ConverseStream을 참조하세요.

Mistral AI

Converse

다음 코드 예제에서는 Bedrock의 Converse API를 사용하여 Mistral에 텍스트 메시지를 보내는 방법을 보여줍니다.

SDK for .NET (v4)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

Bedrock의 Converse API를 사용하여 Mistral에 텍스트 메시지를 보냅니다.

```
// 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;
}
```

- API 세부 정보는 AWS SDK for .NET API 참조의 Converse를 참조하세요.

ConverseStream

다음 코드 예제에서는 Bedrock의 Converse API를 사용하여 Mistral에 텍스트 메시지를 보내고 응답 스트림을 실시간으로 처리하는 방법을 보여줍니다.

SDK for .NET (v4)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

Bedrock의 Converse API를 사용하여 Mistral에 텍스트 메시지를 보내고 응답 스트림을 실시간으로 처리합니다.

```
// 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;
}
```

- API 세부 정보는 AWS SDK for .NET API 참조의 ConverseStream을 참조하세요.

CloudFormation SDK for .NET (v4)를 사용한 예제

다음 코드 예제에서는 AWS SDK for .NET (v4)를와 함께 사용하여 작업을 수행하고 일반적인 시나리오를 구현하는 방법을 보여줍니다 CloudFormation.

각 예시에는 전체 소스 코드에 대한 링크가 포함되어 있으며, 여기에서 컨텍스트에 맞춰 코드를 설정하고 실행하는 방법에 대한 지침을 찾을 수 있습니다.

주제

- 시작하기

시작하기

안녕하세요 CloudFormation

다음 코드 예제에서는 CloudFormation를 사용하여 시작하는 방법을 보여 줍니다.

SDK for .NET (v4)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
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;
```

```
        }
    }
```

- API 세부 정보는 AWS SDK for .NET API 참조의 DescribeStackResources를 참조하세요.

SDK for .NET (v4)를 사용한 CloudWatch 예제

다음 코드 예제에서는 CloudWatch에서 AWS SDK for .NET (v4)를 사용하여 작업을 수행하고 일반적인 시나리오를 구현하는 방법을 보여줍니다.

기본 사항은 서비스 내에서 필수 작업을 수행하는 방법을 보여주는 코드 예제입니다.

작업은 대규모 프로그램에서 발췌한 코드이며 컨텍스트에 맞춰 실행해야 합니다. 작업은 개별 서비스 함수를 직접적으로 호출하는 방법을 보여주며 관련 시나리오의 컨텍스트에 맞는 작업을 볼 수 있습니다.

각 예시에는 전체 소스 코드에 대한 링크가 포함되어 있으며, 여기에서 컨텍스트에 맞춰 코드를 설정하고 실행하는 방법에 대한 지침을 찾을 수 있습니다.

주제

- 시작하기
- 기본 사항
- 작업

시작하기

Hello CloudWatch

다음 코드 예제에서는 CloudWatch 사용을 시작하는 방법을 보여줍니다.

SDK for .NET (v4)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
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();
```

```
            }
        }
    }
}
```

- API 세부 정보는 AWS SDK for .NET API 참조의 ListMetrics를 참조하세요.

기본 사항

기본 사항 알아보기

다음 코드 예제에서는 다음과 같은 작업을 수행하는 방법을 보여줍니다.

- CloudWatch 네임스페이스 및 지표를 나열합니다.
- 지표 및 예상 청구에 대한 통계를 가져옵니다.
- 대시보드를 생성하고 업데이트합니다.
- 데이터를 생성하여 지표에 추가합니다.
- 경보를 생성하고 트리거한 다음 경보 기록을 봅니다.
- 이상 탐지기를 추가합니다.
- 지표 이미지를 가져온 다음 리소스를 정리합니다.

SDK for .NET (v4)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

명령 프롬프트에서 대화형 시나리오를 실행합니다.

```
public class CloudWatchScenario
{
    /*
    Before running this .NET code example, set up your development environment,
 including your credentials.
```

```
    To enable billing metrics and statistics for this example, make sure billing
 alerts are enabled for your account:
    https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/
monitor_estimated_charges_with_cloudwatch.html#turning_on_billing_metrics

    This .NET example performs the following tasks:
        1. List and select a CloudWatch namespace.
        2. List and select a CloudWatch metric.
        3. Get statistics for a CloudWatch metric.
        4. Get estimated billing statistics for the last week.
        5. Create a new CloudWatch dashboard with two metrics.
        6. List current CloudWatch dashboards.
        7. Create a CloudWatch custom metric and add metric data.
        8. Add the custom metric to the dashboard.
        9. Create a CloudWatch alarm for the custom metric.
       10. Describe current CloudWatch alarms.
       11. Get recent data for the custom metric.
       12. Add data to the custom metric to trigger the alarm.
       13. Wait for an alarm state.
       14. Get history for the CloudWatch alarm.
       15. Add an anomaly detector.
       16. Describe current anomaly detectors.
       17. Get and display a metric image.
       18. Clean up resources.
    */

    private static ILogger logger = null!;
    private static CloudWatchWrapper _cloudWatchWrapper = null!;
    private static IConfiguration _configuration = null!;
    private static readonly List<string> _statTypes = new List<string>
 { "SampleCount", "Average", "Sum", "Minimum", "Maximum" };
    private static SingleMetricAnomalyDetector? anomalyDetector = null!;

    static async Task Main(string[] args)
    {
        // Set up dependency injection for the Amazon service.
        using var host = Host.CreateDefaultBuilder(args)
            .ConfigureLogging(logging =>
                logging.AddFilter("System", LogLevel.Debug)
                    .AddFilter<DebugLoggerProvider>("Microsoft",
 LogLevel.Information)
                    .AddFilter<ConsoleLoggerProvider>("Microsoft", LogLevel.Trace))
            .ConfigureServices((_, services) =>
            services.AddAWSService<IAmazonCloudWatch>()
```

```
            .AddTransient<CloudWatchWrapper>()
        )
        .Build();

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

        logger = LoggerFactory.Create(builder => { builder.AddConsole(); })
            .CreateLogger<CloudWatchScenario>();

        _cloudWatchWrapper = host.Services.GetRequiredService<CloudWatchWrapper>();

        Console.WriteLine(new string('-', 80));
        Console.WriteLine("Welcome to the Amazon CloudWatch example scenario.");
        Console.WriteLine(new string('-', 80));

        try
        {
            var selectedNamespace = await SelectNamespace();
            var selectedMetric = await SelectMetric(selectedNamespace);
            await GetAndDisplayMetricStatistics(selectedNamespace, selectedMetric);
            await GetAndDisplayEstimatedBilling();
            await CreateDashboardWithMetrics();
            await ListDashboards();
            await CreateNewCustomMetric();
            await AddMetricToDashboard();
            await CreateMetricAlarm();
            await DescribeAlarms();
            await GetCustomMetricData();
            await AddMetricDataForAlarm();
            await CheckForMetricAlarm();
            await GetAlarmHistory();
            anomalyDetector = await AddAnomalyDetector();
            await DescribeAnomalyDetectors();
            await GetAndOpenMetricImage();
            await CleanupResources();
        }
        catch (Exception ex)
        {
            logger.LogError(ex, "There was a problem executing the scenario.");
```

```
            await CleanupResources();
        }

    }

    /// <summary>
    /// Select a namespace.
    /// </summary>
    /// <returns>The selected namespace.</returns>
    private static async Task<string> SelectNamespace()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"1. Select a CloudWatch Namespace from a list of
Namespaces.");
        var metrics = await _cloudWatchWrapper.ListMetrics();
        // Get a distinct list of namespaces.
        var namespaces = metrics.Select(m => m.Namespace).Distinct().ToList();
        for (int i = 0; i < namespaces.Count; i++)
        {
            Console.WriteLine($"\t{i + 1}. {namespaces[i]}");
        }

        var namespaceChoiceNumber = 0;
        while (namespaceChoiceNumber < 1 || namespaceChoiceNumber >
namespaces.Count)
        {
            Console.WriteLine(
                "Select a namespace by entering a number from the preceding list:");
            var choice = Console.ReadLine();
            Int32.TryParse(choice, out namespaceChoiceNumber);
        }

        var selectedNamespace = namespaces[namespaceChoiceNumber - 1];

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

        return selectedNamespace;
    }

    /// <summary>
    /// Select a metric from a namespace.
    /// </summary>
    /// <param name="metricNamespace">The namespace for metrics.</param>
    /// <returns>The metric name.</returns>
```

```
    private static async Task<Metric> SelectMetric(string metricNamespace)
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"2. Select a CloudWatch metric from a namespace.");

        var namespaceMetrics = await
_cloudWatchWrapper.ListMetrics(metricNamespace);

        for (int i = 0; i < namespaceMetrics.Count && i < 15; i++)
        {
            var dimensionsWithValues = namespaceMetrics[i].Dimensions
                .Where(d => !string.Equals("None", d.Value));
            Console.WriteLine($"\t{i + 1}. {namespaceMetrics[i].MetricName} " +
                              $"{string.Join(", :", dimensionsWithValues.Select(d =>
d.Value))}");
        }

        var metricChoiceNumber = 0;
        while (metricChoiceNumber < 1 || metricChoiceNumber >
namespaceMetrics.Count)
        {
            Console.WriteLine(
                "Select a metric by entering a number from the preceding list:");
            var choice = Console.ReadLine();
            Int32.TryParse(choice, out metricChoiceNumber);
        }

        var selectedMetric = namespaceMetrics[metricChoiceNumber - 1];

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

        return selectedMetric;
    }

    /// <summary>
    /// Get and display metric statistics for a specific metric.
    /// </summary>
    /// <param name="metricNamespace">The namespace for metrics.</param>
    /// <param name="metric">The CloudWatch metric.</param>
    /// <returns>Async task.</returns>
    private static async Task GetAndDisplayMetricStatistics(string metricNamespace,
Metric metric)
    {
        Console.WriteLine(new string('-', 80));
```

```
        Console.WriteLine($"3. Get CloudWatch metric statistics for the last day.");

        for (int i = 0; i < _statTypes.Count; i++)
        {
            Console.WriteLine($"\t{i + 1}. {_statTypes[i]}");
        }

        var statisticChoiceNumber = 0;
        while (statisticChoiceNumber < 1 || statisticChoiceNumber >
_statTypes.Count)
        {
            Console.WriteLine(
                "Select a metric statistic by entering a number from the preceding
list:");
            var choice = Console.ReadLine();
            Int32.TryParse(choice, out statisticChoiceNumber);
        }

        var selectedStatistic = _statTypes[statisticChoiceNumber - 1];
        var statisticsList = new List<string> { selectedStatistic };

        var metricStatistics = await
_cloudWatchWrapper.GetMetricStatistics(metricNamespace, metric.MetricName,
statisticsList, metric.Dimensions, 1, 60);

        if (!metricStatistics.Any())
        {
            Console.WriteLine($"No {selectedStatistic} statistics found for {metric}
in namespace {metricNamespace}.");
        }

        metricStatistics = metricStatistics.OrderBy(s => s.Timestamp).ToList();
        for (int i = 0; i < metricStatistics.Count && i < 10; i++)
        {
            var metricStat = metricStatistics[i];
            var statValue =
metricStat.GetType().GetProperty(selectedStatistic)!.GetValue(metricStat, null);
            Console.WriteLine($"\t{i + 1}. Timestamp
{metricStatistics[i].Timestamp:G} {selectedStatistic}: {statValue}");
        }

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

```
    /// <summary>
    /// Get and display estimated billing statistics.
    /// </summary>
    /// <param name="metricNamespace">The namespace for metrics.</param>
    /// <param name="metric">The CloudWatch metric.</param>
    /// <returns>Async task.</returns>
    private static async Task GetAndDisplayEstimatedBilling()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"4. Get CloudWatch estimated billing for the last
week.");

        var billingStatistics = await SetupBillingStatistics();

        for (int i = 0; i < billingStatistics.Count; i++)
        {
            Console.WriteLine($"\t{i + 1}. Timestamp
{billingStatistics[i].Timestamp:G} : {billingStatistics[i].Maximum}");
        }

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

    /// <summary>
    /// Get billing statistics using a call to a wrapper class.
    /// </summary>
    /// <returns>A collection of billing statistics.</returns>
    private static async Task<List<Datapoint>> SetupBillingStatistics()
    {
        // Make a request for EstimatedCharges with a period of one day for the past
seven days.
        var billingStatistics = await _cloudWatchWrapper.GetMetricStatistics(
            "AWS/Billing",
            "EstimatedCharges",
            new List<string>() { "Maximum" },
            new List<Dimension>() { new Dimension { Name = "Currency", Value =
"USD" } },
            7,
            86400);

        billingStatistics = billingStatistics.OrderBy(n => n.Timestamp).ToList();

        return billingStatistics;
    }
```

```
    /// <summary>
    /// Create a dashboard with metrics.
    /// </summary>
    /// <param name="metricNamespace">The namespace for metrics.</param>
    /// <param name="metric">The CloudWatch metric.</param>
    /// <returns>Async task.</returns>
    private static async Task CreateDashboardWithMetrics()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"5. Create a new CloudWatch dashboard with metrics.");
        var dashboardName = _configuration["dashboardName"];
        var newDashboard = new DashboardModel();
        _configuration.GetSection("dashboardExampleBody").Bind(newDashboard);
        var newDashboardString = JsonSerializer.Serialize(
            newDashboard,
            new JsonSerializerOptions
            {
                DefaultIgnoreCondition = JsonIgnoreCondition.WhenWritingNull
            });
        var validationMessages =
            await _cloudWatchWrapper.PutDashboard(dashboardName,
newDashboardString);

        Console.WriteLine(validationMessages.Any() ? $"\tValidation messages:" :
null);
        for (int i = 0; i < validationMessages.Count; i++)
        {
            Console.WriteLine($"\t{i + 1}. {validationMessages[i].Message}");
        }
        Console.WriteLine($"\tDashboard {dashboardName} was created.");
        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// List dashboards.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task ListDashboards()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"6. List the CloudWatch dashboards in the current
account.");
```

```
        var dashboards = await _cloudWatchWrapper.ListDashboards();

        for (int i = 0; i < dashboards.Count; i++)
        {
            Console.WriteLine($"\t{i + 1}. {dashboards[i].DashboardName}");
        }

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

    /// <summary>
    /// Create and add data for a new custom metric.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task CreateNewCustomMetric()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"7. Create and add data for a new custom metric.");

        var customMetricNamespace = _configuration["customMetricNamespace"];
        var customMetricName = _configuration["customMetricName"];

        var customData = await PutRandomMetricData(customMetricName,
customMetricNamespace);

        var valuesString = string.Join(',', customData.Select(d => d.Value));
        Console.WriteLine($"\tAdded metric values for for metric {customMetricName}:
\n\t{valuesString}");

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


    /// <summary>
    /// Add some metric data using a call to a wrapper class.
    /// </summary>
    /// <param name="customMetricName">The metric name.</param>
    /// <param name="customMetricNamespace">The metric namespace.</param>
    /// <returns></returns>
    private static async Task<List<MetricDatum>> PutRandomMetricData(string
customMetricName,
        string customMetricNamespace)
    {
        List<MetricDatum> customData = new List<MetricDatum>();
```

```
        Random rnd = new Random();

        // Add 10 random values up to 100, starting with a timestamp 15 minutes in
the past.
        var utcNowMinus15 = DateTime.UtcNow.AddMinutes(-15);
        for (int i = 0; i < 10; i++)
        {
            var metricValue = rnd.Next(0, 100);
            customData.Add(
                new MetricDatum
                {
                    MetricName = customMetricName,
                    Value = metricValue,
                    TimestampUtc = utcNowMinus15.AddMinutes(i)
                }
            );
        }

        await _cloudWatchWrapper.PutMetricData(customMetricNamespace, customData);
        return customData;
    }

    /// <summary>
    /// Add the custom metric to the dashboard.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task AddMetricToDashboard()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"8. Add the new custom metric to the dashboard.");

        var dashboardName = _configuration["dashboardName"];

        var customMetricNamespace = _configuration["customMetricNamespace"];
        var customMetricName = _configuration["customMetricName"];

        var validationMessages = await SetupDashboard(customMetricNamespace,
customMetricName, dashboardName);

        Console.WriteLine(validationMessages.Any() ? $"\tValidation messages:" :
null);
        for (int i = 0; i < validationMessages.Count; i++)
        {
            Console.WriteLine($"\t{i + 1}. {validationMessages[i].Message}");
```

```
        }
        Console.WriteLine($"\tDashboard {dashboardName} updated with metric
{customMetricName}.");
        Console.WriteLine(new string('-', 80));
    }


    /// <summary>
    /// Set up a dashboard using a call to the wrapper class.
    /// </summary>
    /// <param name="customMetricNamespace">The metric namespace.</param>
    /// <param name="customMetricName">The metric name.</param>
    /// <param name="dashboardName">The name of the dashboard.</param>
    /// <returns>A list of validation messages.</returns>
    private static async Task<List<DashboardValidationMessage>> SetupDashboard(
        string customMetricNamespace, string customMetricName, string dashboardName)
    {
        // Get the dashboard model from configuration.
        var newDashboard = new DashboardModel();
        _configuration.GetSection("dashboardExampleBody").Bind(newDashboard);

        // Add a new metric to the dashboard.
        newDashboard.Widgets.Add(new Widget
        {
            Height = 8,
            Width = 8,
            Y = 8,
            X = 0,
            Type = "metric",
            Properties = new Properties
            {
                Metrics = new List<List<object>>
                    { new() { customMetricNamespace, customMetricName } },
                View = "timeSeries",
                Region = "us-east-1",
                Stat = "Sum",
                Period = 86400,
                YAxis = new YAxis { Left = new Left { Min = 0, Max = 100 } },
                Title = "Custom Metric Widget",
                LiveData = true,
                Sparkline = true,
                Trend = true,
                Stacked = false,
                SetPeriodToTimeRange = false
```

```
            }
        });

        var newDashboardString = JsonSerializer.Serialize(newDashboard,
            new JsonSerializerOptions
            { DefaultIgnoreCondition = JsonIgnoreCondition.WhenWritingNull });
        var validationMessages =
            await _cloudWatchWrapper.PutDashboard(dashboardName,
newDashboardString);

        return validationMessages;
    }

    /// <summary>
    /// Create a CloudWatch alarm for the new metric.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task CreateMetricAlarm()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"9. Create a CloudWatch alarm for the new metric.");

        var customMetricNamespace = _configuration["customMetricNamespace"];
        var customMetricName = _configuration["customMetricName"];

        var alarmName = _configuration["exampleAlarmName"];
        var accountId = _configuration["accountId"];
        var region = _configuration["region"];
        var emailTopic = _configuration["emailTopic"];
        var alarmActions = new List<string>();

        if (GetYesNoResponse(
                $"\tAdd an email action for topic {emailTopic} to alarm {alarmName}?
(y/n)"))
        {
            _cloudWatchWrapper.AddEmailAlarmAction(accountId, region, emailTopic,
alarmActions);
        }

        await _cloudWatchWrapper.PutMetricEmailAlarm(
            "Example metric alarm",
            alarmName,
            ComparisonOperator.GreaterThanOrEqualToThreshold,
            customMetricName,
```

```
            customMetricNamespace,
            100,
            alarmActions);

        Console.WriteLine($"\tAlarm {alarmName} added for metric
{customMetricName}.");
        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Describe Alarms.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task DescribeAlarms()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"10. Describe CloudWatch alarms in the current
account.");

        var alarms = await _cloudWatchWrapper.DescribeAlarms();
        alarms = alarms.OrderByDescending(a => a.StateUpdatedTimestamp).ToList();

        for (int i = 0; i < alarms.Count && i < 10; i++)
        {
            var alarm = alarms[i];
            Console.WriteLine($"\t{i + 1}. {alarm.AlarmName}");
            Console.WriteLine($"\tState: {alarm.StateValue} for {alarm.MetricName}
{alarm.ComparisonOperator} {alarm.Threshold}");
        }

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

    /// <summary>
    /// Get the recent data for the metric.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task GetCustomMetricData()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"11. Get current data for new custom metric.");

        var customMetricNamespace = _configuration["customMetricNamespace"];
        var customMetricName = _configuration["customMetricName"];
```

```
        var accountId = _configuration["accountId"];

        var query = new List<MetricDataQuery>
        {
            new MetricDataQuery
            {
                AccountId = accountId,
                Id = "m1",
                Label = "Custom Metric Data",
                MetricStat = new MetricStat
                {
                    Metric = new Metric
                    {
                        MetricName = customMetricName,
                        Namespace = customMetricNamespace,
                    },
                    Period = 1,
                    Stat = "Maximum"
                }
            }
        };

        var metricData = await _cloudWatchWrapper.GetMetricData(
            20,
            true,
            DateTime.UtcNow.AddMinutes(1),
            20,
            query);

        for (int i = 0; i < metricData.Count; i++)
        {
            if (metricData[i].Values != null)
            {
                for (int j = 0; j < metricData[i].Values.Count; j++)
                {
                    Console.WriteLine(
                        $"\tTimestamp {metricData[i].Timestamps[j]:G} Value:
{metricData[i].Values[j]}");
                }
            }
        }

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

```
    /// <summary>
    /// Add metric data to trigger an alarm.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task AddMetricDataForAlarm()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"12. Add metric data to the custom metric to trigger an
alarm.");

        var customMetricNamespace = _configuration["customMetricNamespace"];
        var customMetricName = _configuration["customMetricName"];
        var nowUtc = DateTime.UtcNow;
        List<MetricDatum> customData = new List<MetricDatum>
        {
            new MetricDatum
            {
                MetricName = customMetricName,
                Value = 101,
                TimestampUtc = nowUtc.AddMinutes(-2)
            },
            new MetricDatum
            {
                MetricName = customMetricName,
                Value = 101,
                TimestampUtc = nowUtc.AddMinutes(-1)
            },
            new MetricDatum
            {
                MetricName = customMetricName,
                Value = 101,
                TimestampUtc = nowUtc
            }
        };
        var valuesString = string.Join(',', customData.Select(d => d.Value));
        Console.WriteLine($"\tAdded metric values for for metric {customMetricName}:
\n\t{valuesString}");
        await _cloudWatchWrapper.PutMetricData(customMetricNamespace, customData);

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

    /// <summary>
```

```
    /// Check for a metric alarm using the DescribeAlarmsForMetric action.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task CheckForMetricAlarm()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"13. Checking for an alarm state.");

        var customMetricNamespace = _configuration["customMetricNamespace"];
        var customMetricName = _configuration["customMetricName"];
        var hasAlarm = false;
        var retries = 10;
        while (!hasAlarm && retries > 0)
        {
            var alarms = await
_cloudWatchWrapper.DescribeAlarmsForMetric(customMetricNamespace,
customMetricName);
            hasAlarm = alarms.Any(a => a.StateValue == StateValue.ALARM);
            retries--;
            Thread.Sleep(20000);
        }

        Console.WriteLine(hasAlarm
            ? $"\tAlarm state found for {customMetricName}."
            : $"\tNo Alarm state found for {customMetricName} after 10 retries.");

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

    /// <summary>
    /// Get history for an alarm.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task GetAlarmHistory()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"14. Get alarm history.");

        var exampleAlarmName = _configuration["exampleAlarmName"];

        var alarmHistory = await
_cloudWatchWrapper.DescribeAlarmHistory(exampleAlarmName, 2);

        for (int i = 0; i < alarmHistory.Count; i++)
```

```
        {
            var history = alarmHistory[i];
            Console.WriteLine($"\t{i + 1}. {history.HistorySummary}, time
{history.Timestamp:g}");
        }
        if (!alarmHistory.Any())
        {
            Console.WriteLine($"\tNo alarm history data found for
{exampleAlarmName}.");
        }

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

    /// <summary>
    /// Add an anomaly detector.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task<SingleMetricAnomalyDetector> AddAnomalyDetector()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"15. Add an anomaly detector.");

        var customMetricNamespace = _configuration["customMetricNamespace"];
        var customMetricName = _configuration["customMetricName"];

        var detector = new SingleMetricAnomalyDetector
        {
            MetricName = customMetricName,
            Namespace = customMetricNamespace,
            Stat = "Maximum"
        };
        await _cloudWatchWrapper.PutAnomalyDetector(detector);
        Console.WriteLine($"\tAdded anomaly detector for metric
{customMetricName}.");

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

    /// <summary>
    /// Describe anomaly detectors.
    /// </summary>
    /// <returns>Async task.</returns>
```

```
    private static async Task DescribeAnomalyDetectors()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"16. Describe anomaly detectors in the current
account.");

        var customMetricNamespace = _configuration["customMetricNamespace"];
        var customMetricName = _configuration["customMetricName"];

        var detectors = await
_cloudWatchWrapper.DescribeAnomalyDetectors(customMetricNamespace,
customMetricName);

        for (int i = 0; i < detectors.Count; i++)
        {
            var detector = detectors[i];
            Console.WriteLine($"\t{i + 1}.
{detector.SingleMetricAnomalyDetector.MetricName}, state {detector.StateValue}");
        }

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

    /// <summary>
    /// Fetch and open a metrics image for a CloudWatch metric and namespace.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task GetAndOpenMetricImage()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine("17. Get a metric image from CloudWatch.");

        Console.WriteLine($"\tGetting Image data for custom metric.");
        var customMetricNamespace = _configuration["customMetricNamespace"];
        var customMetricName = _configuration["customMetricName"];

        var memoryStream = await
_cloudWatchWrapper.GetTimeSeriesMetricImage(customMetricNamespace,
customMetricName, "Maximum", 10);
        var file = _cloudWatchWrapper.SaveMetricImage(memoryStream, "MetricImages");

        ProcessStartInfo info = new ProcessStartInfo();

        Console.WriteLine($"\tFile saved as {Path.GetFileName(file)}.");
```

```
        Console.WriteLine($"\tPress enter to open the image.");
        Console.ReadLine();
        info.FileName = Path.Combine("ms-photos://", file);
        info.UseShellExecute = true;
        info.CreateNoWindow = true;
        info.Verb = string.Empty;

        Process.Start(info);

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

    /// <summary>
    /// Clean up created resources.
    /// </summary>
    /// <param name="metricNamespace">The namespace for metrics.</param>
    /// <param name="metric">The CloudWatch metric.</param>
    /// <returns>Async task.</returns>
    private static async Task CleanupResources()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"18. Clean up resources.");

        var dashboardName = _configuration["dashboardName"];
        if (GetYesNoResponse($"\tDelete dashboard {dashboardName}? (y/n)"))
        {
            Console.WriteLine($"\tDeleting dashboard.");
            var dashboardList = new List<string> { dashboardName };
            await _cloudWatchWrapper.DeleteDashboards(dashboardList);
        }

        var alarmName = _configuration["exampleAlarmName"];
        if (GetYesNoResponse($"\tDelete alarm {alarmName}? (y/n)"))
        {
            Console.WriteLine($"\tCleaning up alarms.");
            var alarms = new List<string> { alarmName };
            await _cloudWatchWrapper.DeleteAlarms(alarms);
        }

        if (GetYesNoResponse($"\tDelete anomaly detector? (y/n)") &&
anomalyDetector != null)
        {
            Console.WriteLine($"\tCleaning up anomaly detector.");
```

```
            await _cloudWatchWrapper.DeleteAnomalyDetector(
                anomalyDetector);
        }

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

    /// <summary>
    /// Get a yes or no response from the user.
    /// </summary>
    /// <param name="question">The question string to print on the console.</param>
    /// <returns>True if the user responds with a yes.</returns>
    private static bool GetYesNoResponse(string question)
    {
        Console.WriteLine(question);
        var ynResponse = Console.ReadLine();
        var response = ynResponse != null &&
                       ynResponse.Equals("y",
                           StringComparison.InvariantCultureIgnoreCase);
        return response;
    }
}
```

시나리오에서 CloudWatch 작업에 대해 사용하는 래퍼 메서드입니다.

```
/// <summary>
/// Wrapper class for Amazon CloudWatch methods.
/// </summary>
public class CloudWatchWrapper
{
    private readonly IAmazonCloudWatch _amazonCloudWatch;
    private readonly ILogger<CloudWatchWrapper> _logger;

    /// <summary>
    /// Constructor for the CloudWatch wrapper.
    /// </summary>
    /// <param name="amazonCloudWatch">The injected CloudWatch client.</param>
    /// <param name="logger">The injected logger for the wrapper.</param>
    public CloudWatchWrapper(IAmazonCloudWatch amazonCloudWatch,
 ILogger<CloudWatchWrapper> logger)

    {
```

```
        _logger = logger;
        _amazonCloudWatch = amazonCloudWatch;
    }

    /// <summary>
    /// List metrics available, optionally within a namespace.
    /// </summary>
    /// <param name="metricNamespace">Optional CloudWatch namespace to use when
listing metrics.</param>
    /// <param name="filter">Optional dimension filter.</param>
    /// <param name="metricName">Optional metric name filter.</param>
    /// <returns>The list of metrics.</returns>
    public async Task<List<Metric>> ListMetrics(string? metricNamespace = null,
DimensionFilter? filter = null, string? metricName = null)
    {
        var results = new List<Metric>();
        var paginateMetrics = _amazonCloudWatch.Paginators.ListMetrics(
            new ListMetricsRequest
            {
                Namespace = metricNamespace,
                Dimensions = filter != null ? new List<DimensionFilter> { filter } :
null,
                MetricName = metricName
            });
        // Get the entire list using the paginator.
        await foreach (var metric in paginateMetrics.Metrics)
        {
            results.Add(metric);
        }

        return results;
    }

    /// <summary>
    /// Wrapper to get statistics for a specific CloudWatch metric.
    /// </summary>
    /// <param name="metricNamespace">The namespace of the metric.</param>
    /// <param name="metricName">The name of the metric.</param>
    /// <param name="statistics">The list of statistics to include.</param>
    /// <param name="dimensions">The list of dimensions to include.</param>
    /// <param name="days">The number of days in the past to include.</param>
    /// <param name="period">The period for the data.</param>
    /// <returns>A list of DataPoint objects for the statistics.</returns>
    public async Task<List<Datapoint>> GetMetricStatistics(string metricNamespace,
```

```
        string metricName, List<string> statistics, List<Dimension> dimensions, int
 days, int period)
    {
        var metricStatistics = await _amazonCloudWatch.GetMetricStatisticsAsync(
            new GetMetricStatisticsRequest()
            {
                Namespace = metricNamespace,
                MetricName = metricName,
                Dimensions = dimensions,
                Statistics = statistics,
                StartTimeUtc = DateTime.UtcNow.AddDays(-days),
                EndTimeUtc = DateTime.UtcNow,
                Period = period
            });

        return metricStatistics.Datapoints ?? new List<Datapoint>();
    }

    /// <summary>
    /// Wrapper to create or add to a dashboard with metrics.
    /// </summary>
    /// <param name="dashboardName">The name for the dashboard.</param>
    /// <param name="dashboardBody">The metric data in JSON for the dashboard.</
param>
    /// <returns>A list of validation messages for the dashboard.</returns>
    public async Task<List<DashboardValidationMessage>> PutDashboard(string
 dashboardName,
        string dashboardBody)
    {
        // Updating a dashboard replaces all contents.
        // Best practice is to include a text widget indicating this dashboard was
 created programmatically.
        var dashboardResponse = await _amazonCloudWatch.PutDashboardAsync(
            new PutDashboardRequest()
            {
                DashboardName = dashboardName,
                DashboardBody = dashboardBody
            });

        return dashboardResponse.DashboardValidationMessages ?? new
 List<DashboardValidationMessage>();
    }
```

```
    /// <summary>
    /// Get information on a dashboard.
    /// </summary>
    /// <param name="dashboardName">The name of the dashboard.</param>
    /// <returns>A JSON object with dashboard information.</returns>
    public async Task<string> GetDashboard(string dashboardName)
    {
        var dashboardResponse = await _amazonCloudWatch.GetDashboardAsync(
            new GetDashboardRequest()
            {
                DashboardName = dashboardName
            });

        return dashboardResponse.DashboardBody;
    }


    /// <summary>
    /// Get a list of dashboards.
    /// </summary>
    /// <returns>A list of DashboardEntry objects.</returns>
    public async Task<List<DashboardEntry>> ListDashboards()
    {
        var results = new List<DashboardEntry>();
        var paginateDashboards = _amazonCloudWatch.Paginators.ListDashboards(
            new ListDashboardsRequest());
        // Get the entire list using the paginator.
        await foreach (var data in paginateDashboards.DashboardEntries)
        {
            results.Add(data);
        }

        return results;
    }

    /// <summary>
    /// Wrapper to add metric data to a CloudWatch metric.
    /// </summary>
    /// <param name="metricNamespace">The namespace of the metric.</param>
    /// <param name="metricData">A data object for the metric data.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> PutMetricData(string metricNamespace,
        List<MetricDatum> metricData)
    {
```

```
        var putDataResponse = await _amazonCloudWatch.PutMetricDataAsync(
            new PutMetricDataRequest()
            {
                MetricData = metricData,
                Namespace = metricNamespace,
            });

        return putDataResponse.HttpStatusCode == HttpStatusCode.OK;
    }

    /// <summary>
    /// Get an image for a metric graphed over time.
    /// </summary>
    /// <param name="metricNamespace">The namespace of the metric.</param>
    /// <param name="metric">The name of the metric.</param>
    /// <param name="stat">The name of the stat to chart.</param>
    /// <param name="period">The period to use for the chart.</param>
    /// <returns>A memory stream for the chart image.</returns>
    public async Task<MemoryStream> GetTimeSeriesMetricImage(string metricNamespace,
string metric, string stat, int period)
    {
        var metricImageWidget = new
        {
            title = "Example Metric Graph",
            view = "timeSeries",
            stacked = false,
            period = period,
            width = 1400,
            height = 600,
            metrics = new List<List<object>>
                { new() { metricNamespace, metric, new { stat } } }
        };

        var metricImageWidgetString = JsonSerializer.Serialize(metricImageWidget);
        var imageResponse = await _amazonCloudWatch.GetMetricWidgetImageAsync(
            new GetMetricWidgetImageRequest()
            {
                MetricWidget = metricImageWidgetString
            });

        return imageResponse.MetricWidgetImage;
    }

    /// <summary>
```

```
    /// Save a metric image to a file.
    /// </summary>
    /// <param name="memoryStream">The MemoryStream for the metric image.</param>
    /// <param name="metricName">The name of the metric.</param>
    /// <returns>The path to the file.</returns>
    public string SaveMetricImage(MemoryStream memoryStream, string metricName)
    {
        var metricFileName = $"{metricName}_{DateTime.Now.Ticks}.png";
        using var sr = new StreamReader(memoryStream);
        // Writes the memory stream to a file.
        File.WriteAllBytes(metricFileName, memoryStream.ToArray());
        var filePath = Path.Join(AppDomain.CurrentDomain.BaseDirectory,
            metricFileName);
        return filePath;
    }

    /// <summary>
    /// Get data for CloudWatch metrics.
    /// </summary>
    /// <param name="minutesOfData">The number of minutes of data to include.</
param>
    /// <param name="useDescendingTime">True to return the data descending by
 time.</param>
    /// <param name="endDateUtc">The end date for the data, in UTC.</param>
    /// <param name="maxDataPoints">The maximum data points to include.</param>
    /// <param name="dataQueries">Optional data queries to include.</param>
    /// <returns>A list of the requested metric data.</returns>
    public async Task<List<MetricDataResult>> GetMetricData(int minutesOfData, bool
 useDescendingTime, DateTime? endDateUtc = null,
        int maxDataPoints = 0, List<MetricDataQuery>? dataQueries = null)
    {
        var metricData = new List<MetricDataResult>();
        // If no end time is provided, use the current time for the end time.
        endDateUtc ??= DateTime.UtcNow;
        var timeZoneOffset =
 TimeZoneInfo.Local.GetUtcOffset(endDateUtc.Value.ToLocalTime());
        var startTimeUtc = endDateUtc.Value.AddMinutes(-minutesOfData);
        // The timezone string should be in the format +0000, so use the timezone
 offset to format it correctly.
        var timeZoneString = $"{timeZoneOffset.Hours:D2}
{timeZoneOffset.Minutes:D2}";
        // Add the plus sign for positive offsets.
        timeZoneString = timeZoneString.StartsWith('-') ? timeZoneString : "+" +
 timeZoneString;
```

```
        var paginatedMetricData = _amazonCloudWatch.Paginators.GetMetricData(
            new GetMetricDataRequest()
            {
                StartTimeUtc = startTimeUtc,
                EndTimeUtc = endDateUtc.Value,
                LabelOptions = new LabelOptions { Timezone = timeZoneString },
                ScanBy = useDescendingTime ? ScanBy.TimestampDescending :
ScanBy.TimestampAscending,
                MaxDatapoints = maxDataPoints,
                MetricDataQueries = dataQueries,
            });

        if (paginatedMetricData.MetricDataResults != null)
        {
            await foreach (var data in paginatedMetricData.MetricDataResults)
            {
                metricData.Add(data);
            }
        }

        return metricData;
    }

    /// <summary>
    /// Add a metric alarm to send an email when the metric passes a threshold.
    /// </summary>
    /// <param name="alarmDescription">A description of the alarm.</param>
    /// <param name="alarmName">The name for the alarm.</param>
    /// <param name="comparison">The type of comparison to use.</param>
    /// <param name="metricName">The name of the metric for the alarm.</param>
    /// <param name="metricNamespace">The namespace of the metric.</param>
    /// <param name="threshold">The threshold value for the alarm.</param>
    /// <param name="alarmActions">Optional actions to execute when in an alarm
state.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> PutMetricEmailAlarm(string alarmDescription, string
alarmName, ComparisonOperator comparison,
        string metricName, string metricNamespace, double threshold, List<string>
alarmActions = null!)
    {
        try
        {
            var putEmailAlarmResponse = await _amazonCloudWatch.PutMetricAlarmAsync(
                new PutMetricAlarmRequest()
```

```
                {
                    AlarmActions = alarmActions,
                    AlarmDescription = alarmDescription,
                    AlarmName = alarmName,
                    ComparisonOperator = comparison,
                    Threshold = threshold,
                    Namespace = metricNamespace,
                    MetricName = metricName,
                    EvaluationPeriods = 1,
                    Period = 10,
                    Statistic = new Statistic("Maximum"),
                    DatapointsToAlarm = 1,
                    TreatMissingData = "ignore"
                });
            return putEmailAlarmResponse.HttpStatusCode == HttpStatusCode.OK;
        }
        catch (LimitExceededException lex)
        {
            _logger.LogError(lex, $"Unable to add alarm {alarmName}. Alarm quota has
already been reached.");
        }

        return false;
    }

    /// <summary>
    /// Add specific email actions to a list of action strings for a CloudWatch
alarm.
    /// </summary>
    /// <param name="accountId">The AccountId for the alarm.</param>
    /// <param name="region">The region for the alarm.</param>
    /// <param name="emailTopicName">An Amazon Simple Notification Service (SNS)
topic for the alarm email.</param>
    /// <param name="alarmActions">Optional list of existing alarm actions to append
to.</param>
    /// <returns>A list of string actions for an alarm.</returns>
    public List<string> AddEmailAlarmAction(string accountId, string region,
        string emailTopicName, List<string>? alarmActions = null)
    {
        alarmActions ??= new List<string>();
        var snsAlarmAction = $"arn:aws:sns:{region}:{accountId}:{emailTopicName}";
        alarmActions.Add(snsAlarmAction);
        return alarmActions;
    }
```

```
    /// <summary>
    /// Describe the current alarms, optionally filtered by state.
    /// </summary>
    /// <param name="stateValue">Optional filter for alarm state.</param>
    /// <returns>The list of alarm data.</returns>
    public async Task<List<MetricAlarm>> DescribeAlarms(StateValue? stateValue =
null)
    {
        List<MetricAlarm> alarms = new List<MetricAlarm>();
        var paginatedDescribeAlarms = _amazonCloudWatch.Paginators.DescribeAlarms(
            new DescribeAlarmsRequest()
            {
                StateValue = stateValue
            });

        await foreach (var data in paginatedDescribeAlarms.MetricAlarms)
        {
            alarms.Add(data);
        }
        return alarms;
    }

    /// <summary>
    /// Describe the current alarms for a specific metric.
    /// </summary>
    /// <param name="metricNamespace">The namespace of the metric.</param>
    /// <param name="metricName">The name of the metric.</param>
    /// <returns>The list of alarm data.</returns>
    public async Task<List<MetricAlarm>> DescribeAlarmsForMetric(string
metricNamespace, string metricName)
    {
        var alarmsResult = await _amazonCloudWatch.DescribeAlarmsForMetricAsync(
            new DescribeAlarmsForMetricRequest()
            {
                Namespace = metricNamespace,
                MetricName = metricName
            });

        return alarmsResult.MetricAlarms ?? new List<MetricAlarm>();
    }

    /// <summary>
    /// Describe the history of an alarm for a number of days in the past.
```

```
    /// </summary>
    /// <param name="alarmName">The name of the alarm.</param>
    /// <param name="historyDays">The number of days in the past.</param>
    /// <returns>The list of alarm history data.</returns>
    public async Task<List<AlarmHistoryItem>> DescribeAlarmHistory(string alarmName,
int historyDays)
    {
        List<AlarmHistoryItem> alarmHistory = new List<AlarmHistoryItem>();
        var paginatedAlarmHistory =
_amazonCloudWatch.Paginators.DescribeAlarmHistory(
            new DescribeAlarmHistoryRequest()
            {
                AlarmName = alarmName,
                EndDateUtc = DateTime.UtcNow,
                HistoryItemType = HistoryItemType.StateUpdate,
                StartDateUtc = DateTime.UtcNow.AddDays(-historyDays)
            });

        await foreach (var data in paginatedAlarmHistory.AlarmHistoryItems)
        {
            alarmHistory.Add(data);
        }
        return alarmHistory;
    }

    /// <summary>
    /// Delete a list of alarms from CloudWatch.
    /// </summary>
    /// <param name="alarmNames">A list of names of alarms to delete.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DeleteAlarms(List<string> alarmNames)
    {
        var deleteAlarmsResult = await _amazonCloudWatch.DeleteAlarmsAsync(
            new DeleteAlarmsRequest()
            {
                AlarmNames = alarmNames
            });

        return deleteAlarmsResult.HttpStatusCode == HttpStatusCode.OK;
    }

    /// <summary>
    /// Disable the actions for a list of alarms from CloudWatch.
    /// </summary>
```

```
    /// <param name="alarmNames">A list of names of alarms.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DisableAlarmActions(List<string> alarmNames)
    {
        var disableAlarmActionsResult = await
_amazonCloudWatch.DisableAlarmActionsAsync(
            new DisableAlarmActionsRequest()
            {
                AlarmNames = alarmNames
            });

        return disableAlarmActionsResult.HttpStatusCode == HttpStatusCode.OK;
    }

    /// <summary>
    /// Enable the actions for a list of alarms from CloudWatch.
    /// </summary>
    /// <param name="alarmNames">A list of names of alarms.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> EnableAlarmActions(List<string> alarmNames)
    {
        var enableAlarmActionsResult = await
_amazonCloudWatch.EnableAlarmActionsAsync(
            new EnableAlarmActionsRequest()
            {
                AlarmNames = alarmNames
            });

        return enableAlarmActionsResult.HttpStatusCode == HttpStatusCode.OK;
    }

    /// <summary>
    /// Add an anomaly detector for a single metric.
    /// </summary>
    /// <param name="anomalyDetector">A single metric anomaly detector.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> PutAnomalyDetector(SingleMetricAnomalyDetector
anomalyDetector)
    {
        var putAlarmDetectorResult = await
_amazonCloudWatch.PutAnomalyDetectorAsync(
            new PutAnomalyDetectorRequest()
            {
                SingleMetricAnomalyDetector = anomalyDetector
```

```
            });

        return putAlarmDetectorResult.HttpStatusCode == HttpStatusCode.OK;
    }

    /// <summary>
    /// Describe anomaly detectors for a metric and namespace.
    /// </summary>
    /// <param name="metricNamespace">The namespace of the metric.</param>
    /// <param name="metricName">The metric of the anomaly detectors.</param>
    /// <returns>The list of detectors.</returns>
    public async Task<List<AnomalyDetector>> DescribeAnomalyDetectors(string
metricNamespace, string metricName)
    {
        List<AnomalyDetector> detectors = new List<AnomalyDetector>();
        var paginatedDescribeAnomalyDetectors =
_amazonCloudWatch.Paginators.DescribeAnomalyDetectors(
            new DescribeAnomalyDetectorsRequest()
            {
                MetricName = metricName,
                Namespace = metricNamespace
            });

        await foreach (var data in
paginatedDescribeAnomalyDetectors.AnomalyDetectors)
        {
            detectors.Add(data);
        }

        return detectors;
    }

    /// <summary>
    /// Delete a single metric anomaly detector.
    /// </summary>
    /// <param name="anomalyDetector">The anomaly detector to delete.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DeleteAnomalyDetector(SingleMetricAnomalyDetector
anomalyDetector)
    {
        var deleteAnomalyDetectorResponse = await
_amazonCloudWatch.DeleteAnomalyDetectorAsync(
            new DeleteAnomalyDetectorRequest()
            {
```

```
                SingleMetricAnomalyDetector = anomalyDetector
            });

        return deleteAnomalyDetectorResponse.HttpStatusCode == HttpStatusCode.OK;
    }

    /// <summary>
    /// Delete a list of CloudWatch dashboards.
    /// </summary>
    /// <param name="dashboardNames">List of dashboard names to delete.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DeleteDashboards(List<string> dashboardNames)
    {
        var deleteDashboardsResponse = await
 _amazonCloudWatch.DeleteDashboardsAsync(
            new DeleteDashboardsRequest()
            {
                DashboardNames = dashboardNames
            });

        return deleteDashboardsResponse.HttpStatusCode == HttpStatusCode.OK;
    }
}
```

시나리오에 대한 settings.json 값 예제.

```
{
  "dashboardName": "example-new-dashboard",
  "exampleAlarmName": "example-metric-alarm",
  "accountId": "1234567890",
  "region": "us-east-1",
  "emailTopic": "Default_CloudWatch_Alarms_Topic",
  "customMetricNamespace": "example-namespace",
  "customMetricName": "example-custom-metric",
  "dashboardExampleBody": {
    "widgets": [
      {
        "height": 6,
        "width": 6,
        "y": 0,
        "x": 0,
        "type": "text",
```

```
        "properties": {
          "markdown": "# Code Example Dashboard \nThis dashboard was created by
example code.\n"
        }
      },
      {
        "height": 8,
        "width": 8,
        "y": 0,
        "x": 6,
        "type": "metric",
        "properties": {
          "metrics": [
            [
              "AWS/Billing",
              "EstimatedCharges",
              "Currency",
              "USD",
              { "region": "us-east-1" }
            ]
          ],
          "view": "timeSeries",
          "region": "us-east-1",
          "stat": "Maximum",
          "period": 86400,
          "yAxis": {
            "left": {
              "min": 0,
              "max": 100
            }
          },
          "stacked": false,
          "title": "Estimated Billing",
          "setPeriodToTimeRange": false,
          "liveData": true,
          "sparkline": true,
          "trend": true
        }
      },
      {
        "height": 8,
        "width": 8,
        "y": 0,
        "x": 14,
```

```
        "type": "metric",
        "properties": {
          "metrics": [
            [ "AWS/Usage", "CallCount", "Type", "API", "Resource", "ListMetrics",
 "Service", "CloudWatch", "Class", "None" ],
            [ "...", "GetMetricStatistics", ".", ".", ".", "." ],
            [ "...", "GetMetricData", ".", ".", ".", "." ],
            [ "...", "PutDashboard", ".", ".", ".", "." ],
            [ "...", "PutMetricData", ".", ".", ".", "." ]
          ],
          "view": "timeSeries",
          "yAxis": {
            "left": {
              "min": 0,
              "max": 200
            }
          },
          "stacked": false,
          "region": "us-east-1",
          "stat": "Sum",
          "period": 300,
          "title": "CloudWatch Usage",
          "setPeriodToTimeRange": false,
          "liveData": true,
          "sparkline": true,
          "trend": true
        }
      }
    ]
  }
}
```

- API 세부 정보는 AWS SDK for .NET API 참조의 다음 항목을 참조하세요.
  - DeleteAlarms
  - DeleteAnomalyDetector
  - DeleteDashboards
  - DescribeAlarmHistory
  - DescribeAlarms
  - DescribeAlarmsForMetric

- DescribeAnomalyDetectors
- GetMetricData
- GetMetricStatistics
- GetMetricWidgetImage
- ListMetrics
- PutAnomalyDetector
- PutDashboard
- PutMetricAlarm
- PutMetricData

작업

`DeleteAlarms`

다음 코드 예시는 `DeleteAlarms`의 사용 방법을 보여 줍니다.

SDK for .NET (v4)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
/// <summary>
/// Delete a list of alarms from CloudWatch.
/// </summary>
/// <param name="alarmNames">A list of names of alarms to delete.</param>
/// <returns>True if successful.</returns>
public async Task<bool> DeleteAlarms(List<string> alarmNames)
{
    var deleteAlarmsResult = await _amazonCloudWatch.DeleteAlarmsAsync(
        new DeleteAlarmsRequest()
        {
            AlarmNames = alarmNames
        });

    return deleteAlarmsResult.HttpStatusCode == HttpStatusCode.OK;
```

```
    }
```

- API 세부 정보는 AWS SDK for .NET API 참조의 DeleteAlarms를 참조하세요.

`DeleteAnomalyDetector`

다음 코드 예시는 `DeleteAnomalyDetector`의 사용 방법을 보여 줍니다.

SDK for .NET (v4)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
   /// <summary>
   /// Delete a single metric anomaly detector.
   /// </summary>
   /// <param name="anomalyDetector">The anomaly detector to delete.</param>
   /// <returns>True if successful.</returns>
   public async Task<bool> DeleteAnomalyDetector(SingleMetricAnomalyDetector
anomalyDetector)
   {
       var deleteAnomalyDetectorResponse = await
_amazonCloudWatch.DeleteAnomalyDetectorAsync(
           new DeleteAnomalyDetectorRequest()
           {
               SingleMetricAnomalyDetector = anomalyDetector
           });

       return deleteAnomalyDetectorResponse.HttpStatusCode == HttpStatusCode.OK;
   }
```

- API 세부 정보는 AWS SDK for .NET API 참조의 DeleteAnomalyDetector를 참조하세요.

`DeleteDashboards`

다음 코드 예시는 `DeleteDashboards`의 사용 방법을 보여 줍니다.

SDK for .NET (v4)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
    /// <summary>
    /// Delete a list of CloudWatch dashboards.
    /// </summary>
    /// <param name="dashboardNames">List of dashboard names to delete.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DeleteDashboards(List<string> dashboardNames)
    {
        var deleteDashboardsResponse = await
_amazonCloudWatch.DeleteDashboardsAsync(
            new DeleteDashboardsRequest()
            {
                DashboardNames = dashboardNames
            });

        return deleteDashboardsResponse.HttpStatusCode == HttpStatusCode.OK;
    }
```

- API 세부 정보는 AWS SDK for .NET API 참조의 DeleteDashboards를 참조하세요.

`DescribeAlarmHistory`

다음 코드 예시는 `DescribeAlarmHistory`의 사용 방법을 보여 줍니다.

SDK for .NET (v4)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
    /// <summary>
    /// Describe the history of an alarm for a number of days in the past.
    /// </summary>
    /// <param name="alarmName">The name of the alarm.</param>
    /// <param name="historyDays">The number of days in the past.</param>
    /// <returns>The list of alarm history data.</returns>
    public async Task<List<AlarmHistoryItem>> DescribeAlarmHistory(string alarmName,
int historyDays)
    {
        List<AlarmHistoryItem> alarmHistory = new List<AlarmHistoryItem>();
        var paginatedAlarmHistory =
_amazonCloudWatch.Paginators.DescribeAlarmHistory(
            new DescribeAlarmHistoryRequest()
            {
                AlarmName = alarmName,
                EndDateUtc = DateTime.UtcNow,
                HistoryItemType = HistoryItemType.StateUpdate,
                StartDateUtc = DateTime.UtcNow.AddDays(-historyDays)
            });

        await foreach (var data in paginatedAlarmHistory.AlarmHistoryItems)
        {
            alarmHistory.Add(data);
        }
        return alarmHistory;
    }
```

- API 세부 정보는 AWS SDK for .NET API 참조의 DescribeAlarmHistory를 참조하세요.

`DescribeAlarms`

다음 코드 예시는 `DescribeAlarms`의 사용 방법을 보여 줍니다.

SDK for .NET (v4)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
    /// <summary>
    /// Describe the current alarms, optionally filtered by state.
    /// </summary>
    /// <param name="stateValue">Optional filter for alarm state.</param>
    /// <returns>The list of alarm data.</returns>
    public async Task<List<MetricAlarm>> DescribeAlarms(StateValue? stateValue =
null)
    {
        List<MetricAlarm> alarms = new List<MetricAlarm>();
        var paginatedDescribeAlarms = _amazonCloudWatch.Paginators.DescribeAlarms(
            new DescribeAlarmsRequest()
            {
                StateValue = stateValue
            });

        await foreach (var data in paginatedDescribeAlarms.MetricAlarms)
        {
            alarms.Add(data);
        }
        return alarms;
    }
```

- API 세부 정보는 AWS SDK for .NET API 참조의 DescribeAlarms를 참조하세요.

`DescribeAlarmsForMetric`

다음 코드 예시는 `DescribeAlarmsForMetric`의 사용 방법을 보여 줍니다.

SDK for .NET (v4)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
    /// <summary>
    /// Describe the current alarms for a specific metric.
    /// </summary>
    /// <param name="metricNamespace">The namespace of the metric.</param>
```

```
    /// <param name="metricName">The name of the metric.</param>
    /// <returns>The list of alarm data.</returns>
    public async Task<List<MetricAlarm>> DescribeAlarmsForMetric(string
metricNamespace, string metricName)
    {
        var alarmsResult = await _amazonCloudWatch.DescribeAlarmsForMetricAsync(
            new DescribeAlarmsForMetricRequest()
            {
                Namespace = metricNamespace,
                MetricName = metricName
            });

        return alarmsResult.MetricAlarms ?? new List<MetricAlarm>();
    }
```

- API 세부 정보는 AWS SDK for .NET API 참조의 DescribeAlarmsForMetric을 참조하세요.

`DescribeAnomalyDetectors`

다음 코드 예시는 `DescribeAnomalyDetectors`의 사용 방법을 보여 줍니다.

SDK for .NET (v4)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
    /// <summary>
    /// Describe anomaly detectors for a metric and namespace.
    /// </summary>
    /// <param name="metricNamespace">The namespace of the metric.</param>
    /// <param name="metricName">The metric of the anomaly detectors.</param>
    /// <returns>The list of detectors.</returns>
    public async Task<List<AnomalyDetector>> DescribeAnomalyDetectors(string
metricNamespace, string metricName)
    {
        List<AnomalyDetector> detectors = new List<AnomalyDetector>();
        var paginatedDescribeAnomalyDetectors =
_amazonCloudWatch.Paginators.DescribeAnomalyDetectors(
```

```
            new DescribeAnomalyDetectorsRequest()
            {
                MetricName = metricName,
                Namespace = metricNamespace
            });

        await foreach (var data in
paginatedDescribeAnomalyDetectors.AnomalyDetectors)
        {
            detectors.Add(data);
        }

        return detectors;
    }
```

- API 세부 정보는 AWS SDK for .NET API 참조의 DescribeAnomalyDetectors를 참조하세요.

`DisableAlarmActions`

다음 코드 예시는 `DisableAlarmActions`의 사용 방법을 보여 줍니다.

SDK for .NET (v4)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
    /// <summary>
    /// Disable the actions for a list of alarms from CloudWatch.
    /// </summary>
    /// <param name="alarmNames">A list of names of alarms.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DisableAlarmActions(List<string> alarmNames)
    {
        var disableAlarmActionsResult = await
_amazonCloudWatch.DisableAlarmActionsAsync(
            new DisableAlarmActionsRequest()
            {
                AlarmNames = alarmNames
```

```
            });

        return disableAlarmActionsResult.HttpStatusCode == HttpStatusCode.OK;
    }
```

- API 세부 정보에 대한 내용은 AWS SDK for .NET API 참조의 DisableAlarmActions를 참조하세요.

`EnableAlarmActions`

다음 코드 예시는 `EnableAlarmActions`의 사용 방법을 보여 줍니다.

SDK for .NET (v4)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
    /// <summary>
    /// Enable the actions for a list of alarms from CloudWatch.
    /// </summary>
    /// <param name="alarmNames">A list of names of alarms.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> EnableAlarmActions(List<string> alarmNames)
    {
        var enableAlarmActionsResult = await
_amazonCloudWatch.EnableAlarmActionsAsync(
            new EnableAlarmActionsRequest()
            {
                AlarmNames = alarmNames
            });

        return enableAlarmActionsResult.HttpStatusCode == HttpStatusCode.OK;
    }
```

- API 세부 정보는 AWS SDK for .NET API 참조의 EnableAlarmActions를 참조하세요.

GetDashboard

다음 코드 예시는 GetDashboard의 사용 방법을 보여 줍니다.

SDK for .NET (v4)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
/// <summary>
/// Get information on a dashboard.
/// </summary>
/// <param name="dashboardName">The name of the dashboard.</param>
/// <returns>A JSON object with dashboard information.</returns>
public async Task<string> GetDashboard(string dashboardName)
{
    var dashboardResponse = await _amazonCloudWatch.GetDashboardAsync(
        new GetDashboardRequest()
        {
            DashboardName = dashboardName
        });

    return dashboardResponse.DashboardBody;
}
```

- API 세부 정보는 AWS SDK for .NET API 참조의 GetDashboard를 참조하세요.

GetMetricData

다음 코드 예시는 GetMetricData의 사용 방법을 보여 줍니다.

SDK for .NET (v4)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
    /// <summary>
    /// Get data for CloudWatch metrics.
    /// </summary>
    /// <param name="minutesOfData">The number of minutes of data to include.</
param>
    /// <param name="useDescendingTime">True to return the data descending by
 time.</param>
    /// <param name="endDateUtc">The end date for the data, in UTC.</param>
    /// <param name="maxDataPoints">The maximum data points to include.</param>
    /// <param name="dataQueries">Optional data queries to include.</param>
    /// <returns>A list of the requested metric data.</returns>
    public async Task<List<MetricDataResult>> GetMetricData(int minutesOfData, bool
 useDescendingTime, DateTime? endDateUtc = null,
        int maxDataPoints = 0, List<MetricDataQuery>? dataQueries = null)
    {
        var metricData = new List<MetricDataResult>();
        // If no end time is provided, use the current time for the end time.
        endDateUtc ??= DateTime.UtcNow;
        var timeZoneOffset =
 TimeZoneInfo.Local.GetUtcOffset(endDateUtc.Value.ToLocalTime());
        var startTimeUtc = endDateUtc.Value.AddMinutes(-minutesOfData);
        // The timezone string should be in the format +0000, so use the timezone
 offset to format it correctly.
        var timeZoneString = $"{timeZoneOffset.Hours:D2}
{timeZoneOffset.Minutes:D2}";
        // Add the plus sign for positive offsets.
        timeZoneString = timeZoneString.StartsWith('-') ? timeZoneString : "+" +
 timeZoneString;
        var paginatedMetricData = _amazonCloudWatch.Paginators.GetMetricData(
            new GetMetricDataRequest()
            {
                StartTimeUtc = startTimeUtc,
                EndTimeUtc = endDateUtc.Value,
                LabelOptions = new LabelOptions { Timezone = timeZoneString },
```

```
                ScanBy = useDescendingTime ? ScanBy.TimestampDescending :
ScanBy.TimestampAscending,
                MaxDatapoints = maxDataPoints,
                MetricDataQueries = dataQueries,
            });

        if (paginatedMetricData.MetricDataResults != null)
        {
            await foreach (var data in paginatedMetricData.MetricDataResults)
            {
                metricData.Add(data);
            }
        }

        return metricData;
    }
```

- API 세부 정보는 AWS SDK for .NET API 참조의 GetMetricData를 참조하세요.

`GetMetricStatistics`

다음 코드 예시는 `GetMetricStatistics`의 사용 방법을 보여 줍니다.

SDK for .NET (v4)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
    /// <summary>
    /// Get billing statistics using a call to a wrapper class.
    /// </summary>
    /// <returns>A collection of billing statistics.</returns>
    private static async Task<List<Datapoint>> SetupBillingStatistics()
    {
        // Make a request for EstimatedCharges with a period of one day for the past
seven days.
        var billingStatistics = await _cloudWatchWrapper.GetMetricStatistics(
```

```
            "AWS/Billing",
            "EstimatedCharges",
            new List<string>() { "Maximum" },
            new List<Dimension>() { new Dimension { Name = "Currency", Value =
"USD" } },
            7,
            86400);

        billingStatistics = billingStatistics.OrderBy(n => n.Timestamp).ToList();

        return billingStatistics;
    }

    /// <summary>
    /// Wrapper to get statistics for a specific CloudWatch metric.
    /// </summary>
    /// <param name="metricNamespace">The namespace of the metric.</param>
    /// <param name="metricName">The name of the metric.</param>
    /// <param name="statistics">The list of statistics to include.</param>
    /// <param name="dimensions">The list of dimensions to include.</param>
    /// <param name="days">The number of days in the past to include.</param>
    /// <param name="period">The period for the data.</param>
    /// <returns>A list of DataPoint objects for the statistics.</returns>
    public async Task<List<Datapoint>> GetMetricStatistics(string metricNamespace,
        string metricName, List<string> statistics, List<Dimension> dimensions, int
days, int period)
    {
        var metricStatistics = await _amazonCloudWatch.GetMetricStatisticsAsync(
            new GetMetricStatisticsRequest()
            {
                Namespace = metricNamespace,
                MetricName = metricName,
                Dimensions = dimensions,
                Statistics = statistics,
                StartTimeUtc = DateTime.UtcNow.AddDays(-days),
                EndTimeUtc = DateTime.UtcNow,
                Period = period
            });

        return metricStatistics.Datapoints ?? new List<Datapoint>();
    }
```

- API 세부 정보는 AWS SDK for .NET API 참조의 GetMetricStatistics를 참조하세요.

GetMetricWidgetImage

다음 코드 예시는 GetMetricWidgetImage의 사용 방법을 보여 줍니다.

SDK for .NET (v4)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
    /// <summary>
    /// Get an image for a metric graphed over time.
    /// </summary>
    /// <param name="metricNamespace">The namespace of the metric.</param>
    /// <param name="metric">The name of the metric.</param>
    /// <param name="stat">The name of the stat to chart.</param>
    /// <param name="period">The period to use for the chart.</param>
    /// <returns>A memory stream for the chart image.</returns>
    public async Task<MemoryStream> GetTimeSeriesMetricImage(string metricNamespace,
string metric, string stat, int period)
    {
        var metricImageWidget = new
        {
            title = "Example Metric Graph",
            view = "timeSeries",
            stacked = false,
            period = period,
            width = 1400,
            height = 600,
            metrics = new List<List<object>>
                { new() { metricNamespace, metric, new { stat } } }
        };

        var metricImageWidgetString = JsonSerializer.Serialize(metricImageWidget);
        var imageResponse = await _amazonCloudWatch.GetMetricWidgetImageAsync(
            new GetMetricWidgetImageRequest()
            {
                MetricWidget = metricImageWidgetString
            });

        return imageResponse.MetricWidgetImage;
```

```
    }

    /// <summary>
    /// Save a metric image to a file.
    /// </summary>
    /// <param name="memoryStream">The MemoryStream for the metric image.</param>
    /// <param name="metricName">The name of the metric.</param>
    /// <returns>The path to the file.</returns>
    public string SaveMetricImage(MemoryStream memoryStream, string metricName)
    {
        var metricFileName = $"{metricName}_{DateTime.Now.Ticks}.png";
        using var sr = new StreamReader(memoryStream);
        // Writes the memory stream to a file.
        File.WriteAllBytes(metricFileName, memoryStream.ToArray());
        var filePath = Path.Join(AppDomain.CurrentDomain.BaseDirectory,
            metricFileName);
        return filePath;
    }
```

- API 세부 정보는 AWS SDK for .NET API 참조의 GetMetricWidgetImage를 참조하세요.

`ListDashboards`

다음 코드 예시는 `ListDashboards`의 사용 방법을 보여 줍니다.

SDK for .NET (v4)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
    /// <summary>
    /// Get a list of dashboards.
    /// </summary>
    /// <returns>A list of DashboardEntry objects.</returns>
    public async Task<List<DashboardEntry>> ListDashboards()
    {
        var results = new List<DashboardEntry>();
```

```
        var paginateDashboards = _amazonCloudWatch.Paginators.ListDashboards(
            new ListDashboardsRequest());
        // Get the entire list using the paginator.
        await foreach (var data in paginateDashboards.DashboardEntries)
        {
            results.Add(data);
        }

        return results;
    }
```

- API 세부 정보는 AWS SDK for .NET API 참조의 ListDashboards를 참조하세요.

`ListMetrics`

다음 코드 예시는 `ListMetrics`의 사용 방법을 보여 줍니다.

SDK for .NET (v4)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
    /// <summary>
    /// List metrics available, optionally within a namespace.
    /// </summary>
    /// <param name="metricNamespace">Optional CloudWatch namespace to use when
listing metrics.</param>
    /// <param name="filter">Optional dimension filter.</param>
    /// <param name="metricName">Optional metric name filter.</param>
    /// <returns>The list of metrics.</returns>
    public async Task<List<Metric>> ListMetrics(string? metricNamespace = null,
DimensionFilter? filter = null, string? metricName = null)
    {
        var results = new List<Metric>();
        var paginateMetrics = _amazonCloudWatch.Paginators.ListMetrics(
            new ListMetricsRequest
            {
                Namespace = metricNamespace,
```

```
                Dimensions = filter != null ? new List<DimensionFilter> { filter } :
null,
                MetricName = metricName
            });
        // Get the entire list using the paginator.
        await foreach (var metric in paginateMetrics.Metrics)
        {
            results.Add(metric);
        }

        return results;
    }
```

- API 세부 정보는 AWS SDK for .NET API 참조의 ListMetrics를 참조하세요.

`PutAnomalyDetector`

다음 코드 예시는 `PutAnomalyDetector`의 사용 방법을 보여 줍니다.

SDK for .NET (v4)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
    /// <summary>
    /// Add an anomaly detector for a single metric.
    /// </summary>
    /// <param name="anomalyDetector">A single metric anomaly detector.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> PutAnomalyDetector(SingleMetricAnomalyDetector
anomalyDetector)
    {
        var putAlarmDetectorResult = await
_amazonCloudWatch.PutAnomalyDetectorAsync(
            new PutAnomalyDetectorRequest()
            {
                SingleMetricAnomalyDetector = anomalyDetector
            });
```

```
        return putAlarmDetectorResult.HttpStatusCode == HttpStatusCode.OK;
    }
```

- API 세부 정보는 AWS SDK for .NET API 참조의 PutAnomalyDetector를 참조하세요.

`PutDashboard`

다음 코드 예시는 `PutDashboard`의 사용 방법을 보여 줍니다.

SDK for .NET (v4)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
/// <summary>
/// Set up a dashboard using a call to the wrapper class.
/// </summary>
/// <param name="customMetricNamespace">The metric namespace.</param>
/// <param name="customMetricName">The metric name.</param>
/// <param name="dashboardName">The name of the dashboard.</param>
/// <returns>A list of validation messages.</returns>
private static async Task<List<DashboardValidationMessage>> SetupDashboard(
    string customMetricNamespace, string customMetricName, string dashboardName)
{
    // Get the dashboard model from configuration.
    var newDashboard = new DashboardModel();
    _configuration.GetSection("dashboardExampleBody").Bind(newDashboard);

    // Add a new metric to the dashboard.
    newDashboard.Widgets.Add(new Widget
    {
        Height = 8,
        Width = 8,
        Y = 8,
        X = 0,
        Type = "metric",
```

```
            Properties = new Properties
            {
                Metrics = new List<List<object>>
                    { new() { customMetricNamespace, customMetricName } },
                View = "timeSeries",
                Region = "us-east-1",
                Stat = "Sum",
                Period = 86400,
                YAxis = new YAxis { Left = new Left { Min = 0, Max = 100 } },
                Title = "Custom Metric Widget",
                LiveData = true,
                Sparkline = true,
                Trend = true,
                Stacked = false,
                SetPeriodToTimeRange = false
            }
        });

        var newDashboardString = JsonSerializer.Serialize(newDashboard,
            new JsonSerializerOptions
            { DefaultIgnoreCondition = JsonIgnoreCondition.WhenWritingNull });
        var validationMessages =
            await _cloudWatchWrapper.PutDashboard(dashboardName,
 newDashboardString);

        return validationMessages;
    }

    /// <summary>
    /// Wrapper to create or add to a dashboard with metrics.
    /// </summary>
    /// <param name="dashboardName">The name for the dashboard.</param>
    /// <param name="dashboardBody">The metric data in JSON for the dashboard.</
param>
    /// <returns>A list of validation messages for the dashboard.</returns>
    public async Task<List<DashboardValidationMessage>> PutDashboard(string
 dashboardName,
        string dashboardBody)
    {
        // Updating a dashboard replaces all contents.
        // Best practice is to include a text widget indicating this dashboard was
 created programmatically.
        var dashboardResponse = await _amazonCloudWatch.PutDashboardAsync(
            new PutDashboardRequest()
```

```
            {
                DashboardName = dashboardName,
                DashboardBody = dashboardBody
            });

        return dashboardResponse.DashboardValidationMessages ?? new
List<DashboardValidationMessage>();
    }
```

- API 세부 정보는 AWS SDK for .NET API 참조의 PutDashboard를 참조하세요.

`PutMetricAlarm`

다음 코드 예시는 `PutMetricAlarm`의 사용 방법을 보여 줍니다.

SDK for .NET (v4)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
    /// <summary>
    /// Add a metric alarm to send an email when the metric passes a threshold.
    /// </summary>
    /// <param name="alarmDescription">A description of the alarm.</param>
    /// <param name="alarmName">The name for the alarm.</param>
    /// <param name="comparison">The type of comparison to use.</param>
    /// <param name="metricName">The name of the metric for the alarm.</param>
    /// <param name="metricNamespace">The namespace of the metric.</param>
    /// <param name="threshold">The threshold value for the alarm.</param>
    /// <param name="alarmActions">Optional actions to execute when in an alarm
state.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> PutMetricEmailAlarm(string alarmDescription, string
alarmName, ComparisonOperator comparison,
        string metricName, string metricNamespace, double threshold, List<string>
alarmActions = null!)
    {
```

```
        try
        {
            var putEmailAlarmResponse = await _amazonCloudWatch.PutMetricAlarmAsync(
                new PutMetricAlarmRequest()
                {
                    AlarmActions = alarmActions,
                    AlarmDescription = alarmDescription,
                    AlarmName = alarmName,
                    ComparisonOperator = comparison,
                    Threshold = threshold,
                    Namespace = metricNamespace,
                    MetricName = metricName,
                    EvaluationPeriods = 1,
                    Period = 10,
                    Statistic = new Statistic("Maximum"),
                    DatapointsToAlarm = 1,
                    TreatMissingData = "ignore"
                });
            return putEmailAlarmResponse.HttpStatusCode == HttpStatusCode.OK;
        }
        catch (LimitExceededException lex)
        {
            _logger.LogError(lex, $"Unable to add alarm {alarmName}. Alarm quota has
already been reached.");
        }

        return false;
    }

    /// <summary>
    /// Add specific email actions to a list of action strings for a CloudWatch
alarm.
    /// </summary>
    /// <param name="accountId">The AccountId for the alarm.</param>
    /// <param name="region">The region for the alarm.</param>
    /// <param name="emailTopicName">An Amazon Simple Notification Service (SNS)
topic for the alarm email.</param>
    /// <param name="alarmActions">Optional list of existing alarm actions to append
to.</param>
    /// <returns>A list of string actions for an alarm.</returns>
    public List<string> AddEmailAlarmAction(string accountId, string region,
        string emailTopicName, List<string>? alarmActions = null)
    {
        alarmActions ??= new List<string>();
```

```
        var snsAlarmAction = $"arn:aws:sns:{region}:{accountId}:{emailTopicName}";
        alarmActions.Add(snsAlarmAction);
        return alarmActions;
    }
```

- API 세부 정보는 AWS SDK for .NET API 참조의 PutMetricAlarm을 참조하세요.

`PutMetricData`

다음 코드 예시는 `PutMetricData`의 사용 방법을 보여 줍니다.

SDK for .NET (v4)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
    /// <summary>
    /// Add some metric data using a call to a wrapper class.
    /// </summary>
    /// <param name="customMetricName">The metric name.</param>
    /// <param name="customMetricNamespace">The metric namespace.</param>
    /// <returns></returns>
    private static async Task<List<MetricDatum>> PutRandomMetricData(string
customMetricName,
        string customMetricNamespace)
    {
        List<MetricDatum> customData = new List<MetricDatum>();
        Random rnd = new Random();

        // Add 10 random values up to 100, starting with a timestamp 15 minutes in
the past.
        var utcNowMinus15 = DateTime.UtcNow.AddMinutes(-15);
        for (int i = 0; i < 10; i++)
        {
            var metricValue = rnd.Next(0, 100);
            customData.Add(
```

```
                new MetricDatum
                {
                    MetricName = customMetricName,
                    Value = metricValue,
                    TimestampUtc = utcNowMinus15.AddMinutes(i)
                }
            );
        }

        await _cloudWatchWrapper.PutMetricData(customMetricNamespace, customData);
        return customData;
    }

    /// <summary>
    /// Wrapper to add metric data to a CloudWatch metric.
    /// </summary>
    /// <param name="metricNamespace">The namespace of the metric.</param>
    /// <param name="metricData">A data object for the metric data.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> PutMetricData(string metricNamespace,
        List<MetricDatum> metricData)
    {
        var putDataResponse = await _amazonCloudWatch.PutMetricDataAsync(
            new PutMetricDataRequest()
            {
                MetricData = metricData,
                Namespace = metricNamespace,
            });

        return putDataResponse.HttpStatusCode == HttpStatusCode.OK;
    }
```

- API 세부 정보는 AWS SDK for .NET API 참조의 PutMetricData를 참조하세요.

SDK for .NET (v4)를 사용한 CloudWatch Logs 예제

다음 코드 예제에서는 CloudWatch Logs와 함께 AWS SDK for .NET (v4)를 사용하여 작업을 수행하고 일반적인 시나리오를 구현하는 방법을 보여줍니다.

작업은 대규모 프로그램에서 발췌한 코드이며 컨텍스트에 맞춰 실행해야 합니다. 작업은 개별 서비스 함수를 직접 호출하는 방법을 보여주며, 관련 시나리오의 컨텍스트에 맞는 작업을 볼 수 있습니다.

시나리오는 동일한 서비스 내에서 또는 다른 AWS 서비스와 결합된 상태에서 여러 함수를 직접적으로 호출하여 특정 태스크를 수행하는 방법을 보여주는 코드 예제입니다.

각 예시에는 전체 소스 코드에 대한 링크가 포함되어 있으며, 여기에서 컨텍스트에 맞춰 코드를 설정하고 실행하는 방법에 대한 지침을 찾을 수 있습니다.

주제

- 작업
- 시나리오

작업

`GetQueryResults`

다음 코드 예시는 `GetQueryResults`의 사용 방법을 보여 줍니다.

SDK for .NET (v4)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
    /// <summary>
    /// Gets the results of a CloudWatch Logs Insights query.
    /// </summary>
    /// <param name="queryId">The ID of the query.</param>
    /// <returns>The query results response.</returns>
    public async Task<GetQueryResultsResponse?> GetQueryResultsAsync(string queryId)
    {
        try
        {
            var request = new GetQueryResultsRequest
            {
                QueryId = queryId
            };

            var response = await
_amazonCloudWatchLogs.GetQueryResultsAsync(request);
```

```
            return response;
        }
        catch (ResourceNotFoundException ex)
        {
            _logger.LogError($"Query not found: {ex.Message}");
            return null;
        }
        catch (Exception ex)
        {
            _logger.LogError($"An error occurred while getting query results:
 {ex.Message}");
            return null;
        }
    }
```

- API 세부 정보는 AWS SDK for .NET API 참조의 GetQueryResults를 참조하세요.

`StartQuery`

다음 코드 예시는 `StartQuery`의 사용 방법을 보여 줍니다.

SDK for .NET (v4)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
    /// <summary>
    /// Starts a CloudWatch Logs Insights query.
    /// </summary>
    /// <param name="logGroupName">The name of the log group to query.</param>
    /// <param name="queryString">The CloudWatch Logs Insights query string.</param>
    /// <param name="startTime">The start time for the query (seconds since
 epoch).</param>
    /// <param name="endTime">The end time for the query (seconds since epoch).</
param>
    /// <param name="limit">The maximum number of results to return.</param>
    /// <returns>The query ID if successful, null otherwise.</returns>
```

```
    public async Task<string?> StartQueryAsync(
        string logGroupName,
        string queryString,
        long startTime,
        long endTime,
        int limit = 10000)
    {
        try
        {
            var request = new StartQueryRequest
            {
                LogGroupName = logGroupName,
                QueryString = queryString,
                StartTime = startTime,
                EndTime = endTime,
                Limit = limit
            };

            var response = await _amazonCloudWatchLogs.StartQueryAsync(request);
            return response.QueryId;
        }
        catch (InvalidParameterException ex)
        {
            _logger.LogError($"Invalid parameter for query: {ex.Message}");
            return null;
        }
        catch (ResourceNotFoundException ex)
        {
            _logger.LogError($"Log group not found: {ex.Message}");
            return null;
        }
        catch (Exception ex)
        {
            _logger.LogError($"An error occurred while starting query:
{ex.Message}");
            return null;
        }
    }
```

- API 세부 정보는 AWS SDK for .NET API 참조의 StartQuery를 참조하세요.

시나리오

대용량 쿼리 실행

다음 코드 예시에서는 CloudWatch Logs를 사용하여 1만 개 이상의 레코드를 쿼리하는 방법을 보여줍니다.

SDK for .NET (v4)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

이는 대규모 쿼리 시나리오를 보여주는 기본 워크플로입니다.

```
using System.Diagnostics;
using System.Text.RegularExpressions;
using Amazon.CloudFormation;
using Amazon.CloudFormation.Model;
using Amazon.CloudWatchLogs;
using Amazon.CloudWatchLogs.Model;
using CloudWatchLogsActions;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;
using Microsoft.Extensions.Logging;

namespace CloudWatchLogsScenario;

public class LargeQueryWorkflow
{
    /*
    Before running this .NET code example, set up your development environment,
 including your credentials.
    This .NET code example performs the following tasks for the CloudWatch Logs
 Large Query workflow:

    1. Prepare the Application:
       - Prompt the user to deploy CloudFormation stack and generate sample logs.
       - Deploy the CloudFormation template for resource creation.
       - Generate 50,000 sample log entries using CloudWatch Logs API.
```

```
       - Wait 5 minutes for logs to be fully ingested.

    2. Execute Large Query:
       - Perform recursive queries to retrieve all logs using binary search.
       - Display progress for each query executed.
       - Show total execution time and logs found.

    3. Clean up:
       - Prompt the user to delete the CloudFormation stack and all resources.
       - Destroy the CloudFormation stack and wait until removed.
    */

    public static ILogger<LargeQueryWorkflow> _logger = null!;
    public static CloudWatchLogsWrapper _wrapper = null!;
    public static IAmazonCloudFormation _amazonCloudFormation = null!;

    private static string _logGroupName = "/workflows/cloudwatch-logs/large-query";
    private static string _logStreamName = "stream1";
    private static long _queryStartDate;
    private static long _queryEndDate;

    public static bool _interactive = true;
    public static string _stackName = "CloudWatchLargeQueryStack";
    private static string _stackResourcePath = "../../../../../../../scenarios/
features/cloudwatch_logs_large_query/resources/stack.yaml";

    public static async Task Main(string[] args)
    {
        using var host = Host.CreateDefaultBuilder(args)
            .ConfigureLogging(logging =>
                logging.AddFilter("System", LogLevel.Debug)
                    .AddFilter("Microsoft", LogLevel.Information))
            .ConfigureServices((_, services) =>
                services.AddAWSService<IAmazonCloudWatchLogs>()
                    .AddAWSService<IAmazonCloudFormation>()
                    .AddTransient<CloudWatchLogsWrapper>()
            )
            .Build();

        if (_interactive)
        {
            _logger = LoggerFactory.Create(builder => { builder.AddConsole(); })
                .CreateLogger<LargeQueryWorkflow>();
```

```
            _wrapper = host.Services.GetRequiredService<CloudWatchLogsWrapper>();
            _amazonCloudFormation =
host.Services.GetRequiredService<IAmazonCloudFormation>();
        }

        Console.WriteLine(new string('-', 80));
        Console.WriteLine("Welcome to the CloudWatch Logs Large Query Scenario.");
        Console.WriteLine(new string('-', 80));
        Console.WriteLine("This scenario demonstrates how to perform large-scale
queries on");
        Console.WriteLine("CloudWatch Logs using recursive binary search to retrieve
more than");
        Console.WriteLine("the 10,000 result limit.");
        Console.WriteLine();

        try
        {
            Console.WriteLine(new string('-', 80));
            var prepareSuccess = await PrepareApplication();
            Console.WriteLine(new string('-', 80));

            if (prepareSuccess)
            {
                Console.WriteLine(new string('-', 80));
                await ExecuteLargeQuery();
                Console.WriteLine(new string('-', 80));
            }

            Console.WriteLine(new string('-', 80));
            await Cleanup();
            Console.WriteLine(new string('-', 80));
        }
        catch (Exception ex)
        {
            _logger.LogError(ex, "There was a problem with the scenario, initiating
cleanup...");
            _interactive = false;
            await Cleanup();
        }

        Console.WriteLine("CloudWatch Logs Large Query scenario completed.");
    }

    /// <summary>
```

```
    /// Runs the scenario workflow. Used for testing.
    /// </summary>
    public static async Task RunScenario()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine("Welcome to the CloudWatch Logs Large Query Scenario.");
        Console.WriteLine(new string('-', 80));
        Console.WriteLine("This scenario demonstrates how to perform large-scale
queries on");
        Console.WriteLine("CloudWatch Logs using recursive binary search to retrieve
more than");
        Console.WriteLine("the 10,000 result limit.");
        Console.WriteLine();

        try
        {
            Console.WriteLine(new string('-', 80));
            var prepareSuccess = await PrepareApplication();
            Console.WriteLine(new string('-', 80));

            if (prepareSuccess)
            {
                Console.WriteLine(new string('-', 80));
                await ExecuteLargeQuery();
                Console.WriteLine(new string('-', 80));
            }

            Console.WriteLine(new string('-', 80));
            await Cleanup();
            Console.WriteLine(new string('-', 80));
        }
        catch (Exception ex)
        {
            _logger.LogError(ex, "There was a problem with the scenario, initiating
cleanup...");
            _interactive = false;
            await Cleanup();
        }

        Console.WriteLine("CloudWatch Logs Large Query scenario completed.");
    }

    /// <summary>
    /// Prepares the application by creating the necessary resources.
```

```
    /// </summary>
    /// <returns>True if the application was prepared successfully.</returns>
    public static async Task<bool> PrepareApplication()
    {
        Console.WriteLine("Preparing the application...");
        Console.WriteLine();

        try
        {
            var deployStack = !_interactive || GetYesNoResponse(
                "Would you like to deploy the CloudFormation stack and generate
sample logs? (y/n) ");

            if (deployStack)
            {
                if (_interactive)
                {
                    Console.Write(
                        $"Enter a path for the CloudFormation stack resource .yaml
file (or press Enter for default '{_stackResourcePath}'): ");
                    string? inputPath = Console.ReadLine();
                    if (!string.IsNullOrWhiteSpace(inputPath))
                    {
                        _stackResourcePath = inputPath;
                    }
                }

                _stackName = PromptUserForStackName();

                var deploySuccess = await DeployCloudFormationStack(_stackName);

                if (deploySuccess)
                {
                    Console.WriteLine();
                    Console.WriteLine("Generating 50,000 sample log entries...");
                    var generateSuccess = await GenerateSampleLogs();

                    if (generateSuccess)
                    {
                        Console.WriteLine();
                        Console.WriteLine("Sample logs created. Waiting 5 minutes
for logs to be fully ingested...");
                        await WaitWithCountdown(300);
```

```
                        Console.WriteLine("Application preparation complete.");
                        return true;
                    }
                }
            }
            else
            {
                _logGroupName = PromptUserForInput("Enter the log group name ",
_logGroupName);
                _logStreamName = PromptUserForInput("Enter the log stream name ",
_logStreamName);

                var startDateMs = PromptUserForLong("Enter the query start date
(milliseconds since epoch): ");
                var endDateMs = PromptUserForLong("Enter the query end date
(milliseconds since epoch): ");

                _queryStartDate = startDateMs / 1000;
                _queryEndDate = endDateMs / 1000;

                Console.WriteLine("Application preparation complete.");
                return true;
            }
        }
        catch (Exception ex)
        {
            _logger.LogError(ex, "An error occurred while preparing the
application.");
        }

        Console.WriteLine("Application preparation failed.");
        return false;
    }

    /// <summary>
    /// Deploys the CloudFormation stack with the necessary resources.
    /// </summary>
    /// <param name="stackName">The name of the CloudFormation stack.</param>
    /// <returns>True if the stack was deployed successfully.</returns>
    private static async Task<bool> DeployCloudFormationStack(string stackName)
    {
        Console.WriteLine($"\nDeploying CloudFormation stack: {stackName}");

        try
```

```
        {
            var request = new CreateStackRequest
            {
                StackName = stackName,
                TemplateBody = await File.ReadAllTextAsync(_stackResourcePath)
            };

            var response = await _amazonCloudFormation.CreateStackAsync(request);

            if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
            {
                Console.WriteLine($"CloudFormation stack creation started:
{stackName}");

                bool stackCreated = await WaitForStackCompletion(response.StackId);

                if (stackCreated)
                {
                    Console.WriteLine("CloudFormation stack created successfully.");
                    return true;
                }
                else
                {
                    _logger.LogError($"CloudFormation stack creation failed:
{stackName}");
                    return false;
                }
            }
            else
            {
                _logger.LogError($"Failed to create CloudFormation stack:
{stackName}");
                return false;
            }
        }
        catch (AlreadyExistsException)
        {
            _logger.LogWarning($"CloudFormation stack '{stackName}' already exists.
Please provide a unique name.");
            var newStackName = PromptUserForStackName();
            return await DeployCloudFormationStack(newStackName);
        }
        catch (Exception ex)
        {
```

```
            _logger.LogError(ex, $"An error occurred while deploying the
CloudFormation stack: {stackName}");
            return false;
        }
    }

    /// <summary>
    /// Waits for the CloudFormation stack to be in the CREATE_COMPLETE state.
    /// </summary>
    /// <param name="stackId">The ID of the CloudFormation stack.</param>
    /// <returns>True if the stack was created successfully.</returns>
    private static async Task<bool> WaitForStackCompletion(string stackId)
    {
        int retryCount = 0;
        const int maxRetries = 30;
        const int retryDelay = 10000;

        while (retryCount < maxRetries)
        {
            var describeStacksRequest = new DescribeStacksRequest
            {
                StackName = stackId
            };

            var describeStacksResponse = await
_amazonCloudFormation.DescribeStacksAsync(describeStacksRequest);

            if (describeStacksResponse.Stacks.Count > 0)
            {
                if (describeStacksResponse.Stacks[0].StackStatus ==
StackStatus.CREATE_COMPLETE)
                {
                    return true;
                }
                if (describeStacksResponse.Stacks[0].StackStatus ==
StackStatus.CREATE_FAILED ||
                    describeStacksResponse.Stacks[0].StackStatus ==
StackStatus.ROLLBACK_COMPLETE)
                {
                    return false;
                }
            }
```

```
            Console.WriteLine("Waiting for CloudFormation stack creation to
complete...");
            await Task.Delay(retryDelay);
            retryCount++;
        }

        _logger.LogError("Timed out waiting for CloudFormation stack creation to
complete.");
        return false;
    }

    /// <summary>
    /// Generates sample logs directly using CloudWatch Logs API.
    /// Creates 50,000 log entries spanning 5 minutes.
    /// </summary>
    /// <returns>True if logs were generated successfully.</returns>
    private static async Task<bool> GenerateSampleLogs()
    {
        const int totalEntries = 50000;
        const int entriesPerBatch = 10000;
        const int fiveMinutesMs = 5 * 60 * 1000;

        try
        {
            // Calculate timestamps
            var startTimeMs = DateTimeOffset.UtcNow.ToUnixTimeMilliseconds();
            var timestampIncrement = fiveMinutesMs / totalEntries;

            Console.WriteLine($"Generating {totalEntries} log entries...");

            var entryCount = 0;
            var currentTimestamp = startTimeMs;
            var numBatches = totalEntries / entriesPerBatch;

            // Generate and upload logs in batches
            for (int batchNum = 0; batchNum < numBatches; batchNum++)
            {
                var logEvents = new List<InputLogEvent>();

                for (int i = 0; i < entriesPerBatch; i++)
                {
                    logEvents.Add(new InputLogEvent
                    {
```

```
                        Timestamp =
 DateTimeOffset.FromUnixTimeMilliseconds(currentTimestamp).UtcDateTime,
                        Message = $"Entry {entryCount}"
                    });

                    entryCount++;
                    currentTimestamp += timestampIncrement;
                }

                // Upload batch
                var success = await _wrapper.PutLogEventsAsync(_logGroupName,
 _logStreamName, logEvents);
                if (!success)
                {
                    _logger.LogError($"Failed to upload batch {batchNum + 1}/
{numBatches}");
                    return false;
                }

                Console.WriteLine($"Uploaded batch {batchNum + 1}/{numBatches}");
            }

            // Set query date range (convert milliseconds to seconds for query API)
            _queryStartDate = startTimeMs / 1000;
            _queryEndDate = (currentTimestamp - timestampIncrement) / 1000;

            Console.WriteLine($"Query start date:
 {DateTimeOffset.FromUnixTimeSeconds(_queryStartDate):yyyy-MM-ddTHH:mm:ss.fffZ}");
            Console.WriteLine($"Query end date:
 {DateTimeOffset.FromUnixTimeSeconds(_queryEndDate):yyyy-MM-ddTHH:mm:ss.fffZ}");
            Console.WriteLine($"Successfully uploaded {totalEntries} log entries");

            return true;
        }
        catch (Exception ex)
        {
            _logger.LogError(ex, "An error occurred while generating sample logs.");
            return false;
        }
    }

    /// <summary>
    /// Executes the large query workflow.
    /// </summary>
```

```
    public static async Task ExecuteLargeQuery()
    {
        Console.WriteLine("Starting recursive query to retrieve all logs...");
        Console.WriteLine();

        var queryLimit = PromptUserForInteger("Enter the query limit (max 10000) ",
10000);
        if (queryLimit > 10000) queryLimit = 10000;

        var queryString = "fields @timestamp, @message | sort @timestamp asc";

        var stopwatch = Stopwatch.StartNew();
        var allResults = await PerformLargeQuery(_logGroupName, queryString,
_queryStartDate, _queryEndDate, queryLimit);
        stopwatch.Stop();

        Console.WriteLine();
        Console.WriteLine($"Queries finished in {stopwatch.Elapsed.TotalSeconds:F3}
seconds.");
        Console.WriteLine($"Total logs found: {allResults.Count}");

        // Check for duplicates
        Console.WriteLine();
        Console.WriteLine("Checking for duplicate logs...");
        var duplicates = FindDuplicateLogs(allResults);
        if (duplicates.Count > 0)
        {
            Console.WriteLine($"WARNING: Found {duplicates.Count} duplicate log
entries!");
            Console.WriteLine("Duplicate entries (showing first 10):");
            foreach (var dup in duplicates.Take(10))
            {
                Console.WriteLine($"  [{dup.Timestamp}] {dup.Message} (appears
{dup.Count} times)");
            }

            var uniqueCount = allResults.Count - duplicates.Sum(d => d.Count - 1);
            Console.WriteLine($"Unique logs: {uniqueCount}");
        }
        else
        {
            Console.WriteLine("No duplicates found. All logs are unique.");
        }
        Console.WriteLine();
```

```
        var viewSample = !_interactive || GetYesNoResponse("Would you like to see a
sample of the logs? (y/n) ");
        if (viewSample)
        {
            Console.WriteLine();
            Console.WriteLine($"Sample logs (first 10 of {allResults.Count}):");
            for (int i = 0; i < Math.Min(10, allResults.Count); i++)
            {
                var timestamp = allResults[i].Find(f => f.Field ==
"@timestamp")?.Value ?? "N/A";
                var message = allResults[i].Find(f => f.Field ==
"@message")?.Value ?? "N/A";
                Console.WriteLine($"[{timestamp}] {message}");
            }
        }
    }

    /// <summary>
    /// Performs a large query using recursive binary search.
    /// </summary>
    private static async Task<List<List<ResultField>>> PerformLargeQuery(
        string logGroupName,
        string queryString,
        long startTime,
        long endTime,
        int limit)
    {
        var queryId = await _wrapper.StartQueryAsync(logGroupName, queryString,
startTime, endTime, limit);
        if (queryId == null)
        {
            return new List<List<ResultField>>();
        }

        var results = await PollQueryResults(queryId);
        if (results == null || results.Count == 0)
        {
            return new List<List<ResultField>>();
        }

        var startDate =
DateTimeOffset.FromUnixTimeSeconds(startTime).ToString("yyyy-MM-ddTHH:mm:ss.fffZ");
```

```
        var endDate = DateTimeOffset.FromUnixTimeSeconds(endTime).ToString("yyyy-MM-
ddTHH:mm:ss.fffZ");
        Console.WriteLine($"Query date range: {startDate} ({startTime}s) to
 {endDate} ({endTime}s). Found {results.Count} logs.");

        if (results.Count < limit)
        {
            Console.WriteLine($"  -> Returning {results.Count} logs (less than limit
 of {limit})");
            return results;
        }

        Console.WriteLine($"  -> Hit limit of {limit}. Need to split and recurse.");

        // Get the timestamp of the last log (sorted to find the actual last one)
        var lastLogTimestamp = GetLastLogTimestamp(results);
        if (lastLogTimestamp == null)
        {
            Console.WriteLine($"  -> No timestamp found in results. Returning
 {results.Count} logs.");
            return results;
        }

        Console.WriteLine($"  -> Last log timestamp: {lastLogTimestamp}");

        // Parse the timestamp and add 1 millisecond to avoid querying the same log
 again
        var lastLogDate = DateTimeOffset.Parse(lastLogTimestamp + " +0000");
        Console.WriteLine($"  -> Last log as DateTimeOffset: {lastLogDate:yyyy-MM-
ddTHH:mm:ss.fffZ} ({lastLogDate.ToUnixTimeSeconds()}s)");

        var offsetLastLogDate = lastLogDate.AddMilliseconds(1);
        Console.WriteLine($"  -> Offset timestamp (last
 + 1ms): {offsetLastLogDate:yyyy-MM-ddTHH:mm:ss.fffZ}
 ({offsetLastLogDate.ToUnixTimeSeconds()}s)");

        // Convert to seconds, but round UP to the next second to avoid overlapping
 with logs in the same second
        // This ensures we don't re-query logs that share the same second as the
 last log
        var offsetLastLogTime = offsetLastLogDate.ToUnixTimeSeconds();
        if (offsetLastLogDate.Millisecond > 0)
        {
            offsetLastLogTime++; // Move to the next full second
```

```
            Console.WriteLine($"  -> Adjusted to next full second:
 {offsetLastLogTime}s ({DateTimeOffset.FromUnixTimeSeconds(offsetLastLogTime):yyyy-
MM-ddTHH:mm:ss.fffZ})");
        }

        Console.WriteLine($"  -> Comparing: offsetLastLogTime={offsetLastLogTime}s
 vs endTime={endTime}s");
        Console.WriteLine($"  -> End time as date:
 {DateTimeOffset.FromUnixTimeSeconds(endTime):yyyy-MM-ddTHH:mm:ss.fffZ}");

        // Check if there's any time range left to query
        if (offsetLastLogTime >= endTime)
        {
            Console.WriteLine($"  -> No time range left to query. Offset time
 ({offsetLastLogTime}s) >= end time ({endTime}s)");
            return results;
        }

        // Split the remaining date range in half
        var (range1Start, range1End, range2Start, range2End) =
 SplitDateRange(offsetLastLogTime, endTime);

        var range1StartDate =
 DateTimeOffset.FromUnixTimeSeconds(range1Start).ToString("yyyy-MM-
ddTHH:mm:ss.fffZ");
        var range1EndDate =
 DateTimeOffset.FromUnixTimeSeconds(range1End).ToString("yyyy-MM-ddTHH:mm:ss.fffZ");
        var range2StartDate =
 DateTimeOffset.FromUnixTimeSeconds(range2Start).ToString("yyyy-MM-
ddTHH:mm:ss.fffZ");
        var range2EndDate =
 DateTimeOffset.FromUnixTimeSeconds(range2End).ToString("yyyy-MM-ddTHH:mm:ss.fffZ");

        Console.WriteLine($"  -> Splitting remaining range:");
        Console.WriteLine($"     Range 1: {range1StartDate} ({range1Start}s) to
 {range1EndDate} ({range1End}s)");
        Console.WriteLine($"     Range 2: {range2StartDate} ({range2Start}s) to
 {range2EndDate} ({range2End}s)");

        // Query both halves recursively
        Console.WriteLine($"  -> Querying range 1...");
        var results1 = await PerformLargeQuery(logGroupName, queryString,
 range1Start, range1End, limit);
        Console.WriteLine($"  -> Range 1 returned {results1.Count} logs");
```

```
        Console.WriteLine($"  -> Querying range 2...");
        var results2 = await PerformLargeQuery(logGroupName, queryString,
range2Start, range2End, limit);
        Console.WriteLine($"  -> Range 2 returned {results2.Count} logs");

        // Combine all results
        var allResults = new List<List<ResultField>>(results);
        allResults.AddRange(results1);
        allResults.AddRange(results2);

        Console.WriteLine($"  -> Combined total: {allResults.Count} logs
({results.Count} + {results1.Count} + {results2.Count})");

        return allResults;
    }

    /// <summary>
    /// Gets the timestamp string of the most recent log from a list of logs.
    /// Sorts timestamps to find the actual last one.
    /// </summary>
    private static string? GetLastLogTimestamp(List<List<ResultField>> logs)
    {
        var timestamps = logs
            .Select(log => log.Find(f => f.Field == "@timestamp")?.Value)
            .Where(t => !string.IsNullOrEmpty(t))
            .OrderBy(t => t)
            .ToList();

        if (timestamps.Count == 0)
        {
            return null;
        }

        return timestamps[timestamps.Count - 1];
    }

    /// <summary>
    /// Splits a date range in half.
    /// Range 2 starts at midpoint + 1 second to avoid overlap.
    /// </summary>
    private static (long range1Start, long range1End, long range2Start, long
range2End) SplitDateRange(long startTime, long endTime)
    {
```

```
        var midpoint = startTime + (endTime - startTime) / 2;
        // Range 2 starts at midpoint + 1 to avoid querying the same second twice
        return (startTime, midpoint, midpoint + 1, endTime);
    }

    /// <summary>
    /// Polls for query results until complete.
    /// </summary>
    private static async Task<List<List<ResultField>>?> PollQueryResults(string
queryId)
    {
        int retryCount = 0;
        const int maxRetries = 60;
        const int retryDelay = 1000;

        while (retryCount < maxRetries)
        {
            var response = await _wrapper.GetQueryResultsAsync(queryId);
            if (response == null)
            {
                return null;
            }

            if (response.Status == QueryStatus.Complete)
            {
                return response.Results;
            }

            if (response.Status == QueryStatus.Failed ||
                response.Status == QueryStatus.Cancelled ||
                response.Status == QueryStatus.Timeout ||
                response.Status == QueryStatus.Unknown)
            {
                _logger.LogError($"Query failed with status: {response.Status}");
                return null;
            }

            await Task.Delay(retryDelay);
            retryCount++;
        }

        _logger.LogError("Timed out waiting for query results.");
        return null;
    }
```

```
   /// <summary>
   /// Cleans up the resources created during the scenario.
   /// </summary>
   public static async Task<bool> Cleanup()
   {
       var cleanup = !_interactive || GetYesNoResponse(
           "Do you want to delete the CloudFormation stack and all resources? (y/n)
");

       if (cleanup)
       {
           try
           {
               var stackDeleteSuccess = await DeleteCloudFormationStack(_stackName,
false);
               return stackDeleteSuccess;
           }
           catch (Exception ex)
           {
               _logger.LogError(ex, "An error occurred while cleaning up the
resources.");
               return false;
           }
       }

       Console.WriteLine($"Resources will remain. Stack name: {_stackName}, Log
group: {_logGroupName}");
       _logger.LogInformation("CloudWatch Logs Large Query scenario is complete.");
       return true;
   }

   /// <summary>
   /// Deletes the CloudFormation stack and waits for confirmation.
   /// </summary>
   private static async Task<bool> DeleteCloudFormationStack(string stackName, bool
forceDelete)
   {
       var request = new DeleteStackRequest
       {
           StackName = stackName,
       };

       if (forceDelete)
```

```
        {
            request.DeletionMode = DeletionMode.FORCE_DELETE_STACK;
        }

        await _amazonCloudFormation.DeleteStackAsync(request);
        Console.WriteLine($"CloudFormation stack '{stackName}' is being deleted.
This may take a few minutes.");

        bool stackDeleted = await WaitForStackDeletion(stackName, forceDelete);

        if (stackDeleted)
        {
            Console.WriteLine($"CloudFormation stack '{stackName}' has been
deleted.");
            return true;
        }
        else
        {
            _logger.LogError($"Failed to delete CloudFormation stack
'{stackName}'.");
            return false;
        }
    }

    /// <summary>
    /// Waits for the stack to be deleted.
    /// </summary>
    private static async Task<bool> WaitForStackDeletion(string stackName, bool
forceDelete)
    {
        int retryCount = 0;
        const int maxRetries = 30;
        const int retryDelay = 10000;

        while (retryCount < maxRetries)
        {
            var describeStacksRequest = new DescribeStacksRequest
            {
                StackName = stackName
            };

            try
            {
```

```
                var describeStacksResponse = await
_amazonCloudFormation.DescribeStacksAsync(describeStacksRequest);

                if (describeStacksResponse.Stacks.Count == 0 ||
                    describeStacksResponse.Stacks[0].StackStatus ==
StackStatus.DELETE_COMPLETE)
                {
                    return true;
                }

                if (!forceDelete && describeStacksResponse.Stacks[0].StackStatus ==
StackStatus.DELETE_FAILED)
                {
                    return await DeleteCloudFormationStack(stackName, true);
                }
            }
            catch (AmazonCloudFormationException ex) when (ex.ErrorCode ==
"ValidationError")
            {
                return true;
            }

            Console.WriteLine($"Waiting for CloudFormation stack '{stackName}' to be
deleted...");
            await Task.Delay(retryDelay);
            retryCount++;
        }

        _logger.LogError($"Timed out waiting for CloudFormation stack '{stackName}'
to be deleted.");
        return false;
    }

    /// <summary>
    /// Waits with a countdown display.
    /// </summary>
    private static async Task WaitWithCountdown(int seconds)
    {
        for (int i = seconds; i > 0; i--)
        {
            Console.Write($"\rWaiting: {i} seconds remaining...  ");
            await Task.Delay(1000);
        }
        Console.WriteLine("\rWait complete.                      ");
```

```
    }

    /// <summary>
    /// Helper method to get a yes or no response from the user.
    /// </summary>
    private static bool GetYesNoResponse(string question)
    {
        Console.WriteLine(question);
        var ynResponse = Console.ReadLine();
        var response = ynResponse != null && ynResponse.Equals("y",
StringComparison.InvariantCultureIgnoreCase);
        return response;
    }

    /// <summary>
    /// Prompts the user for a stack name.
    /// </summary>
    private static string PromptUserForStackName()
    {
        if (_interactive)
        {
            Console.Write($"Enter a name for the CloudFormation stack (press Enter
for default '{_stackName}'): ");
            string? input = Console.ReadLine();
            if (!string.IsNullOrWhiteSpace(input))
            {
                var regex = "[a-zA-Z][-a-zA-Z0-9]*";
                if (!Regex.IsMatch(input, regex))
                {
                    Console.WriteLine($"Invalid stack name. Using default:
{_stackName}");
                    return _stackName;
                }
                return input;
            }
        }
        return _stackName;
    }

    /// <summary>
    /// Prompts the user for input with a default value.
    /// </summary>
    private static string PromptUserForInput(string prompt, string defaultValue)
    {
```

```
        if (_interactive)
        {
            Console.Write($"{prompt}(press Enter for default '{defaultValue}'): ");
            string? input = Console.ReadLine();
            return string.IsNullOrWhiteSpace(input) ? defaultValue : input;
        }
        return defaultValue;
    }

    /// <summary>
    /// Prompts the user for an integer value.
    /// </summary>
    private static int PromptUserForInteger(string prompt, int defaultValue)
    {
        if (_interactive)
        {
            Console.Write($"{prompt}(press Enter for default '{defaultValue}'): ");
            string? input = Console.ReadLine();
            if (string.IsNullOrWhiteSpace(input) || !int.TryParse(input, out var
result))
            {
                return defaultValue;
            }
            return result;
        }
        return defaultValue;
    }

    /// <summary>
    /// Prompts the user for a long value.
    /// </summary>
    private static long PromptUserForLong(string prompt)
    {
        if (_interactive)
        {
            Console.Write(prompt);
            string? input = Console.ReadLine();
            if (long.TryParse(input, out var result))
            {
                return result;
            }
        }
        return 0;
    }
```

```
    /// <summary>
    /// Finds duplicate log entries based on timestamp and message.
    /// </summary>
    private static List<(string Timestamp, string Message, int Count)>
 FindDuplicateLogs(List<List<ResultField>> logs)
    {
        var logSignatures = new Dictionary<string, int>();

        foreach (var log in logs)
        {
            var timestamp = log.Find(f => f.Field == "@timestamp")?.Value ?? "";
            var message = log.Find(f => f.Field == "@message")?.Value ?? "";
            var signature = $"{timestamp}|{message}";

            if (logSignatures.ContainsKey(signature))
            {
                logSignatures[signature]++;
            }
            else
            {
                logSignatures[signature] = 1;
            }
        }

        return logSignatures
            .Where(kvp => kvp.Value > 1)
            .Select(kvp =>
            {
                var parts = kvp.Key.Split('|');
                return (Timestamp: parts[0], Message: parts[1], Count: kvp.Value);
            })
            .OrderByDescending(x => x.Count)
            .ToList();
    }
}
```

- API 세부 정보는 AWS SDK for .NET API 참조의 다음 주제를 참조하세요.
  - GetQueryResults
  - StartQuery

SDK for .NET (v4)를 사용한 Amazon Cognito 자격 증명 공급자 예제

다음 코드 예제에서는 Amazon Cognito 자격 증명 공급자와 함께 AWS SDK for .NET (v4)를 사용하여 작업을 수행하고 일반적인 시나리오를 구현하는 방법을 보여줍니다.

작업은 대규모 프로그램에서 발췌한 코드이며 컨텍스트에 맞춰 실행해야 합니다. 작업은 개별 서비스 함수를 직접적으로 호출하는 방법을 보여주며 관련 시나리오의 컨텍스트에 맞는 작업을 볼 수 있습니다.

각 예시에는 전체 소스 코드에 대한 링크가 포함되어 있으며, 여기에서 컨텍스트에 맞춰 코드를 설정하고 실행하는 방법에 대한 지침을 찾을 수 있습니다.

주제

- 작업

작업

`ListUserPools`

다음 코드 예시는 `ListUserPools`의 사용 방법을 보여 줍니다.

SDK for .NET (v4)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

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

- API 세부 정보는 AWS SDK for .NET API 참조의 ListUserPools를 참조하세요.

AWS Control Tower SDK for .NET (v4)를 사용한 예제

다음 코드 예제에서는 AWS SDK for .NET (v4)를와 함께 사용하여 작업을 수행하고 일반적인 시나리오를 구현하는 방법을 보여줍니다 AWS Control Tower.

기본 사항은 서비스 내에서 필수 작업을 수행하는 방법을 보여주는 코드 예제입니다.

작업은 대규모 프로그램에서 발췌한 코드이며 컨텍스트에 맞춰 실행해야 합니다. 작업은 개별 서비스 함수를 직접적으로 호출하는 방법을 보여주며 관련 시나리오의 컨텍스트에 맞는 작업을 볼 수 있습니다.

각 예시에는 전체 소스 코드에 대한 링크가 포함되어 있으며, 여기에서 컨텍스트에 맞춰 코드를 설정하고 실행하는 방법에 대한 지침을 찾을 수 있습니다.

주제

- 시작하기
- 기본 사항
- 작업

시작하기

안녕하세요 AWS Control Tower

다음 코드 예제에서는 AWS Control Tower를 사용하여 시작하는 방법을 보여 줍니다.

SDK for .NET (v4)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

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

- API 세부 정보는 AWS SDK for .NET API 참조의 ListBaselines을 참조하세요.

기본 사항

기본 사항 알아보기

다음 코드 예제에서는 다음과 같은 작업을 수행하는 방법을 보여줍니다.

- 랜딩 존을 나열합니다.
- 기준을 나열, 활성화, 조회, 재설정, 비활성화합니다.
- 제어 기능을 나열, 활성화, 조회, 비활성화합니다.

SDK for .NET (v4)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

AWS Control Tower 기능을 보여주는 대화형 시나리오를 실행합니다.

```
using Amazon.ControlCatalog;
using Amazon.ControlTower;
using Amazon.ControlTower.Model;
using Amazon.Organizations;
using Amazon.Organizations.Model;
using Amazon.SecurityToken;
using Amazon.SecurityToken.Model;
using ControlTowerActions;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;
using Microsoft.Extensions.Logging;

namespace ControlTowerBasics;

/// <summary>
/// Scenario class for AWS Control Tower basics.
/// </summary>
public class ControlTowerBasics
{
    public static bool isInteractive = true;
```

```
    public static ILogger logger = null!;
    public static IAmazonOrganizations? orgClient = null;
    public static IAmazonSecurityTokenService? stsClient = null;
    public static ControlTowerWrapper? wrapper = null;
    private static string? ouArn;
    private static bool useLandingZone = false;

    /// <summary>
    /// Main entry point for the AWS Control Tower basics scenario.
    /// </summary>
    /// <param name="args">Command line arguments.</param>
    public static async Task Main(string[] args)
    {
        using var host = Host.CreateDefaultBuilder(args)
            .ConfigureServices((_, services) =>
                services.AddAWSService<IAmazonControlTower>()
                .AddAWSService<IAmazonControlCatalog>()
                .AddAWSService<IAmazonOrganizations>()
                .AddAWSService<IAmazonSecurityTokenService>()
                .AddTransient<ControlTowerWrapper>()
            )
            .Build();

        logger = LoggerFactory.Create(builder => { builder.AddConsole(); })
            .CreateLogger<ControlTowerBasics>();

        wrapper = host.Services.GetRequiredService<ControlTowerWrapper>();
        orgClient = host.Services.GetRequiredService<IAmazonOrganizations>();
        stsClient = host.Services.GetRequiredService<IAmazonSecurityTokenService>();

        await RunScenario();
    }

    /// <summary>
    /// Runs the example scenario.
    /// </summary>
    public static async Task RunScenario()
    {
        Console.WriteLine(new string('-', 88));
        Console.WriteLine("\tWelcome to the AWS Control Tower with ControlCatalog
example scenario.");
        Console.WriteLine(new string('-', 88));
        Console.WriteLine("This demo will walk you through working with AWS Control
Tower for landing zones,");
```

```
        Console.WriteLine("managing baselines, and working with controls.");

        try
        {
            var accountId = (await stsClient!.GetCallerIdentityAsync(new
 GetCallerIdentityRequest())).Account;
            Console.WriteLine($"\nAccount ID: {accountId}");

            Console.WriteLine("\nSome demo operations require the use of a landing
 zone.");
            Console.WriteLine("You can use an existing landing zone or opt out of
 these operations in the demo.");
            Console.WriteLine("For instructions on how to set up a landing zone,");
            Console.WriteLine("see https://docs.aws.amazon.com/controltower/latest/
userguide/getting-started-from-console.html");

            // List available landing zones
            var landingZones = await wrapper!.ListLandingZonesAsync();
            if (landingZones.Count > 0)
            {
                Console.WriteLine("\nAvailable Landing Zones:");
                for (int i = 0; i < landingZones.Count; i++)
                {
                    Console.WriteLine($"{i + 1}. {landingZones[i].Arn}");
                }

                Console.Write($"\nDo you want to use the first landing zone in the
 list ({landingZones[0].Arn})? (y/n): ");
                if (GetUserConfirmation())
                {
                    useLandingZone = true;
                    Console.WriteLine($"Using landing zone: {landingZones[0].Arn}");
                    ouArn = await SetupOrganizationAsync();
                }
            }

            // Managing Baselines
            Console.WriteLine("\nManaging Baselines:");
            var baselines = await wrapper.ListBaselinesAsync();
            Console.WriteLine("\nListing available Baselines:");
            BaselineSummary? controlTowerBaseline = null;
            foreach (var baseline in baselines)
            {
                if (baseline.Name == "AWSControlTowerBaseline")
```

```
                    controlTowerBaseline = baseline;
                Console.WriteLine($"  - {baseline.Name}");
            }

            EnabledBaselineSummary? identityCenterBaseline = null;
            string? baselineArn = null;

            if (useLandingZone && ouArn != null)
            {
                Console.WriteLine("\nListing enabled baselines:");
                var enabledBaselines = await wrapper.ListEnabledBaselinesAsync();
                foreach (var baseline in enabledBaselines)
                {
                    if (baseline.BaselineIdentifier.Contains("baseline/
LN25R72TTG6IGPTQ"))
                        identityCenterBaseline = baseline;
                    Console.WriteLine($"  - {baseline.BaselineIdentifier}");
                }

                if (controlTowerBaseline != null)
                {
                    Console.Write("\nDo you want to enable the Control Tower
 Baseline? (y/n): ");
                    if (GetUserConfirmation())
                    {
                        Console.WriteLine("\nEnabling Control Tower Baseline.");
                        var icBaselineArn = identityCenterBaseline?.Arn;
                        baselineArn = await wrapper.EnableBaselineAsync(ouArn,
                            controlTowerBaseline.Arn, "5.0", icBaselineArn ?? "");
                        var alreadyEnabled = false;
                        if (baselineArn != null)
                        {
                            Console.WriteLine($"Enabled baseline ARN:
 {baselineArn}");
                        }
                        else
                        {
                            // Find the enabled baseline
                            foreach (var enabled in enabledBaselines)
                            {
                                if (enabled.BaselineIdentifier ==
 controlTowerBaseline.Arn)
                                {
                                    baselineArn = enabled.Arn;
```

```
                                    alreadyEnabled = true;
                                    Console.WriteLine("No change, the selected
baseline was already enabled.");
                                    break;
                                }
                            }
                        }

                        if (baselineArn != null)
                        {
                            Console.Write("\nDo you want to reset the Control Tower
Baseline? (y/n): ");
                            if (GetUserConfirmation())
                            {
                                Console.WriteLine($"\nResetting Control Tower
Baseline: {baselineArn}");
                                var operationId = await
wrapper.ResetEnabledBaselineAsync(baselineArn);
                                Console.WriteLine($"Reset baseline operation id:
{operationId}");
                            }

                            Console.Write("\nDo you want to disable the Control
Tower Baseline? (y/n): ");
                            if (GetUserConfirmation())
                            {
                                Console.WriteLine($"Disabling baseline ARN:
{baselineArn}");
                                var operationId = await
wrapper.DisableBaselineAsync(baselineArn);
                                Console.WriteLine($"Disabled baseline operation id:
{operationId}");
                                if (alreadyEnabled)
                                {
                                    Console.WriteLine($"\nRe-enabling Control Tower
Baseline: {baselineArn}");
                                    // Re-enable the Control Tower baseline if it
was originally enabled.
                                    await wrapper.EnableBaselineAsync(ouArn,
                                        controlTowerBaseline.Arn, "5.0",
icBaselineArn ?? "");
                                }
                            }
                        }
```

```
                    }
                }
            }

            // Managing Controls
            Console.WriteLine("\nManaging Controls:");
            var controls = await wrapper.ListControlsAsync();
            Console.WriteLine("\nListing first 5 available Controls:");
            for (int i = 0; i < Math.Min(5, controls.Count); i++)
            {
                Console.WriteLine($"{i + 1}. {controls[i].Name} -
 {controls[i].Arn}");
            }

            if (useLandingZone && ouArn != null)
            {
                var enabledControls = await wrapper.ListEnabledControlsAsync(ouArn);
                Console.WriteLine("\nListing enabled controls:");
                for (int i = 0; i < enabledControls.Count; i++)
                {
                    Console.WriteLine($"{i + 1}.
 {enabledControls[i].ControlIdentifier}");
                }

                // Find first non-enabled control
                var enabledControlArns = enabledControls.Select(c =>
 c.Arn).ToHashSet();
                var controlArn = controls.FirstOrDefault(c => !
enabledControlArns.Contains(c.Arn))?.Arn;

                if (controlArn != null)
                {
                    Console.Write($"\nDo you want to enable the control
 {controlArn}? (y/n): ");
                    if (GetUserConfirmation())
                    {
                        Console.WriteLine($"\nEnabling control: {controlArn}");
                        var operationId = await
 wrapper.EnableControlAsync(controlArn, ouArn);
                        if (operationId != null)
                        {
                            Console.WriteLine($"Enabled control with operation id:
 {operationId}");
```

```
                            Console.Write("\nDo you want to disable the control? (y/
n): ");
                            if (GetUserConfirmation())
                            {
                                Console.WriteLine("\nDisabling the control...");
                                var disableOpId = await
 wrapper.DisableControlAsync(controlArn, ouArn);
                                Console.WriteLine($"Disable operation ID:
 {disableOpId}");
                            }
                        }
                    }
                }
            }

            Console.WriteLine("\nThis concludes the example scenario.");
            Console.WriteLine("Thanks for watching!");
            Console.WriteLine(new string('-', 88));
        }
        catch (Exception ex)
        {
            logger.LogError(ex, "An error occurred during the Control Tower
 scenario.");
            Console.WriteLine($"An error occurred: {ex.Message}");
        }
    }

    /// <summary>
    /// Sets up AWS Organizations and creates or finds a Sandbox OU.
    /// </summary>
    /// <returns>The ARN of the Sandbox organizational unit.</returns>
    private static async Task<string> SetupOrganizationAsync()
    {
        Console.WriteLine("\nChecking organization status...");

        try
        {
            var orgResponse = await orgClient!.DescribeOrganizationAsync(new
 DescribeOrganizationRequest());
            var orgId = orgResponse.Organization.Id;
            Console.WriteLine($"Account is part of organization: {orgId}");
        }
        catch (AWSOrganizationsNotInUseException)
        {
```

```
            Console.WriteLine("No organization found. Creating a new
organization...");
            var createResponse = await orgClient!.CreateOrganizationAsync(new
CreateOrganizationRequest { FeatureSet = OrganizationFeatureSet.ALL });
            var orgId = createResponse.Organization.Id;
            Console.WriteLine($"Created new organization: {orgId}");
        }

        // Look for Sandbox OU
        var roots = await orgClient.ListRootsAsync(new ListRootsRequest());
        var rootId = roots.Roots[0].Id;

        Console.WriteLine("Checking for Sandbox OU...");
        var ous = await orgClient.ListOrganizationalUnitsForParentAsync(new
ListOrganizationalUnitsForParentRequest { ParentId = rootId });
        var sandboxOu = ous.OrganizationalUnits.FirstOrDefault(ou => ou.Name ==
"Sandbox");

        if (sandboxOu == null)
        {
            Console.WriteLine("Creating Sandbox OU...");
            var createOuResponse = await orgClient.CreateOrganizationalUnitAsync(new
CreateOrganizationalUnitRequest { ParentId = rootId, Name = "Sandbox" });
            sandboxOu = createOuResponse.OrganizationalUnit;
            Console.WriteLine($"Created new Sandbox OU: {sandboxOu.Id}");
        }
        else
        {
            Console.WriteLine($"Found existing Sandbox OU: {sandboxOu.Id}");
        }

        return sandboxOu.Arn;
    }

    /// <summary>
    /// Gets user confirmation by waiting for input or returning true if not
interactive.
    /// </summary>
    /// <returns>True if user enters 'y' or if isInteractive is false, otherwise
false.</returns>
    private static bool GetUserConfirmation()
    {
        return Console.ReadLine()?.ToLower() == "y" || !isInteractive;
    }
```

```
}
```

Aurora 작업을 관리하기 위해 시나리오가 직접적으로 호출하는 래퍼 메서드.

```
using Amazon.ControlCatalog;
using Amazon.ControlCatalog.Model;
using Amazon.ControlTower;
using Amazon.ControlTower.Model;
using ValidationException = Amazon.ControlTower.Model.ValidationException;

namespace ControlTowerActions;

/// <summary>
/// Methods to perform AWS Control Tower actions.
/// </summary>
public class ControlTowerWrapper
{
    private readonly IAmazonControlTower _controlTowerService;
    private readonly IAmazonControlCatalog _controlCatalogService;

    /// <summary>
    /// Constructor for the wrapper class containing AWS Control Tower actions.
    /// </summary>
    /// <param name="controlTowerService">The AWS Control Tower client object.</
param>
    /// <param name="controlCatalogService">The AWS Control Catalog client object.</
param>
    public ControlTowerWrapper(IAmazonControlTower controlTowerService,
 IAmazonControlCatalog controlCatalogService)
    {
        _controlTowerService = controlTowerService;
        _controlCatalogService = controlCatalogService;
    }

    /// <summary>
    /// List the AWS Control Tower landing zones for an account.
    /// </summary>
    /// <returns>A list of LandingZoneSummary objects.</returns>
    public async Task<List<LandingZoneSummary>> ListLandingZonesAsync()
    {
```

```
        try
        {
            var landingZones = new List<LandingZoneSummary>();

            var landingZonesPaginator =
_controlTowerService.Paginators.ListLandingZones(new ListLandingZonesRequest());

            await foreach (var response in landingZonesPaginator.Responses)
            {
                landingZones.AddRange(response.LandingZones);
            }

            return landingZones;
        }
        catch (AmazonControlTowerException ex)
        {
            Console.WriteLine($"Couldn't list landing zones. Here's why:
{ex.ErrorCode}: {ex.Message}");
            throw;
        }
    }


    /// <summary>
    /// List all baselines.
    /// </summary>
    /// <returns>A list of baseline summaries.</returns>
    public async Task<List<BaselineSummary>> ListBaselinesAsync()
    {
        try
        {
            var baselines = new List<BaselineSummary>();

            var baselinesPaginator =
_controlTowerService.Paginators.ListBaselines(new ListBaselinesRequest());

            await foreach (var response in baselinesPaginator.Responses)
            {
                baselines.AddRange(response.Baselines);
            }

            return baselines;
        }
        catch (AmazonControlTowerException ex)
```

```
        {
            Console.WriteLine($"Couldn't list baselines. Here's why: {ex.ErrorCode}:
 {ex.Message}");
            throw;
        }
    }


    /// <summary>
    /// List all enabled baselines.
    /// </summary>
    /// <returns>A list of enabled baseline summaries.</returns>
    public async Task<List<EnabledBaselineSummary>> ListEnabledBaselinesAsync()
    {
        try
        {
            var enabledBaselines = new List<EnabledBaselineSummary>();

            var enabledBaselinesPaginator =
 _controlTowerService.Paginators.ListEnabledBaselines(new
 ListEnabledBaselinesRequest());

            await foreach (var response in enabledBaselinesPaginator.Responses)
            {
                enabledBaselines.AddRange(response.EnabledBaselines);
            }

            return enabledBaselines;
        }
        catch (AmazonControlTowerException ex)
        {
            Console.WriteLine($"Couldn't list enabled baselines. Here's why:
 {ex.ErrorCode}: {ex.Message}");
            throw;
        }
    }


    /// <summary>
    /// Enable a baseline for the specified target.
    /// </summary>
    /// <param name="targetIdentifier">The ARN of the target.</param>
    /// <param name="baselineIdentifier">The identifier of baseline to enable.</
param>
```

```
    /// <param name="baselineVersion">The version of baseline to enable.</param>
    /// <param name="identityCenterBaseline">The identifier of identity center
baseline if it is enabled.</param>
    /// <returns>The enabled baseline ARN or null.</returns>
    public async Task<string?> EnableBaselineAsync(string targetIdentifier, string
baselineIdentifier, string baselineVersion, string identityCenterBaseline)
    {
        try
        {
            var parameters = new List<EnabledBaselineParameter>();
            if (!string.IsNullOrEmpty(identityCenterBaseline))
            {
                parameters.Add(
                    new EnabledBaselineParameter
                    {
                        Key = "IdentityCenterEnabledBaselineArn",
                        Value = identityCenterBaseline
                    });
            }
            var request = new EnableBaselineRequest
            {
                BaselineIdentifier = baselineIdentifier,
                BaselineVersion = baselineVersion,
                TargetIdentifier = targetIdentifier,
                Parameters = parameters
            };

            var response = await _controlTowerService.EnableBaselineAsync(request);
            var operationId = response.OperationIdentifier;

            // Wait for operation to complete
            while (true)
            {
                var status = await GetBaselineOperationAsync(operationId);
                Console.WriteLine($"Baseline operation status: {status}");
                if (status == BaselineOperationStatus.SUCCEEDED || status ==
BaselineOperationStatus.FAILED)
                {
                    break;
                }
                await Task.Delay(30000); // Wait 30 seconds
            }

            return response.Arn;
```

```
        }
        catch (ValidationException ex)
        {
            if (ex.Message.Contains("already enabled"))
                Console.WriteLine("Baseline is already enabled for this target");
            else { Console.WriteLine(ex.Message); }
            // Write the message and return null if baseline cannot be enabled.
            return null;
        }
        catch (AmazonControlTowerException ex)
        {
            Console.WriteLine($"Couldn't enable baseline. Here's why:
{ex.ErrorCode}: {ex.Message}");
            throw;
        }
    }


    /// <summary>
    /// Disable a baseline for a specific target and wait for the operation to
complete.
    /// </summary>
    /// <param name="enabledBaselineIdentifier">The identifier of the baseline to
disable.</param>
    /// <returns>The operation ID or null if there was a conflict.</returns>
    public async Task<string?> DisableBaselineAsync(string
enabledBaselineIdentifier)
    {
        try
        {
            var request = new DisableBaselineRequest
            {
                EnabledBaselineIdentifier = enabledBaselineIdentifier
            };

            var response = await _controlTowerService.DisableBaselineAsync(request);
            var operationId = response.OperationIdentifier;

            // Wait for operation to complete
            while (true)
            {
                var status = await GetBaselineOperationAsync(operationId);
                Console.WriteLine($"Baseline operation status: {status}");
```

```
                if (status == BaselineOperationStatus.SUCCEEDED || status ==
BaselineOperationStatus.FAILED)
                {
                    break;
                }
                await Task.Delay(30000); // Wait 30 seconds
            }

            return operationId;
        }
        catch (ConflictException ex)
        {
            Console.WriteLine($"Conflict disabling baseline: {ex.Message}. Skipping
disable step.");
            return null;
        }
        catch (AmazonControlTowerException ex)
        {
            Console.WriteLine($"Couldn't disable baseline. Here's why:
{ex.ErrorCode}: {ex.Message}");
            throw;
        }
    }


    /// <summary>
    /// Reset an enabled baseline for a specific target.
    /// </summary>
    /// <param name="enabledBaselineIdentifier">The identifier of the enabled
baseline to reset.</param>
    /// <returns>The operation ID.</returns>
    public async Task<string> ResetEnabledBaselineAsync(string
enabledBaselineIdentifier)
    {
        try
        {
            var request = new ResetEnabledBaselineRequest
            {
                EnabledBaselineIdentifier = enabledBaselineIdentifier
            };

            var response = await
_controlTowerService.ResetEnabledBaselineAsync(request);
            var operationId = response.OperationIdentifier;
```

```
            // Wait for operation to complete
            while (true)
            {
                var status = await GetBaselineOperationAsync(operationId);
                Console.WriteLine($"Baseline operation status: {status}");
                if (status == BaselineOperationStatus.SUCCEEDED || status ==
BaselineOperationStatus.FAILED)
                {
                    break;
                }
                await Task.Delay(30000); // Wait 30 seconds
            }

            return operationId;
        }
        catch (Amazon.ControlTower.Model.ResourceNotFoundException)
        {
            Console.WriteLine("Target not found, unable to reset enabled
baseline.");
            throw;
        }
        catch (AmazonControlTowerException ex)
        {
            Console.WriteLine($"Couldn't reset enabled baseline. Here's why:
{ex.ErrorCode}: {ex.Message}");
            throw;
        }
    }


    /// <summary>
    /// Get the status of a baseline operation.
    /// </summary>
    /// <param name="operationId">The ID of the baseline operation.</param>
    /// <returns>The operation status.</returns>
    public async Task<BaselineOperationStatus> GetBaselineOperationAsync(string
operationId)
    {
        try
        {
            var request = new GetBaselineOperationRequest
            {
                OperationIdentifier = operationId
```

```
            };

            var response = await
 _controlTowerService.GetBaselineOperationAsync(request);
            return response.BaselineOperation.Status;
        }
        catch (Amazon.ControlTower.Model.ResourceNotFoundException)
        {
            Console.WriteLine("Operation not found.");
            throw;
        }
        catch (AmazonControlTowerException ex)
        {
            Console.WriteLine($"Couldn't get baseline operation status. Here's why:
 {ex.ErrorCode}: {ex.Message}");
            throw;
        }
    }


    /// <summary>
    /// List enabled controls for a target organizational unit.
    /// </summary>
    /// <param name="targetIdentifier">The target organizational unit identifier.</
param>
    /// <returns>A list of enabled control summaries.</returns>
    public async Task<List<EnabledControlSummary>> ListEnabledControlsAsync(string
 targetIdentifier)
    {
        try
        {
            var request = new ListEnabledControlsRequest
            {
                TargetIdentifier = targetIdentifier
            };

            var enabledControls = new List<EnabledControlSummary>();

            var enabledControlsPaginator =
 _controlTowerService.Paginators.ListEnabledControls(request);

            await foreach (var response in enabledControlsPaginator.Responses)
            {
                enabledControls.AddRange(response.EnabledControls);
```

```
            }

            return enabledControls;
        }
        catch (Amazon.ControlTower.Model.ResourceNotFoundException ex) when
(ex.Message.Contains("not registered with AWS Control Tower"))
        {
            Console.WriteLine("AWS Control Tower must be enabled to work with
enabling controls.");
            return new List<EnabledControlSummary>();
        }
        catch (AmazonControlTowerException ex)
        {
            Console.WriteLine($"Couldn't list enabled controls. Here's why:
{ex.ErrorCode}: {ex.Message}");
            throw;
        }
    }


    /// <summary>
    /// Enable a control for a specified target.
    /// </summary>
    /// <param name="controlArn">The ARN of the control to enable.</param>
    /// <param name="targetIdentifier">The identifier of the target (e.g., OU
ARN).</param>
    /// <returns>The operation ID or null if already enabled.</returns>
    public async Task<string?> EnableControlAsync(string controlArn, string
targetIdentifier)
    {
        try
        {
            Console.WriteLine(controlArn);
            Console.WriteLine(targetIdentifier);

            var request = new EnableControlRequest
            {
                ControlIdentifier = controlArn,
                TargetIdentifier = targetIdentifier
            };

            var response = await _controlTowerService.EnableControlAsync(request);
            var operationId = response.OperationIdentifier;
```

```
            // Wait for operation to complete
            while (true)
            {
                var status = await GetControlOperationAsync(operationId);
                Console.WriteLine($"Control operation status: {status}");
                if (status == ControlOperationStatus.SUCCEEDED || status ==
ControlOperationStatus.FAILED)
                {
                    break;
                }
                await Task.Delay(30000); // Wait 30 seconds
            }

            return operationId;
        }
        catch (Amazon.ControlTower.Model.ValidationException ex) when
(ex.Message.Contains("already enabled"))
        {
            Console.WriteLine("Control is already enabled for this target");
            return null;
        }
        catch (Amazon.ControlTower.Model.ResourceNotFoundException ex) when
(ex.Message.Contains("not registered with AWS Control Tower"))
        {
            Console.WriteLine("AWS Control Tower must be enabled to work with
enabling controls.");
            return null;
        }
        catch (AmazonControlTowerException ex)
        {
            Console.WriteLine($"Couldn't enable control. Here's why: {ex.ErrorCode}:
{ex.Message}");
            throw;
        }
    }


    /// <summary>
    /// Disable a control for a specified target.
    /// </summary>
    /// <param name="controlArn">The ARN of the control to disable.</param>
    /// <param name="targetIdentifier">The identifier of the target (e.g., OU
ARN).</param>
    /// <returns>The operation ID.</returns>
```

```
   public async Task<string> DisableControlAsync(string controlArn, string
targetIdentifier)
   {
       try
       {
           var request = new DisableControlRequest
           {
               ControlIdentifier = controlArn,
               TargetIdentifier = targetIdentifier
           };

           var response = await _controlTowerService.DisableControlAsync(request);
           var operationId = response.OperationIdentifier;

           // Wait for operation to complete
           while (true)
           {
               var status = await GetControlOperationAsync(operationId);
               Console.WriteLine($"Control operation status: {status}");
               if (status == ControlOperationStatus.SUCCEEDED || status ==
ControlOperationStatus.FAILED)
               {
                   break;
               }
               await Task.Delay(30000); // Wait 30 seconds
           }

           return operationId;
       }
       catch (Amazon.ControlTower.Model.ResourceNotFoundException)
       {
           Console.WriteLine("Control not found.");
           throw;
       }
       catch (AmazonControlTowerException ex)
       {
           Console.WriteLine($"Couldn't disable control. Here's why:
{ex.ErrorCode}: {ex.Message}");
           throw;
       }
   }


   /// <summary>
```

```
    /// Get the status of a control operation.
    /// </summary>
    /// <param name="operationId">The ID of the control operation.</param>
    /// <returns>The operation status.</returns>
    public async Task<ControlOperationStatus> GetControlOperationAsync(string
operationId)
    {
        try
        {
            var request = new GetControlOperationRequest
            {
                OperationIdentifier = operationId
            };

            var response = await
_controlTowerService.GetControlOperationAsync(request);
            return response.ControlOperation.Status;
        }
        catch (Amazon.ControlTower.Model.ResourceNotFoundException)
        {
            Console.WriteLine("Operation not found.");
            throw;
        }
        catch (AmazonControlTowerException ex)
        {
            Console.WriteLine($"Couldn't get control operation status. Here's why:
{ex.ErrorCode}: {ex.Message}");
            throw;
        }
    }


    /// <summary>
    /// List all controls in the Control Tower control catalog.
    /// </summary>
    /// <returns>A list of control summaries.</returns>
    public async Task<List<ControlSummary>> ListControlsAsync()
    {
        try
        {
            var controls = new List<ControlSummary>();
```

```
            var controlsPaginator =
_controlCatalogService.Paginators.ListControls(new
Amazon.ControlCatalog.Model.ListControlsRequest());

            await foreach (var response in controlsPaginator.Responses)
            {
                controls.AddRange(response.Controls);
            }

            return controls;
        }
        catch (AmazonControlCatalogException ex)
        {
            Console.WriteLine($"Couldn't list controls. Here's why: {ex.ErrorCode}:
{ex.Message}");
            throw;
        }
    }

}
```

- API 세부 정보는 AWS SDK for .NET API 참조의 다음 주제를 참조하세요.
  - CreateLandingZone
  - DeleteLandingZone
  - DisableBaseline
  - DisableControl
  - EnableBaseline
  - EnableControl
  - GetControlOperation
  - GetLandingZoneOperation
  - ListBaselines
  - ListEnabledBaselines
  - ListEnabledControls
  - ListLandingZones
  - ResetEnabledBaseline

작업

`DisableBaseline`

다음 코드 예시는 `DisableBaseline`의 사용 방법을 보여 줍니다.

SDK for .NET (v4)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

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

- API 세부 정보는 AWS SDK for .NET API 참조의 DisableBaseline을 참조하세요.

`DisableControl`

다음 코드 예시는 `DisableControl`의 사용 방법을 보여 줍니다.

SDK for .NET (v4)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

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

- API 세부 정보는 AWS SDK for .NET API 참조의 DisableControl을 참조하세요.

`EnableBaseline`

다음 코드 예시는 `EnableBaseline`의 사용 방법을 보여 줍니다.

SDK for .NET (v4)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
    /// <summary>
    /// Enable a baseline for the specified target.
    /// </summary>
    /// <param name="targetIdentifier">The ARN of the target.</param>
    /// <param name="baselineIdentifier">The identifier of baseline to enable.</
param>
    /// <param name="baselineVersion">The version of baseline to enable.</param>
    /// <param name="identityCenterBaseline">The identifier of identity center
 baseline if it is enabled.</param>
    /// <returns>The enabled baseline ARN or null.</returns>
    public async Task<string?> EnableBaselineAsync(string targetIdentifier, string
 baselineIdentifier, string baselineVersion, string identityCenterBaseline)
    {
        try
        {
            var parameters = new List<EnabledBaselineParameter>();
            if (!string.IsNullOrEmpty(identityCenterBaseline))
            {
                parameters.Add(
                    new EnabledBaselineParameter
                    {
                        Key = "IdentityCenterEnabledBaselineArn",
                        Value = identityCenterBaseline
                    });
            }
```

```
            var request = new EnableBaselineRequest
            {
                BaselineIdentifier = baselineIdentifier,
                BaselineVersion = baselineVersion,
                TargetIdentifier = targetIdentifier,
                Parameters = parameters
            };

            var response = await _controlTowerService.EnableBaselineAsync(request);
            var operationId = response.OperationIdentifier;

            // Wait for operation to complete
            while (true)
            {
                var status = await GetBaselineOperationAsync(operationId);
                Console.WriteLine($"Baseline operation status: {status}");
                if (status == BaselineOperationStatus.SUCCEEDED || status ==
BaselineOperationStatus.FAILED)
                {
                    break;
                }
                await Task.Delay(30000); // Wait 30 seconds
            }

            return response.Arn;
        }
        catch (ValidationException ex)
        {
            if (ex.Message.Contains("already enabled"))
                Console.WriteLine("Baseline is already enabled for this target");
            else { Console.WriteLine(ex.Message); }
            // Write the message and return null if baseline cannot be enabled.
            return null;
        }
        catch (AmazonControlTowerException ex)
        {
            Console.WriteLine($"Couldn't enable baseline. Here's why:
{ex.ErrorCode}: {ex.Message}");
            throw;
        }
    }
```

- API 세부 정보는 AWS SDK for .NET API 참조의 EnableBaseline을 참조하세요.

`EnableControl`

다음 코드 예시는 `EnableControl`의 사용 방법을 보여 줍니다.

SDK for .NET (v4)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

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

- API 세부 정보는 AWS SDK for .NET API 참조의 EnableControl을 참조하세요.

GetBaselineOperation

다음 코드 예시는 GetBaselineOperation의 사용 방법을 보여 줍니다.

SDK for .NET (v4)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

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

- API 세부 정보는 AWS SDK for .NET API 참조의 GetBaselineOperation을 참조하세요.

`GetControlOperation`

다음 코드 예시는 `GetControlOperation`의 사용 방법을 보여 줍니다.

SDK for .NET (v4)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

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

- API 세부 정보는 AWS SDK for .NET API 참조의 GetControlOperation을 참조하세요.

`ListBaselines`

다음 코드 예시는 `ListBaselines`의 사용 방법을 보여 줍니다.

SDK for .NET (v4)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

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

- API 세부 정보는 AWS SDK for .NET API 참조의 ListBaselines을 참조하세요.

`ListEnabledBaselines`

다음 코드 예시는 `ListEnabledBaselines`의 사용 방법을 보여 줍니다.

SDK for .NET (v4)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

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

- API 세부 정보는 AWS SDK for .NET API 참조의 ListEnabledBaselines을 참조하세요.

`ListEnabledControls`

다음 코드 예시는 `ListEnabledControls`의 사용 방법을 보여 줍니다.

SDK for .NET (v4)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

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

- API 세부 정보는 AWS SDK for .NET API 참조의 ListEnabledControls을 참조하세요.

`ListLandingZones`

다음 코드 예시는 `ListLandingZones`의 사용 방법을 보여 줍니다.

SDK for .NET (v4)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

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

- API 세부 정보는 AWS SDK for .NET API 참조의 ListLandingZones을 참조하세요.

`ResetEnabledBaseline`

다음 코드 예시는 `ResetEnabledBaseline`의 사용 방법을 보여 줍니다.

SDK for .NET (v4)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

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

- API 세부 정보는 AWS SDK for .NET API 참조의 ResetEnabledBaseline을 참조하세요.

SDK for .NET (v4)를 사용한 DynamoDB 예제

다음 코드 예제에서는 DynamoDB에서 AWS SDK for .NET (v4)를 사용하여 작업을 수행하고 일반적인 시나리오를 구현하는 방법을 보여줍니다.

기본 사항은 서비스 내에서 필수 작업을 수행하는 방법을 보여주는 코드 예제입니다.

작업은 대규모 프로그램에서 발췌한 코드이며 컨텍스트에 맞춰 실행해야 합니다. 작업은 개별 서비스 함수를 직접적으로 호출하는 방법을 보여주며 관련 시나리오의 컨텍스트에 맞는 작업을 볼 수 있습니다.

각 예시에는 전체 소스 코드에 대한 링크가 포함되어 있으며, 여기에서 컨텍스트에 맞춰 코드를 설정하고 실행하는 방법에 대한 지침을 찾을 수 있습니다.

주제

- 시작하기
- 기본 사항
- 작업

시작하기

Hello DynamoDB

다음 코드 예제는 DynamoDB를 사용하여 시작하는 방법을 보여 줍니다.

SDK for .NET (v4)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

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

- API 세부 정보는 AWS SDK for .NET API 참조의 ListTables를 참조하세요.

기본 사항

기본 사항 알아보기

다음 코드 예제에서는 다음과 같은 작업을 수행하는 방법을 보여줍니다.

- 영화 데이터를 저장할 수 있는 테이블을 생성합니다.
- 테이블에 하나의 영화를 추가하고 가져오고 업데이트합니다.
- 샘플 JSON 파일에서 테이블에 영화 데이터를 씁니다.
- 특정 연도에 개봉된 영화를 쿼리합니다.
- 특정 연도 범위 동안 개봉된 영화를 스캔합니다.
- 테이블에서 영화를 삭제한 다음, 테이블을 삭제합니다.

SDK for .NET (v4)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

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

테이블 작업에 주입된 클라이언트를 사용합니다.

```
using System.Text.Json;
using Amazon.DynamoDBv2;
using Amazon.DynamoDBv2.DataModel;
using Amazon.DynamoDBv2.DocumentModel;
using Amazon.DynamoDBv2.Model;

namespace DynamoDBActions;
```

```
/// <summary>
/// Methods of this class perform Amazon DynamoDB operations.
/// </summary>
public class DynamoDbWrapper
{
    private readonly IAmazonDynamoDB _amazonDynamoDB;

    /// <summary>
    /// Constructor for the DynamoDbWrapper class.
    /// </summary>
    /// <param name="amazonDynamoDB">The injected DynamoDB client.</param>
    public DynamoDbWrapper(IAmazonDynamoDB amazonDynamoDB)
    {
        _amazonDynamoDB = amazonDynamoDB;
    }
```

영화 데이터를 포함할 테이블을 생성합니다.

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

테이블에 하나의 영화를 추가합니다.

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

테이블에서 하나의 항목을 업데이트합니다.

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

영화 테이블에서 하나의 항목을 가져옵니다.

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

영화 테이블에 항목 배치를 씁니다.

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

테이블에서 하나의 항목을 삭제합니다.

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

테이블에서 특정 연도에 릴리스된 영화를 쿼리합니다.

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

테이블에서 특정 연도 범위 동안 릴리스된 영화를 스캔합니다.

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

영화 테이블을 삭제합니다.

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

- API 세부 정보는 AWS SDK for .NET API 참조의 다음 주제를 참조하세요.
  - BatchWriteItem
  - CreateTable
  - DeleteItem
  - DeleteTable
  - DescribeTable
  - GetItem
  - PutItem
  - Query
  - Scan
  - UpdateItem

작업

`BatchWriteItem`

다음 코드 예시는 `BatchWriteItem`의 사용 방법을 보여 줍니다.

SDK for .NET (v4)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

영화 테이블에 항목 배치를 씁니다.

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

- API 세부 정보는 AWS SDK for .NET API 참조의 BatchWriteItem을 참조하세요.

`CreateTable`

다음 코드 예시는 `CreateTable`의 사용 방법을 보여 줍니다.

SDK for .NET (v4)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

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

- API 세부 정보는 AWS SDK for .NET API 참조의 CreateTable을 참조하세요.

`DeleteItem`

다음 코드 예시는 `DeleteItem`의 사용 방법을 보여 줍니다.

SDK for .NET (v4)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

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

- API 세부 정보는 AWS SDK for .NET API 참조의 DeleteItem을 참조하세요.

`DeleteTable`

다음 코드 예시는 `DeleteTable`의 사용 방법을 보여 줍니다.

SDK for .NET (v4)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

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

- API 세부 정보는 AWS SDK for .NET API 참조의 DeleteTable을 참조하세요.

`GetItem`

다음 코드 예시는 `GetItem`의 사용 방법을 보여 줍니다.

SDK for .NET (v4)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

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

- API 세부 정보는 AWS SDK for .NET API 참조의 GetItem을 참조하세요.

`PutItem`

다음 코드 예시는 `PutItem`의 사용 방법을 보여 줍니다.

SDK for .NET (v4)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

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

- API 세부 정보는 AWS SDK for .NET API 참조의 PutItem을 참조하세요.

`Query`

다음 코드 예시는 `Query`의 사용 방법을 보여 줍니다.

SDK for .NET (v4)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

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

- API 세부 정보는 AWS SDK for .NET API 참조의 Query를 참조하세요.

Scan

다음 코드 예시는 Scan의 사용 방법을 보여 줍니다.

SDK for .NET (v4)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

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

- API 세부 정보는 AWS SDK for .NET API 참조의 Scan을 참조하세요.

`UpdateItem`

다음 코드 예시는 `UpdateItem`의 사용 방법을 보여 줍니다.

SDK for .NET (v4)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

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

- API 세부 정보는 AWS SDK for .NET API 참조의 UpdateItem을 참조하세요.

SDK for .NET (v4)를 사용한 Amazon EC2 예제

다음 코드 예제에서는 Amazon EC2에서 AWS SDK for .NET (v4)를 사용하여 작업을 수행하고 일반적인 시나리오를 구현하는 방법을 보여줍니다.

각 예시에는 전체 소스 코드에 대한 링크가 포함되어 있으며, 여기에서 컨텍스트에 맞춰 코드를 설정하고 실행하는 방법에 대한 지침을 찾을 수 있습니다.

주제

- 시작하기

시작하기

Hello Amazon EC2

다음 코드 예제에서는 Amazon EC2 사용을 시작하는 방법을 보여줍니다.

SDK for .NET (v4)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

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

- API 세부 정보는 AWS SDK for .NET API 참조의 DescribeSecurityGroups를 참조하세요.

SDK for .NET (v4)를 사용한 Amazon ECS 예제

다음 코드 예제에서는 Amazon ECS에서 AWS SDK for .NET (v4)를 사용하여 작업을 수행하고 일반적인 시나리오를 구현하는 방법을 보여줍니다.

각 예시에는 전체 소스 코드에 대한 링크가 포함되어 있으며, 여기에서 컨텍스트에 맞춰 코드를 설정하고 실행하는 방법에 대한 지침을 찾을 수 있습니다.

주제

- 시작하기

시작하기

Hello Amazon ECS

다음 코드 예제에서는 Amazon ECS 사용을 시작하는 방법을 보여줍니다.

SDK for .NET (v4)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
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.");
        }

    }
}
```

- API에 대한 세부 정보는 AWS SDK for .NET API 참조의 ListClusters를 참조하세요.

AWS IoT SDK for .NET (v4)를 사용한 예제

다음 코드 예제에서는 AWS SDK for .NET (v4)를와 함께 사용하여 작업을 수행하고 일반적인 시나리오를 구현하는 방법을 보여줍니다 AWS IoT.

기본 사항은 서비스 내에서 필수 작업을 수행하는 방법을 보여주는 코드 예제입니다.

작업은 대규모 프로그램에서 발췌한 코드이며 컨텍스트에 맞춰 실행해야 합니다. 작업은 개별 서비스 함수를 직접적으로 호출하는 방법을 보여주며 관련 시나리오의 컨텍스트에 맞는 작업을 볼 수 있습니다.

각 예시에는 전체 소스 코드에 대한 링크가 포함되어 있으며, 여기에서 컨텍스트에 맞춰 코드를 설정하고 실행하는 방법에 대한 지침을 찾을 수 있습니다.

주제

- 시작하기
- 기본 사항
- 작업

시작하기

안녕하세요 AWS IoT

다음 코드 예제에서는 AWS IoT를 사용하여 시작하는 방법을 보여 줍니다.

SDK for .NET (v4)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
/// <summary>
/// Hello AWS IoT example.
/// </summary>
public class HelloIoT
{
    /// <summary>
    /// Main method to run the Hello IoT example.
    /// </summary>
    /// <param name="args">Command line arguments.</param>
    /// <returns>A Task object.</returns>
    public static async Task Main(string[] args)
    {
        var iotClient = new AmazonIoTClient();

        try
        {
            Console.WriteLine("Hello AWS IoT! Let's list your IoT Things:");
            Console.WriteLine(new string('-', 80));

            // Use pages of 10.
            var request = new ListThingsRequest()
            {
                MaxResults = 10
            };
            var response = await iotClient.ListThingsAsync(request);

            // Since there is not a built-in paginator, use the NextMarker to
 paginate.
```

```
            bool hasMoreResults = true;

            var things = new List<ThingAttribute>();
            while (hasMoreResults)
            {
                things.AddRange(response.Things);

                // If NextMarker is not null, there are more results. Get the next
page of results.
                if (!String.IsNullOrEmpty(response.NextMarker))
                {
                    request.Marker = response.NextMarker;
                    response = await iotClient.ListThingsAsync(request);
                }
                else
                    hasMoreResults = false;
            }

            if (things is { Count: > 0 })
            {
                Console.WriteLine($"Found {things.Count} IoT Things:");
                foreach (var thing in things)
                {
                    Console.WriteLine($"- Thing Name: {thing.ThingName}");
                    Console.WriteLine($"  Thing ARN: {thing.ThingArn}");
                    Console.WriteLine($"  Thing Type: {thing.ThingTypeName ?? "No
type specified"}");
                    Console.WriteLine($"  Version: {thing.Version}");

                    if (thing.Attributes?.Count > 0)
                    {
                        Console.WriteLine("  Attributes:");
                        foreach (var attr in thing.Attributes)
                        {
                            Console.WriteLine($"    {attr.Key}: {attr.Value}");
                        }
                    }
                    Console.WriteLine();
                }
            }
            else
            {
                Console.WriteLine("No IoT Things found in your account.");
```

```
                Console.WriteLine("You can create IoT Things using the IoT Basics
 scenario example.");
            }

            Console.WriteLine("Hello IoT completed successfully.");
        }
        catch (Amazon.IoT.Model.ThrottlingException ex)
        {
            Console.WriteLine($"Request throttled, please try again later:
 {ex.Message}");
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Couldn't list Things. Here's why: {ex.Message}");
        }
    }
}
```

- API 세부 정보는 AWS SDK for .NET API 참조의 listThings를 참조하세요.

기본 사항

기본 사항 알아보기

다음 코드 예제에서는 다음과 같은 작업을 수행하는 방법을 보여줍니다.

- AWS IoT 사물을 생성합니다.
- 디바이스 인증서를 생성합니다.
- 속성을 사용하여 AWS IoT 사물을 업데이트합니다.
- 고유한 엔드포인트를 반환합니다.
- AWS IoT 인증서를 나열합니다.
- 섀도우를 업데이트 AWS IoT 합니다.
- 상태 정보를 씁니다.
- 규칙을 생성합니다.
- 규칙을 나열합니다.
- 사물 이름을 사용하여 사물을 검색합니다.
- AWS IoT 사물을 삭제합니다.

SDK for .NET (v4)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

AWS IoT 기능을 보여주는 대화형 시나리오를 실행합니다.

```
/// <summary>
/// Scenario class for AWS IoT basics.
/// </summary>
public class IoTBasics
{
    public static bool IsInteractive = true;
    public static IoTWrapper? Wrapper = null;
    public static IAmazonCloudFormation? CloudFormationClient = null;
    public static ILogger<IoTBasics> logger = null!;
    private static IoTWrapper _iotWrapper = null!;
    private static IAmazonCloudFormation _amazonCloudFormation = null!;
    private static ILogger<IoTBasics> _logger = null!;

    private static string _stackName = "IoTBasicsStack";
    private static string _stackResourcePath = "../../../../../../scenarios/basics/
iot/iot_usecase/resources/cfn_template.yaml";

    /// <summary>
    /// Main method for the IoT Basics scenario.
    /// </summary>
    /// <param name="args">Command line arguments.</param>
    /// <returns>A Task object.</returns>
    public static async Task Main(string[] args)
    {
        // Set up dependency injection for the Amazon service.
        using var host = Host.CreateDefaultBuilder(args)
            .ConfigureServices((_, services) =>
                services.AddAWSService<IAmazonIoT>(new AWSOptions() { Region =
 RegionEndpoint.USEast1 })
                    .AddAWSService<IAmazonCloudFormation>()
                        .AddTransient<IoTWrapper>()
                        .AddLogging(builder => builder.AddConsole())
                        .AddSingleton<IAmazonIotData>(sp =>
```

```
                        {
                            var iotService = sp.GetRequiredService<IAmazonIoT>();
                            var request = new DescribeEndpointRequest
                            {
                                EndpointType = "iot:Data-ATS"
                            };
                            var response =
 iotService.DescribeEndpointAsync(request).Result;
                            return new AmazonIotDataClient($"https://
{response.EndpointAddress}/");
                        })
            )
            .Build();

        logger = LoggerFactory.Create(builder => builder.AddConsole())
            .CreateLogger<IoTBasics>();

        Wrapper = host.Services.GetRequiredService<IoTWrapper>();
        CloudFormationClient =
 host.Services.GetRequiredService<IAmazonCloudFormation>();

        // Set the private fields for backwards compatibility
        _logger = logger;
        _iotWrapper = Wrapper;
        _amazonCloudFormation = CloudFormationClient;

        Console.WriteLine(new string('-', 80));
        Console.WriteLine("Welcome to the AWS IoT example scenario.");
        Console.WriteLine("This example program demonstrates various interactions
 with the AWS Internet of Things (IoT) Core service.");
        Console.WriteLine();
        if (IsInteractive)
        {
            Console.WriteLine("Press Enter to continue...");
            Console.ReadLine();
        }
        Console.WriteLine(new string('-', 80));

        try
        {
            await RunScenarioAsync();
        }
        catch (Exception ex)
        {
```

```
            _logger.LogError(ex, "There was a problem running the scenario.");
            Console.WriteLine($"\nAn error occurred: {ex.Message}");
        }

        Console.WriteLine(new string('-', 80));
        Console.WriteLine("The AWS IoT scenario has successfully completed.");
        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Run the IoT Basics scenario.
    /// </summary>
    /// <returns>A Task object.</returns>
    public static async Task RunScenarioAsync()
    {
        // Use static properties if available, otherwise use private fields
        var iotWrapper = Wrapper ?? _iotWrapper;
        var cloudFormationClient = CloudFormationClient ?? _amazonCloudFormation;
        var scenarioLogger = logger ?? _logger;

        await RunScenarioInternalAsync(iotWrapper, cloudFormationClient,
 scenarioLogger);
    }

    /// <summary>
    /// Internal method to run the IoT Basics scenario with injected dependencies.
    /// </summary>
    /// <param name="iotWrapper">The IoT wrapper instance.</param>
    /// <param name="cloudFormationClient">The CloudFormation client instance.</
param>
    /// <param name="scenarioLogger">The logger instance.</param>
    /// <returns>A Task object.</returns>
    private static async Task RunScenarioInternalAsync(IoTWrapper iotWrapper,
 IAmazonCloudFormation cloudFormationClient, ILogger<IoTBasics> scenarioLogger)
    {
        string thingName = $"iot-thing-{Guid.NewGuid():N}";
        string certificateArn = "";
        string certificateId = "";
        string ruleName = $"iotruledefault";
        string snsTopicArn = "";

        try
        {
            // Step 1: Create an AWS IoT Thing
```

```
            Console.WriteLine(new string('-', 80));
            Console.WriteLine("1. Create an AWS IoT Thing.");
            Console.WriteLine("An AWS IoT Thing represents a virtual entity in the
AWS IoT service that can be associated with a physical device.");
            Console.WriteLine();

            if (IsInteractive)
            {
                Console.Write("Enter Thing name: ");
                var userInput = Console.ReadLine();
                if (!string.IsNullOrEmpty(userInput))
                    thingName = userInput;
            }
            else
            {
                Console.WriteLine($"Using default Thing name: {thingName}");
            }

            var thingArn = await iotWrapper.CreateThingAsync(thingName);
            Console.WriteLine($"{thingName} was successfully created. The ARN value
is {thingArn}");
            Console.WriteLine(new string('-', 80));

            // Step 1.1: List AWS IoT Things
            Console.WriteLine(new string('-', 80));
            Console.WriteLine("2. List AWS IoT Things.");
            Console.WriteLine("Now let's list the IoT Things to see the Thing we
just created.");
            Console.WriteLine();
            if (IsInteractive)
            {
                Console.WriteLine("Press Enter to continue...");
                Console.ReadLine();
            }

            var things = await iotWrapper.ListThingsAsync();
            Console.WriteLine($"Found {things.Count} IoT Things:");
            foreach (var thing in things.Take(10)) // Show first 10 things
            {
                Console.WriteLine($"Thing Name: {thing.ThingName}");
                Console.WriteLine($"Thing ARN: {thing.ThingArn}");
                if (thing.Attributes != null && thing.Attributes.Any())
                {
                    Console.WriteLine("Attributes:");
```

```
                    foreach (var attr in thing.Attributes)
                    {
                        Console.WriteLine($"  {attr.Key}: {attr.Value}");
                    }
                }
                Console.WriteLine("--------------");
            }
            Console.WriteLine();
            Console.WriteLine(new string('-', 80));

            // Step 2: Generate a Device Certificate
            Console.WriteLine(new string('-', 80));
            Console.WriteLine("3. Generate a device certificate.");
            Console.WriteLine("A device certificate performs a role in securing the
communication between devices (Things) and the AWS IoT platform.");
            Console.WriteLine();

            var createCert = "y";
            if (IsInteractive)
            {
                Console.Write($"Do you want to create a certificate for {thingName}?
(y/n)");
                createCert = Console.ReadLine();
            }
            else
            {
                Console.WriteLine($"Creating certificate for {thingName}...");
            }

            if (createCert?.ToLower() == "y")
            {
                var certificateResult = await
iotWrapper.CreateKeysAndCertificateAsync();
                if (certificateResult.HasValue)
                {
                    var (certArn, certPem, certId) = certificateResult.Value;
                    certificateArn = certArn;
                    certificateId = certId;

                    Console.WriteLine($"\nCertificate:");
                    // Show only first few lines of certificate for brevity
                    var lines = certPem.Split('\n');
                    for (int i = 0; i < Math.Min(lines.Length, 5); i++)
                    {
```

```
                    Console.WriteLine(lines[i]);
                }
                if (lines.Length > 5)
                {
                    Console.WriteLine("...");
                }

                Console.WriteLine($"\nCertificate ARN:");
                Console.WriteLine(certificateArn);

                // Step 3: Attach the Certificate to the AWS IoT Thing
                Console.WriteLine("Attach the certificate to the AWS IoT
Thing.");
                var attachResult = await
iotWrapper.AttachThingPrincipalAsync(thingName, certificateArn);
                if (attachResult)
                {
                    Console.WriteLine("Certificate attached to Thing
successfully.");
                }
                else
                {
                    Console.WriteLine("Failed to attach certificate to Thing.");
                }

                Console.WriteLine("Thing Details:");
                Console.WriteLine($"Thing Name: {thingName}");
                Console.WriteLine($"Thing ARN: {thingArn}");
            }
            else
            {
                Console.WriteLine("Failed to create certificate.");
            }
        }
        Console.WriteLine(new string('-', 80));

        // Step 4: Update an AWS IoT Thing with Attributes
        Console.WriteLine(new string('-', 80));
        Console.WriteLine("4. Update an AWS IoT Thing with Attributes.");
        Console.WriteLine("IoT Thing attributes, represented as key-value pairs,
offer a pivotal advantage in facilitating efficient data");
        Console.WriteLine("management and retrieval within the AWS IoT
ecosystem.");
        Console.WriteLine();
```

```
            if (IsInteractive)
            {
                Console.WriteLine("Press Enter to continue...");
                Console.ReadLine();
            }

            var attributes = new Dictionary<string, string>
            {
                { "Location", "Seattle" },
                { "DeviceType", "Sensor" },
                { "Firmware", "1.2.3" }
            };

            await iotWrapper.UpdateThingAsync(thingName, attributes);
            Console.WriteLine("Thing attributes updated successfully.");
            Console.WriteLine(new string('-', 80));

            // Step 5: Return a unique endpoint specific to the Amazon Web Services
account
            Console.WriteLine(new string('-', 80));
            Console.WriteLine("5. Return a unique endpoint specific to the Amazon
Web Services account.");
            Console.WriteLine();
            if (IsInteractive)
            {
                Console.WriteLine("Press Enter to continue...");
                Console.ReadLine();
            }

            var endpoint = await iotWrapper.DescribeEndpointAsync();
            if (endpoint != null)
            {
                var subdomain = endpoint.Split('.')[0];
                Console.WriteLine($"Extracted subdomain: {subdomain}");
                Console.WriteLine($"Full Endpoint URL: https://{endpoint}");
            }
            else
            {
                Console.WriteLine("Failed to retrieve endpoint.");
            }
            Console.WriteLine(new string('-', 80));

            // Step 6: List your AWS IoT certificates
            Console.WriteLine(new string('-', 80));
```

```
            Console.WriteLine("6. List your AWS IoT certificates");
            if (IsInteractive)
            {
                Console.WriteLine("Press Enter to continue...");
                Console.ReadLine();
            }

            var certificates = await iotWrapper.ListCertificatesAsync();
            foreach (var cert in certificates.Take(5)) // Show first 5 certificates
            {
                Console.WriteLine($"Cert id: {cert.CertificateId}");
                Console.WriteLine($"Cert Arn: {cert.CertificateArn}");
            }
            Console.WriteLine();
            Console.WriteLine(new string('-', 80));

            // Step 7: Create an IoT shadow
            Console.WriteLine(new string('-', 80));
            Console.WriteLine("7. Update an IoT shadow that refers to a digital
representation or virtual twin of a physical IoT device");
            Console.WriteLine();
            if (IsInteractive)
            {
                Console.WriteLine("Press Enter to continue...");
                Console.ReadLine();
            }

            var shadowPayload = JsonSerializer.Serialize(new
            {
                state = new
                {
                    desired = new
                    {
                        temperature = 25,
                        humidity = 50
                    }
                }
            });

            await iotWrapper.UpdateThingShadowAsync(thingName, shadowPayload);
            Console.WriteLine("Thing Shadow updated successfully.");
            Console.WriteLine(new string('-', 80));

            // Step 8: Write out the state information, in JSON format
```

```
            Console.WriteLine(new string('-', 80));
            Console.WriteLine("8. Write out the state information, in JSON
format.");
            if (IsInteractive)
            {
                Console.WriteLine("Press Enter to continue...");
                Console.ReadLine();
            }

            var shadowData = await iotWrapper.GetThingShadowAsync(thingName);
            Console.WriteLine($"Received Shadow Data: {shadowData}");
            Console.WriteLine(new string('-', 80));

            // Step 9: Set up resources (SNS topic and IAM role) and create a rule
            Console.WriteLine(new string('-', 80));
            Console.WriteLine("9. Set up resources and create a rule");
            Console.WriteLine();

            // Deploy CloudFormation stack to create SNS topic and IAM role
            Console.WriteLine("Deploying CloudFormation stack to create SNS topic
and IAM role...");

            var deployStack = !IsInteractive || GetYesNoResponse("Would you like to
deploy the CloudFormation stack? (y/n) ");
            if (deployStack)
            {
                if (IsInteractive)
                {
                    Console.Write(
                        $"Enter stack resource file path (or press Enter for default
'{_stackResourcePath}'): ");
                    var userResourcePath = Console.ReadLine();
                    if (!string.IsNullOrEmpty(userResourcePath))
                        _stackResourcePath = userResourcePath;
                }

                _stackName = PromptUserForStackName();

                var deploySuccess = await DeployCloudFormationStack(_stackName,
cloudFormationClient, scenarioLogger);

                if (deploySuccess)
                {
                    // Get stack outputs
```

```
                    var stackOutputs = await GetStackOutputs(_stackName,
cloudFormationClient, scenarioLogger);
                    if (stackOutputs != null)
                    {
                        snsTopicArn = stackOutputs["SNSTopicArn"];
                        string roleArn = stackOutputs["RoleArn"];

                        Console.WriteLine($"Successfully deployed stack. SNS topic:
{snsTopicArn}");
                        Console.WriteLine($"Successfully deployed stack. IAM role:
{roleArn}");

                        if (IsInteractive)
                        {
                            Console.Write($"Enter Rule name (press Enter for default
'{ruleName}'): ");
                            var userRuleName = Console.ReadLine();
                            if (!string.IsNullOrEmpty(userRuleName))
                                ruleName = userRuleName;
                        }
                        else
                        {
                            Console.WriteLine($"Using default rule name:
{ruleName}");
                        }

                        // Now create the IoT rule with the CloudFormation outputs
                        var ruleResult = await
iotWrapper.CreateTopicRuleAsync(ruleName, snsTopicArn, roleArn);
                        if (ruleResult)
                        {
                            Console.WriteLine("IoT Rule created successfully.");
                        }
                        else
                        {
                            Console.WriteLine("Failed to create IoT rule.");
                        }
                    }
                    else
                    {
                        Console.WriteLine("Failed to get stack outputs. Skipping
rule creation.");
                    }
                }
```

```
                else
                {
                    Console.WriteLine("Failed to deploy CloudFormation stack.
 Skipping rule creation.");
                }
            }
            else
            {
                Console.WriteLine("Skipping CloudFormation stack deployment and rule
 creation.");
            }
            Console.WriteLine(new string('-', 80));

            // Step 10: List your rules
            Console.WriteLine(new string('-', 80));
            Console.WriteLine("10. List your rules.");
            if (IsInteractive)
            {
                Console.WriteLine("Press Enter to continue...");
                Console.ReadLine();
            }

            var rules = await iotWrapper.ListTopicRulesAsync();
            Console.WriteLine("List of IoT Rules:");
            foreach (var rule in rules.Take(5)) // Show first 5 rules
            {
                Console.WriteLine($"Rule Name: {rule.RuleName}");
                Console.WriteLine($"Rule ARN: {rule.RuleArn}");
                Console.WriteLine("--------------");
            }
            Console.WriteLine();
            Console.WriteLine(new string('-', 80));

            // Step 11: Search things using the Thing name
            Console.WriteLine(new string('-', 80));
            Console.WriteLine("11. Search things using the Thing name.");
            if (IsInteractive)
            {
                Console.WriteLine("Press Enter to continue...");
                Console.ReadLine();
            }

            var searchResults = await iotWrapper.SearchIndexAsync($"thingName:
{thingName}");
```

```
            if (searchResults.Any())
            {
                Console.WriteLine($"Thing id found using search is
{searchResults.First().ThingId}");
            }
            else
            {
                Console.WriteLine($"No search results found for Thing:
{thingName}");
            }
            Console.WriteLine(new string('-', 80));

            // Step 12: Cleanup - Detach and delete certificate
            if (!string.IsNullOrEmpty(certificateArn))
            {
                Console.WriteLine(new string('-', 80));
                var deleteCert = "y";
                if (IsInteractive)
                {
                    Console.Write($"Do you want to detach and delete the certificate
for {thingName}? (y/n)");
                    deleteCert = Console.ReadLine();
                }
                else
                {
                    Console.WriteLine($"Detaching and deleting certificate for
{thingName}...");
                }

                if (deleteCert?.ToLower() == "y")
                {
                    Console.WriteLine("12. You selected to detach and delete the
certificate.");
                    if (IsInteractive)
                    {
                        Console.WriteLine("Press Enter to continue...");
                        Console.ReadLine();
                    }

                    await iotWrapper.DetachThingPrincipalAsync(thingName,
certificateArn);
                    Console.WriteLine($"{certificateArn} was successfully removed
from {thingName}");
```

```
                    await iotWrapper.DeleteCertificateAsync(certificateId);
                    Console.WriteLine($"{certificateArn} was successfully
deleted.");
                }
                Console.WriteLine(new string('-', 80));
            }

            // Step 13: Delete the AWS IoT Thing
            Console.WriteLine(new string('-', 80));
            Console.WriteLine("13. Delete the AWS IoT Thing.");
            var deleteThing = "y";
            if (IsInteractive)
            {
                Console.Write($"Do you want to delete the IoT Thing? (y/n)");
                deleteThing = Console.ReadLine();
            }
            else
            {
                Console.WriteLine($"Deleting IoT Thing {thingName}...");
            }

            if (deleteThing?.ToLower() == "y")
            {
                await iotWrapper.DeleteThingAsync(thingName);
                Console.WriteLine($"Deleted Thing {thingName}");
            }
            Console.WriteLine(new string('-', 80));

            // Step 14: Clean up CloudFormation stack
            if (!string.IsNullOrEmpty(snsTopicArn))
            {
                Console.WriteLine(new string('-', 80));
                Console.WriteLine("14. Clean up CloudFormation stack.");
                Console.WriteLine("Deleting the CloudFormation stack and all
resources...");

                var cleanup = !IsInteractive || GetYesNoResponse("Do you want to
delete the CloudFormation stack and all resources? (y/n) ");
                if (cleanup)
                {
                    var ruleCleanupSuccess = await
iotWrapper.DeleteTopicRuleAsync(ruleName);
```

```
                    var stackCleanupSuccess = await
 DeleteCloudFormationStack(_stackName, cloudFormationClient, scenarioLogger);
                    if (ruleCleanupSuccess && stackCleanupSuccess)
                    {
                        Console.WriteLine("Successfully cleaned up CloudFormation
 stack and all resources.");
                    }
                    else
                    {
                        Console.WriteLine("Some cleanup operations failed. Check the
 logs for details.");
                    }
                }
                else
                {
                    Console.WriteLine($"Resources will remain. Stack name:
 {_stackName}");
                }
                Console.WriteLine(new string('-', 80));
            }
        }
        catch (Exception ex)
        {
            scenarioLogger.LogError(ex, "Error occurred during scenario
 execution.");

            // Cleanup on error
            if (!string.IsNullOrEmpty(certificateArn) && !
string.IsNullOrEmpty(thingName))
            {
                try
                {
                    await iotWrapper.DetachThingPrincipalAsync(thingName,
 certificateArn);
                    await iotWrapper.DeleteCertificateAsync(certificateId);
                }
                catch (Exception cleanupEx)
                {
                    scenarioLogger.LogError(cleanupEx, "Error during cleanup.");
                }
            }

            if (!string.IsNullOrEmpty(thingName))
            {
```

```
                try
                {
                    await iotWrapper.DeleteThingAsync(thingName);
                }
                catch (Exception cleanupEx)
                {
                    scenarioLogger.LogError(cleanupEx, "Error during Thing
cleanup.");
                }
            }

            // Clean up CloudFormation stack on error
            if (!string.IsNullOrEmpty(snsTopicArn))
            {
                try
                {
                    await _iotWrapper.DeleteTopicRuleAsync(ruleName);
                    await DeleteCloudFormationStack(_stackName,
cloudFormationClient, scenarioLogger);
                }
                catch (Exception cleanupEx)
                {
                    scenarioLogger.LogError(cleanupEx, "Error during CloudFormation
stack cleanup.");
                }
            }

            throw;
        }
    }

    /// <summary>
    /// Deploys the CloudFormation stack with the necessary resources.
    /// </summary>
    /// <param name="stackName">The name of the CloudFormation stack.</param>
    /// <param name="cloudFormationClient">The CloudFormation client.</param>
    /// <param name="scenarioLogger">The logger.</param>
    /// <returns>True if the stack was deployed successfully.</returns>
    private static async Task<bool> DeployCloudFormationStack(string stackName,
IAmazonCloudFormation cloudFormationClient, ILogger<IoTBasics> scenarioLogger)
    {
        Console.WriteLine($"\nDeploying CloudFormation stack: {stackName}");

        try
```

```
        {
            var request = new CreateStackRequest
            {
                StackName = stackName,
                TemplateBody = await File.ReadAllTextAsync(_stackResourcePath),
                Capabilities = new List<string> { Capability.CAPABILITY_NAMED_IAM }
            };

            var response = await cloudFormationClient.CreateStackAsync(request);

            if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
            {
                Console.WriteLine($"CloudFormation stack creation started:
{stackName}");

                bool stackCreated = await WaitForStackCompletion(response.StackId,
cloudFormationClient, scenarioLogger);

                if (stackCreated)
                {
                    Console.WriteLine("CloudFormation stack created successfully.");
                    return true;
                }
                else
                {
                    scenarioLogger.LogError($"CloudFormation stack creation failed:
{stackName}");
                    return false;
                }
            }
            else
            {
                scenarioLogger.LogError($"Failed to create CloudFormation stack:
{stackName}");
                return false;
            }
        }
        catch (AlreadyExistsException)
        {
            scenarioLogger.LogWarning($"CloudFormation stack '{stackName}' already
exists. Please provide a unique name.");
            var newStackName = PromptUserForStackName();
            return await DeployCloudFormationStack(newStackName,
cloudFormationClient, scenarioLogger);
```

```
        }
        catch (Exception ex)
        {
            scenarioLogger.LogError(ex, $"An error occurred while deploying the
CloudFormation stack: {stackName}");
            return false;
        }
    }

    /// <summary>
    /// Waits for the CloudFormation stack to be in the CREATE_COMPLETE state.
    /// </summary>
    /// <param name="stackId">The ID of the CloudFormation stack.</param>
    /// <param name="cloudFormationClient">The CloudFormation client.</param>
    /// <param name="scenarioLogger">The logger.</param>
    /// <returns>True if the stack was created successfully.</returns>
    private static async Task<bool> WaitForStackCompletion(string stackId,
IAmazonCloudFormation cloudFormationClient, ILogger<IoTBasics> scenarioLogger)
    {
        int retryCount = 0;
        const int maxRetries = 30;
        const int retryDelay = 10000;

        while (retryCount < maxRetries)
        {
            var describeStacksRequest = new DescribeStacksRequest
            {
                StackName = stackId
            };

            var describeStacksResponse = await
cloudFormationClient.DescribeStacksAsync(describeStacksRequest);

            if (describeStacksResponse.Stacks.Count > 0)
            {
                if (describeStacksResponse.Stacks[0].StackStatus ==
StackStatus.CREATE_COMPLETE)
                {
                    return true;
                }
                if (describeStacksResponse.Stacks[0].StackStatus ==
StackStatus.CREATE_FAILED ||
                    describeStacksResponse.Stacks[0].StackStatus ==
StackStatus.ROLLBACK_COMPLETE)
```

```
                {
                    return false;
                }
            }

            Console.WriteLine("Waiting for CloudFormation stack creation to
complete...");
            await Task.Delay(retryDelay);
            retryCount++;
        }

        scenarioLogger.LogError("Timed out waiting for CloudFormation stack creation
to complete.");
        return false;
    }

    /// <summary>
    /// Gets the outputs from the CloudFormation stack.
    /// </summary>
    /// <param name="stackName">The name of the CloudFormation stack.</param>
    /// <param name="cloudFormationClient">The CloudFormation client.</param>
    /// <param name="scenarioLogger">The logger.</param>
    /// <returns>A dictionary of stack outputs.</returns>
    private static async Task<Dictionary<string, string>?> GetStackOutputs(string
stackName, IAmazonCloudFormation cloudFormationClient, ILogger<IoTBasics>
scenarioLogger)
    {
        try
        {
            var describeStacksRequest = new DescribeStacksRequest
            {
                StackName = stackName
            };

            var response = await
cloudFormationClient.DescribeStacksAsync(describeStacksRequest);

            if (response.Stacks.Count > 0)
            {
                var outputs = new Dictionary<string, string>();
                foreach (var output in response.Stacks[0].Outputs)
                {
                    outputs[output.OutputKey] = output.OutputValue;
                }
```

```
                return outputs;
            }

            return null;
        }
        catch (Exception ex)
        {
            scenarioLogger.LogError(ex, $"Failed to get stack outputs for
{stackName}");
            return null;
        }
    }

    /// <summary>
    /// Deletes the CloudFormation stack and waits for confirmation.
    /// </summary>
    /// <param name="stackName">The name of the CloudFormation stack.</param>
    /// <param name="cloudFormationClient">The CloudFormation client.</param>
    /// <param name="scenarioLogger">The logger.</param>
    /// <returns>True if the stack was deleted successfully.</returns>
    private static async Task<bool> DeleteCloudFormationStack(string stackName,
IAmazonCloudFormation cloudFormationClient, ILogger<IoTBasics> scenarioLogger)
    {
        try
        {
            var request = new DeleteStackRequest
            {
                StackName = stackName
            };

            await cloudFormationClient.DeleteStackAsync(request);
            Console.WriteLine($"CloudFormation stack '{stackName}' is being deleted.
This may take a few minutes.");

            bool stackDeleted = await WaitForStackDeletion(stackName,
cloudFormationClient, scenarioLogger);

            if (stackDeleted)
            {
                Console.WriteLine($"CloudFormation stack '{stackName}' has been
deleted.");
                return true;
            }
            else
```

```
            {
                scenarioLogger.LogError($"Failed to delete CloudFormation stack
'{stackName}'.");
                return false;
            }
        }
        catch (Exception ex)
        {
            scenarioLogger.LogError(ex, $"An error occurred while deleting the
CloudFormation stack: {stackName}");
            return false;
        }
    }

    /// <summary>
    /// Waits for the stack to be deleted.
    /// </summary>
    /// <param name="stackName">The name of the CloudFormation stack.</param>
    /// <param name="cloudFormationClient">The CloudFormation client.</param>
    /// <param name="scenarioLogger">The logger.</param>
    /// <returns>True if the stack was deleted successfully.</returns>
    private static async Task<bool> WaitForStackDeletion(string stackName,
IAmazonCloudFormation cloudFormationClient, ILogger<IoTBasics> scenarioLogger)
    {
        int retryCount = 0;
        const int maxRetries = 30;
        const int retryDelay = 10000;

        while (retryCount < maxRetries)
        {
            var describeStacksRequest = new DescribeStacksRequest
            {
                StackName = stackName
            };

            try
            {
                var describeStacksResponse = await
cloudFormationClient.DescribeStacksAsync(describeStacksRequest);

                if (describeStacksResponse.Stacks.Count == 0 ||
                    describeStacksResponse.Stacks[0].StackStatus ==
StackStatus.DELETE_COMPLETE)
                {
```

```
                    return true;
                }
            }
            catch (AmazonCloudFormationException ex) when (ex.ErrorCode ==
"ValidationError")
            {
                return true;
            }

            Console.WriteLine($"Waiting for CloudFormation stack '{stackName}' to be
deleted...");
            await Task.Delay(retryDelay);
            retryCount++;
        }

        scenarioLogger.LogError($"Timed out waiting for CloudFormation stack
'{stackName}' to be deleted.");
        return false;
    }

    /// <summary>
    /// Helper method to get a yes or no response from the user.
    /// </summary>
    private static bool GetYesNoResponse(string question)
    {
        Console.WriteLine(question);
        var ynResponse = Console.ReadLine();
        var response = ynResponse != null && ynResponse.Equals("y",
StringComparison.InvariantCultureIgnoreCase);
        return response;
    }

    /// <summary>
    /// Prompts the user for a stack name.
    /// </summary>
    private static string PromptUserForStackName()
    {
        if (IsInteractive)
        {
            Console.Write($"Enter a name for the CloudFormation stack (press Enter
for default '{_stackName}'): ");
            string? input = Console.ReadLine();
            if (!string.IsNullOrWhiteSpace(input))
            {
```

```
                var regex = new System.Text.RegularExpressions.Regex("[a-zA-Z][-a-
zA-Z0-9]*");
                if (!regex.IsMatch(input))
                {
                    Console.WriteLine($"Invalid stack name. Using default:
 {_stackName}");
                    return _stackName;
                }
                return input;
            }
        }
        return _stackName;
    }
}
```

AWS IoT SDK 메서드의 래퍼 클래스입니다.

```
/// <summary>
/// Wrapper methods to use Amazon IoT Core with .NET.
/// </summary>
public class IoTWrapper
{
    private readonly IAmazonIoT _amazonIoT;
    private readonly IAmazonIotData _amazonIotData;
    private readonly ILogger<IoTWrapper> _logger;

    /// <summary>
    /// Constructor for the IoT wrapper.
    /// </summary>
    /// <param name="amazonIoT">The injected IoT client.</param>
    /// <param name="amazonIotData">The injected IoT Data client.</param>
    /// <param name="logger">The injected logger.</param>
    public IoTWrapper(IAmazonIoT amazonIoT, IAmazonIotData amazonIotData,
 ILogger<IoTWrapper> logger)
    {
        _amazonIoT = amazonIoT;
        _amazonIotData = amazonIotData;
        _logger = logger;
    }

    /// <summary>
    /// Creates an AWS IoT Thing.
```

```
    /// </summary>
    /// <param name="thingName">The name of the Thing to create.</param>
    /// <returns>The ARN of the Thing created, or null if creation failed.</returns>
    public async Task<string?> CreateThingAsync(string thingName)
    {
        try
        {
            var request = new CreateThingRequest
            {
                ThingName = thingName
            };

            var response = await _amazonIoT.CreateThingAsync(request);
            _logger.LogInformation($"Created Thing {thingName} with ARN
{response.ThingArn}");
            return response.ThingArn;
        }
        catch (Amazon.IoT.Model.ResourceAlreadyExistsException ex)
        {
            _logger.LogWarning($"Thing {thingName} already exists: {ex.Message}");
            return null;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't create Thing {thingName}. Here's why:
{ex.Message}");
            return null;
        }
    }

    /// <summary>
    /// Creates a device certificate for AWS IoT.
    /// </summary>
    /// <returns>The certificate details including ARN and certificate PEM, or null
if creation failed.</returns>
    public async Task<(string CertificateArn, string CertificatePem, string
CertificateId)?> CreateKeysAndCertificateAsync()
    {
        try
        {
            var request = new CreateKeysAndCertificateRequest
            {
                SetAsActive = true
            };
```

```
            var response = await _amazonIoT.CreateKeysAndCertificateAsync(request);
            _logger.LogInformation($"Created certificate with ARN
{response.CertificateArn}");
            return (response.CertificateArn, response.CertificatePem,
response.CertificateId);
        }
        catch (Amazon.IoT.Model.ThrottlingException ex)
        {
            _logger.LogWarning($"Request throttled, please try again later:
{ex.Message}");
            return null;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't create certificate. Here's why:
{ex.Message}");
            return null;
        }
    }

    /// <summary>
    /// Attaches a certificate to an IoT Thing.
    /// </summary>
    /// <param name="thingName">The name of the Thing.</param>
    /// <param name="certificateArn">The ARN of the certificate to attach.</param>
    /// <returns>True if successful, false otherwise.</returns>
    public async Task<bool> AttachThingPrincipalAsync(string thingName, string
certificateArn)
    {
        try
        {
            var request = new AttachThingPrincipalRequest
            {
                ThingName = thingName,
                Principal = certificateArn
            };

            await _amazonIoT.AttachThingPrincipalAsync(request);
            _logger.LogInformation($"Attached certificate {certificateArn} to Thing
{thingName}");
            return true;
        }
        catch (Amazon.IoT.Model.ResourceNotFoundException ex)
```

```
        {
            _logger.LogError($"Cannot attach certificate - resource not found:
{ex.Message}");
            return false;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't attach certificate to Thing. Here's why:
{ex.Message}");
            return false;
        }
    }

    /// <summary>
    /// Updates an IoT Thing with attributes.
    /// </summary>
    /// <param name="thingName">The name of the Thing to update.</param>
    /// <param name="attributes">Dictionary of attributes to add.</param>
    /// <returns>True if successful, false otherwise.</returns>
    public async Task<bool> UpdateThingAsync(string thingName, Dictionary<string,
string> attributes)
    {
        try
        {
            var request = new UpdateThingRequest
            {
                ThingName = thingName,
                AttributePayload = new AttributePayload
                {
                    Attributes = attributes,
                    Merge = true
                }
            };

            await _amazonIoT.UpdateThingAsync(request);
            _logger.LogInformation($"Updated Thing {thingName} with attributes");
            return true;
        }
        catch (Amazon.IoT.Model.ResourceNotFoundException ex)
        {
            _logger.LogError($"Cannot update Thing - resource not found:
{ex.Message}");
            return false;
        }
```

```
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't update Thing attributes. Here's why:
{ex.Message}");
            return false;
        }
    }

    /// <summary>
    /// Gets the AWS IoT endpoint URL.
    /// </summary>
    /// <returns>The endpoint URL, or null if retrieval failed.</returns>
    public async Task<string?> DescribeEndpointAsync()
    {
        try
        {
            var request = new DescribeEndpointRequest
            {
                EndpointType = "iot:Data-ATS"
            };

            var response = await _amazonIoT.DescribeEndpointAsync(request);
            _logger.LogInformation($"Retrieved endpoint:
{response.EndpointAddress}");
            return response.EndpointAddress;
        }
        catch (Amazon.IoT.Model.ThrottlingException ex)
        {
            _logger.LogWarning($"Request throttled, please try again later:
{ex.Message}");
            return null;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't describe endpoint. Here's why:
{ex.Message}");
            return null;
        }
    }

    /// <summary>
    /// Lists all certificates associated with the account.
    /// </summary>
```

```
    /// <returns>List of certificate information, or empty list if listing failed.</
returns>
    public async Task<List<Certificate>> ListCertificatesAsync()
    {
        try
        {
            var request = new ListCertificatesRequest();
            var response = await _amazonIoT.ListCertificatesAsync(request);

            _logger.LogInformation($"Retrieved {response.Certificates.Count}
 certificates");
            return response.Certificates;
        }
        catch (Amazon.IoT.Model.ThrottlingException ex)
        {
            _logger.LogWarning($"Request throttled, please try again later:
 {ex.Message}");
            return new List<Certificate>();
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't list certificates. Here's why:
 {ex.Message}");
            return new List<Certificate>();
        }
    }

    /// <summary>
    /// Updates the Thing's shadow with new state information.
    /// </summary>
    /// <param name="thingName">The name of the Thing.</param>
    /// <param name="shadowPayload">The shadow payload in JSON format.</param>
    /// <returns>True if successful, false otherwise.</returns>
    public async Task<bool> UpdateThingShadowAsync(string thingName, string
 shadowPayload)
    {
        try
        {
            var request = new UpdateThingShadowRequest
            {
                ThingName = thingName,
                Payload = new
 MemoryStream(System.Text.Encoding.UTF8.GetBytes(shadowPayload))
            };
```

```
            await _amazonIotData.UpdateThingShadowAsync(request);
            _logger.LogInformation($"Updated shadow for Thing {thingName}");
            return true;
        }
        catch (Amazon.IotData.Model.ResourceNotFoundException ex)
        {
            _logger.LogError($"Cannot update Thing shadow - resource not found:
{ex.Message}");
            return false;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't update Thing shadow. Here's why:
{ex.Message}");
            return false;
        }
    }

    /// <summary>
    /// Gets the Thing's shadow information.
    /// </summary>
    /// <param name="thingName">The name of the Thing.</param>
    /// <returns>The shadow data as a string, or null if retrieval failed.</returns>
    public async Task<string?> GetThingShadowAsync(string thingName)
    {
        try
        {
            var request = new GetThingShadowRequest
            {
                ThingName = thingName
            };

            var response = await _amazonIotData.GetThingShadowAsync(request);
            using var reader = new StreamReader(response.Payload);
            var shadowData = await reader.ReadToEndAsync();

            _logger.LogInformation($"Retrieved shadow for Thing {thingName}");
            return shadowData;
        }
        catch (Amazon.IotData.Model.ResourceNotFoundException ex)
        {
            _logger.LogError($"Cannot get Thing shadow - resource not found:
{ex.Message}");
```

```
            return null;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't get Thing shadow. Here's why:
{ex.Message}");
            return null;
        }
    }

    /// <summary>
    /// Creates an IoT topic rule.
    /// </summary>
    /// <param name="ruleName">The name of the rule.</param>
    /// <param name="snsTopicArn">The ARN of the SNS topic for the action.</param>
    /// <param name="roleArn">The ARN of the IAM role.</param>
    /// <returns>True if successful, false otherwise.</returns>
    public async Task<bool> CreateTopicRuleAsync(string ruleName, string
snsTopicArn, string roleArn)
    {
        try
        {
            var request = new CreateTopicRuleRequest
            {
                RuleName = ruleName,
                TopicRulePayload = new TopicRulePayload
                {
                    Sql = "SELECT * FROM 'topic/subtopic'",
                    Description = $"Rule created by .NET example: {ruleName}",
                    Actions = new List<Amazon.IoT.Model.Action>
                    {
                        new Amazon.IoT.Model.Action
                        {
                            Sns = new SnsAction
                            {
                                TargetArn = snsTopicArn,
                                RoleArn = roleArn
                            }
                        },
                    },
                    RuleDisabled = false
                }
            };
```

```
            await _amazonIoT.CreateTopicRuleAsync(request);
            _logger.LogInformation($"Created IoT rule {ruleName}");
            return true;
        }
        catch (Amazon.IoT.Model.ResourceAlreadyExistsException ex)
        {
            _logger.LogWarning($"Rule {ruleName} already exists: {ex.Message}");
            return false;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't create topic rule. Here's why:
{ex.Message}");
            return false;
        }
    }

    /// <summary>
    /// Deletes an IoT topic rule.
    /// </summary>
    /// <param name="ruleName">The name of the rule.</param>
    /// <returns>True if successful, false otherwise.</returns>
    public async Task<bool> DeleteTopicRuleAsync(string ruleName)
    {
        try
        {
            var request = new DeleteTopicRuleRequest
            {
                RuleName = ruleName,
            };

            await _amazonIoT.DeleteTopicRuleAsync(request);
            _logger.LogInformation($"Deleted IoT rule {ruleName}");
            return true;
        }
        catch (Amazon.IoT.Model.ResourceNotFoundException ex)
        {
            _logger.LogWarning($"Rule {ruleName} not found: {ex.Message}");
            return false;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't delete topic rule. Here's why:
{ex.Message}");
```

```
            return false;
        }
    }

    /// <summary>
    /// Lists all IoT topic rules.
    /// </summary>
    /// <returns>List of topic rules, or empty list if listing failed.</returns>
    public async Task<List<TopicRuleListItem>> ListTopicRulesAsync()
    {
        try
        {
            var request = new ListTopicRulesRequest();
            var response = await _amazonIoT.ListTopicRulesAsync(request);

            _logger.LogInformation($"Retrieved {response.Rules.Count} IoT rules");
            return response.Rules;
        }
        catch (Amazon.IoT.Model.ThrottlingException ex)
        {
            _logger.LogWarning($"Request throttled, please try again later:
{ex.Message}");
            return new List<TopicRuleListItem>();
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't list topic rules. Here's why:
{ex.Message}");
            return new List<TopicRuleListItem>();
        }
    }

    /// <summary>
    /// Searches for IoT Things using the search index.
    /// </summary>
    /// <param name="queryString">The search query string.</param>
    /// <returns>List of Things that match the search criteria, or empty list if
search failed.</returns>
    public async Task<List<ThingDocument>> SearchIndexAsync(string queryString)
    {
        try
        {
            // First, try to perform the search
            var request = new SearchIndexRequest
```

```
            {
                QueryString = queryString
            };

            var response = await _amazonIoT.SearchIndexAsync(request);
            _logger.LogInformation($"Search found {response.Things.Count} Things");
            return response.Things;
        }
        catch (Amazon.IoT.Model.IndexNotReadyException ex)
        {
            _logger.LogWarning($"Search index not ready, setting up indexing
configuration: {ex.Message}");
            return await SetupIndexAndRetrySearchAsync(queryString);
        }
        catch (Amazon.IoT.Model.ResourceNotFoundException ex) when
(ex.Message.Contains("index") || ex.Message.Contains("Index"))
        {
            _logger.LogWarning($"Search index not configured, setting up indexing
configuration: {ex.Message}");
            return await SetupIndexAndRetrySearchAsync(queryString);
        }
        catch (Amazon.IoT.Model.ThrottlingException ex)
        {
            _logger.LogWarning($"Request throttled, please try again later:
{ex.Message}");
            return new List<ThingDocument>();
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't search index. Here's why: {ex.Message}");
            return new List<ThingDocument>();
        }
    }

    /// <summary>
    /// Sets up the indexing configuration and retries the search after waiting for
the index to be ready.
    /// </summary>
    /// <param name="queryString">The search query string.</param>
    /// <returns>List of Things that match the search criteria, or empty list if
setup/search failed.</returns>
    private async Task<List<ThingDocument>> SetupIndexAndRetrySearchAsync(string
queryString)
    {
```

```
        try
        {
            // Update indexing configuration to REGISTRY mode
            _logger.LogInformation("Setting up IoT search indexing
configuration...");
            await _amazonIoT.UpdateIndexingConfigurationAsync(
                new UpdateIndexingConfigurationRequest()
                {
                    ThingIndexingConfiguration = new ThingIndexingConfiguration()
                    {
                        ThingIndexingMode = ThingIndexingMode.REGISTRY
                    }
                });

            _logger.LogInformation("Indexing configuration updated. Waiting for
index to be ready...");

            // Wait for the index to be set up - this can take some time
            const int maxRetries = 10;
            const int retryDelaySeconds = 10;

            for (int attempt = 1; attempt <= maxRetries; attempt++)
            {
                try
                {
                    _logger.LogInformation($"Waiting for index to be ready (attempt
{attempt}/{maxRetries})...");
                    await Task.Delay(TimeSpan.FromSeconds(retryDelaySeconds));

                    // Try to get the current indexing configuration to see if it's
ready
                    var configResponse = await
_amazonIoT.GetIndexingConfigurationAsync(new GetIndexingConfigurationRequest());
                    if (configResponse.ThingIndexingConfiguration?.ThingIndexingMode
== ThingIndexingMode.REGISTRY)
                    {
                        // Try the search again
                        var request = new SearchIndexRequest
                        {
                            QueryString = queryString
                        };

                        var response = await _amazonIoT.SearchIndexAsync(request);
```

```
                        _logger.LogInformation($"Search found
{response.Things.Count} Things after index setup");
                        return response.Things;
                    }
                }
                catch (Amazon.IoT.Model.IndexNotReadyException)
                {
                    // Index still not ready, continue waiting
                    _logger.LogInformation("Index still not ready, continuing to
wait...");
                    continue;
                }
                catch (Amazon.IoT.Model.InvalidRequestException ex) when
(ex.Message.Contains("index") || ex.Message.Contains("Index"))
                {
                    // Index still not ready, continue waiting
                    _logger.LogInformation("Index still not ready, continuing to
wait...");
                    continue;
                }
            }

            _logger.LogWarning("Timeout waiting for search index to be ready after
configuration update");
            return new List<ThingDocument>();
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't set up search index configuration. Here's
why: {ex.Message}");
            return new List<ThingDocument>();
        }
    }

    /// <summary>
    /// Detaches a certificate from an IoT Thing.
    /// </summary>
    /// <param name="thingName">The name of the Thing.</param>
    /// <param name="certificateArn">The ARN of the certificate to detach.</param>
    /// <returns>True if successful, false otherwise.</returns>
    public async Task<bool> DetachThingPrincipalAsync(string thingName, string
certificateArn)
    {
        try
```

```
        {
            var request = new DetachThingPrincipalRequest
            {
                ThingName = thingName,
                Principal = certificateArn
            };

            await _amazonIoT.DetachThingPrincipalAsync(request);
            _logger.LogInformation($"Detached certificate {certificateArn} from
Thing {thingName}");
            return true;
        }
        catch (Amazon.IoT.Model.ResourceNotFoundException ex)
        {
            _logger.LogError($"Cannot detach certificate - resource not found:
{ex.Message}");
            return false;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't detach certificate from Thing. Here's why:
{ex.Message}");
            return false;
        }
    }

    /// <summary>
    /// Deletes an IoT certificate.
    /// </summary>
    /// <param name="certificateId">The ID of the certificate to delete.</param>
    /// <returns>True if successful, false otherwise.</returns>
    public async Task<bool> DeleteCertificateAsync(string certificateId)
    {
        try
        {
            // First, update the certificate to inactive state
            var updateRequest = new UpdateCertificateRequest
            {
                CertificateId = certificateId,
                NewStatus = CertificateStatus.INACTIVE
            };
            await _amazonIoT.UpdateCertificateAsync(updateRequest);

            // Then delete the certificate
```

```
            var deleteRequest = new DeleteCertificateRequest
            {
                CertificateId = certificateId
            };

            await _amazonIoT.DeleteCertificateAsync(deleteRequest);
            _logger.LogInformation($"Deleted certificate {certificateId}");
            return true;
        }
        catch (Amazon.IoT.Model.ResourceNotFoundException ex)
        {
            _logger.LogError($"Cannot delete certificate - resource not found:
{ex.Message}");
            return false;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't delete certificate. Here's why:
{ex.Message}");
            return false;
        }
    }

    /// <summary>
    /// Deletes an IoT Thing.
    /// </summary>
    /// <param name="thingName">The name of the Thing to delete.</param>
    /// <returns>True if successful, false otherwise.</returns>
    public async Task<bool> DeleteThingAsync(string thingName)
    {
        try
        {
            var request = new DeleteThingRequest
            {
                ThingName = thingName
            };

            await _amazonIoT.DeleteThingAsync(request);
            _logger.LogInformation($"Deleted Thing {thingName}");
            return true;
        }
        catch (Amazon.IoT.Model.ResourceNotFoundException ex)
        {
```

```
            _logger.LogError($"Cannot delete Thing - resource not found:
{ex.Message}");
            return false;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't delete Thing. Here's why: {ex.Message}");
            return false;
        }
    }

    /// <summary>
    /// Lists IoT Things with pagination support.
    /// </summary>
    /// <returns>List of Things, or empty list if listing failed.</returns>
    public async Task<List<ThingAttribute>> ListThingsAsync()
    {
        try
        {
            // Use pages of 10.
            var request = new ListThingsRequest()
            {
                MaxResults = 10
            };
            var response = await _amazonIoT.ListThingsAsync(request);

            // Since there is not a built-in paginator, use the NextMarker to
paginate.
            bool hasMoreResults = true;

            var things = new List<ThingAttribute>();
            while (hasMoreResults)
            {
                things.AddRange(response.Things);

                // If NextMarker is not null, there are more results. Get the next
page of results.
                if (!String.IsNullOrEmpty(response.NextMarker))
                {
                    request.Marker = response.NextMarker;
                    response = await _amazonIoT.ListThingsAsync(request);
                }
                else
                    hasMoreResults = false;
```

```
            }

            _logger.LogInformation($"Retrieved {things.Count} Things");
            return things;
        }
        catch (Amazon.IoT.Model.ThrottlingException ex)
        {
            _logger.LogWarning($"Request throttled, please try again later:
 {ex.Message}");
            return new List<ThingAttribute>();
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't list Things. Here's why: {ex.Message}");
            return new List<ThingAttribute>();
        }
    }

}
```

- API 세부 정보는 AWS SDK for .NET API 참조의 다음 주제를 참조하세요.
  - AttachThingPrincipal
  - CreateKeysAndCertificate
  - CreateThing
  - CreateTopicRule
  - DeleteCertificate
  - DeleteThing
  - DeleteTopicRule
  - DescribeEndpoint
  - DescribeThing
  - DetachThingPrincipal
  - ListCertificates
  - ListThings
  - SearchIndex
  - UpdateIndexingConfiguration
  - UpdateThing

작업

`AttachThingPrincipal`

다음 코드 예시는 `AttachThingPrincipal`의 사용 방법을 보여 줍니다.

SDK for .NET (v4)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
   /// <summary>
   /// Attaches a certificate to an IoT Thing.
   /// </summary>
   /// <param name="thingName">The name of the Thing.</param>
   /// <param name="certificateArn">The ARN of the certificate to attach.</param>
   /// <returns>True if successful, false otherwise.</returns>
   public async Task<bool> AttachThingPrincipalAsync(string thingName, string
certificateArn)
   {
       try
       {
           var request = new AttachThingPrincipalRequest
           {
               ThingName = thingName,
               Principal = certificateArn
           };

           await _amazonIoT.AttachThingPrincipalAsync(request);
           _logger.LogInformation($"Attached certificate {certificateArn} to Thing
{thingName}");
           return true;
       }
       catch (Amazon.IoT.Model.ResourceNotFoundException ex)
       {
           _logger.LogError($"Cannot attach certificate - resource not found:
{ex.Message}");
           return false;
       }
```

```
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't attach certificate to Thing. Here's why:
{ex.Message}");
            return false;
        }
    }
```

- API 세부 정보는 AWS SDK for .NET API 참조의 AttachThingPrincipal을 참조하세요.

`CreateKeysAndCertificate`

다음 코드 예시는 `CreateKeysAndCertificate`의 사용 방법을 보여 줍니다.

SDK for .NET (v4)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
    /// <summary>
    /// Creates a device certificate for AWS IoT.
    /// </summary>
    /// <returns>The certificate details including ARN and certificate PEM, or null
if creation failed.</returns>
    public async Task<(string CertificateArn, string CertificatePem, string
CertificateId)?> CreateKeysAndCertificateAsync()
    {
        try
        {
            var request = new CreateKeysAndCertificateRequest
            {
                SetAsActive = true
            };

            var response = await _amazonIoT.CreateKeysAndCertificateAsync(request);
            _logger.LogInformation($"Created certificate with ARN
{response.CertificateArn}");
```

```
            return (response.CertificateArn, response.CertificatePem,
response.CertificateId);
        }
        catch (Amazon.IoT.Model.ThrottlingException ex)
        {
            _logger.LogWarning($"Request throttled, please try again later:
{ex.Message}");
            return null;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't create certificate. Here's why:
{ex.Message}");
            return null;
        }
    }
```

- API 세부 정보는 AWS SDK for .NET API 참조에서 CreateKeysAndCertificate를 참조하세요.

`CreateThing`

다음 코드 예시는 `CreateThing`의 사용 방법을 보여 줍니다.

SDK for .NET (v4)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
    /// <summary>
    /// Creates an AWS IoT Thing.
    /// </summary>
    /// <param name="thingName">The name of the Thing to create.</param>
    /// <returns>The ARN of the Thing created, or null if creation failed.</returns>
    public async Task<string?> CreateThingAsync(string thingName)
    {
        try
        {
            var request = new CreateThingRequest
```

```
            {
                ThingName = thingName
            };

            var response = await _amazonIoT.CreateThingAsync(request);
            _logger.LogInformation($"Created Thing {thingName} with ARN
{response.ThingArn}");
            return response.ThingArn;
        }
        catch (Amazon.IoT.Model.ResourceAlreadyExistsException ex)
        {
            _logger.LogWarning($"Thing {thingName} already exists: {ex.Message}");
            return null;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't create Thing {thingName}. Here's why:
{ex.Message}");
            return null;
        }
    }
```

- API 세부 정보는 AWS SDK for .NET API 참조의 CreateThing을 참조하세요.

`CreateTopicRule`

다음 코드 예시는 `CreateTopicRule`의 사용 방법을 보여 줍니다.

SDK for .NET (v4)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
    /// <summary>
    /// Creates an IoT topic rule.
    /// </summary>
    /// <param name="ruleName">The name of the rule.</param>
    /// <param name="snsTopicArn">The ARN of the SNS topic for the action.</param>
```

```
    /// <param name="roleArn">The ARN of the IAM role.</param>
    /// <returns>True if successful, false otherwise.</returns>
    public async Task<bool> CreateTopicRuleAsync(string ruleName, string
snsTopicArn, string roleArn)
    {
        try
        {
            var request = new CreateTopicRuleRequest
            {
                RuleName = ruleName,
                TopicRulePayload = new TopicRulePayload
                {
                    Sql = "SELECT * FROM 'topic/subtopic'",
                    Description = $"Rule created by .NET example: {ruleName}",
                    Actions = new List<Amazon.IoT.Model.Action>
                    {
                        new Amazon.IoT.Model.Action
                        {
                            Sns = new SnsAction
                            {
                                TargetArn = snsTopicArn,
                                RoleArn = roleArn
                            }
                        }
                    },
                    RuleDisabled = false
                }
            };

            await _amazonIoT.CreateTopicRuleAsync(request);
            _logger.LogInformation($"Created IoT rule {ruleName}");
            return true;
        }
        catch (Amazon.IoT.Model.ResourceAlreadyExistsException ex)
        {
            _logger.LogWarning($"Rule {ruleName} already exists: {ex.Message}");
            return false;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't create topic rule. Here's why:
{ex.Message}");
            return false;
        }
```

```
    }
```

- API 세부 정보는 AWS SDK for .NET API 참조의 CreateTopicRule을 참조하세요.

`DeleteCertificate`

다음 코드 예시는 `DeleteCertificate`의 사용 방법을 보여 줍니다.

SDK for .NET (v4)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
/// <summary>
/// Deletes an IoT certificate.
/// </summary>
/// <param name="certificateId">The ID of the certificate to delete.</param>
/// <returns>True if successful, false otherwise.</returns>
public async Task<bool> DeleteCertificateAsync(string certificateId)
{
    try
    {
        // First, update the certificate to inactive state
        var updateRequest = new UpdateCertificateRequest
        {
            CertificateId = certificateId,
            NewStatus = CertificateStatus.INACTIVE
        };
        await _amazonIoT.UpdateCertificateAsync(updateRequest);

        // Then delete the certificate
        var deleteRequest = new DeleteCertificateRequest
        {
            CertificateId = certificateId
        };

        await _amazonIoT.DeleteCertificateAsync(deleteRequest);
        _logger.LogInformation($"Deleted certificate {certificateId}");
```

```
            return true;
        }
        catch (Amazon.IoT.Model.ResourceNotFoundException ex)
        {
            _logger.LogError($"Cannot delete certificate - resource not found:
{ex.Message}");
            return false;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't delete certificate. Here's why:
{ex.Message}");
            return false;
        }
    }
```

- API 세부 정보는 AWS SDK for .NET API 참조의 DeleteCertificate를 참조하세요.

`DeleteThing`

다음 코드 예시는 `DeleteThing`의 사용 방법을 보여 줍니다.

SDK for .NET (v4)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
    /// <summary>
    /// Deletes an IoT Thing.
    /// </summary>
    /// <param name="thingName">The name of the Thing to delete.</param>
    /// <returns>True if successful, false otherwise.</returns>
    public async Task<bool> DeleteThingAsync(string thingName)
    {
        try
        {
            var request = new DeleteThingRequest
            {
```

```
                ThingName = thingName
            };

            await _amazonIoT.DeleteThingAsync(request);
            _logger.LogInformation($"Deleted Thing {thingName}");
            return true;
        }
        catch (Amazon.IoT.Model.ResourceNotFoundException ex)
        {
            _logger.LogError($"Cannot delete Thing - resource not found:
{ex.Message}");
            return false;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't delete Thing. Here's why: {ex.Message}");
            return false;
        }
    }
```

- API 세부 정보는 AWS SDK for .NET API 참조의 DeleteThing을 참조하세요.

`DescribeEndpoint`

다음 코드 예시는 `DescribeEndpoint`의 사용 방법을 보여 줍니다.

SDK for .NET (v4)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
    /// <summary>
    /// Gets the AWS IoT endpoint URL.
    /// </summary>
    /// <returns>The endpoint URL, or null if retrieval failed.</returns>
    public async Task<string?> DescribeEndpointAsync()
    {
        try
```

```
        {
            var request = new DescribeEndpointRequest
            {
                EndpointType = "iot:Data-ATS"
            };

            var response = await _amazonIoT.DescribeEndpointAsync(request);
            _logger.LogInformation($"Retrieved endpoint:
{response.EndpointAddress}");
            return response.EndpointAddress;
        }
        catch (Amazon.IoT.Model.ThrottlingException ex)
        {
            _logger.LogWarning($"Request throttled, please try again later:
{ex.Message}");
            return null;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't describe endpoint. Here's why:
{ex.Message}");
            return null;
        }
    }
```

- API 세부 정보는 AWS SDK for .NET API 참조의 DescribeEndpoint를 참조하세요.

`DetachThingPrincipal`

다음 코드 예시는 `DetachThingPrincipal`의 사용 방법을 보여 줍니다.

SDK for .NET (v4)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
    /// <summary>
```

```
    /// Detaches a certificate from an IoT Thing.
    /// </summary>
    /// <param name="thingName">The name of the Thing.</param>
    /// <param name="certificateArn">The ARN of the certificate to detach.</param>
    /// <returns>True if successful, false otherwise.</returns>
    public async Task<bool> DetachThingPrincipalAsync(string thingName, string
certificateArn)
    {
        try
        {
            var request = new DetachThingPrincipalRequest
            {
                ThingName = thingName,
                Principal = certificateArn
            };

            await _amazonIoT.DetachThingPrincipalAsync(request);
            _logger.LogInformation($"Detached certificate {certificateArn} from
Thing {thingName}");
            return true;
        }
        catch (Amazon.IoT.Model.ResourceNotFoundException ex)
        {
            _logger.LogError($"Cannot detach certificate - resource not found:
{ex.Message}");
            return false;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't detach certificate from Thing. Here's why:
{ex.Message}");
            return false;
        }
    }
```

- API 세부 정보는 AWS SDK for .NET API 참조의 DetachThingPrincipal을 참조하세요.

`ListCertificates`

다음 코드 예시는 `ListCertificates`의 사용 방법을 보여 줍니다.

SDK for .NET (v4)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
    /// <summary>
    /// Lists all certificates associated with the account.
    /// </summary>
    /// <returns>List of certificate information, or empty list if listing failed.</
returns>
    public async Task<List<Certificate>> ListCertificatesAsync()
    {
        try
        {
            var request = new ListCertificatesRequest();
            var response = await _amazonIoT.ListCertificatesAsync(request);

            _logger.LogInformation($"Retrieved {response.Certificates.Count}
 certificates");
            return response.Certificates;
        }
        catch (Amazon.IoT.Model.ThrottlingException ex)
        {
            _logger.LogWarning($"Request throttled, please try again later:
 {ex.Message}");
            return new List<Certificate>();
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't list certificates. Here's why:
 {ex.Message}");
            return new List<Certificate>();
        }
    }
```

- API 세부 정보는 AWS SDK for .NET API 참조의 ListCertificates를 참조하세요.

`ListThings`

다음 코드 예시는 `ListThings`의 사용 방법을 보여 줍니다.

SDK for .NET (v4)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
    /// <summary>
    /// Lists IoT Things with pagination support.
    /// </summary>
    /// <returns>List of Things, or empty list if listing failed.</returns>
    public async Task<List<ThingAttribute>> ListThingsAsync()
    {
        try
        {
            // Use pages of 10.
            var request = new ListThingsRequest()
            {
                MaxResults = 10
            };
            var response = await _amazonIoT.ListThingsAsync(request);

            // Since there is not a built-in paginator, use the NextMarker to
paginate.
            bool hasMoreResults = true;

            var things = new List<ThingAttribute>();
            while (hasMoreResults)
            {
                things.AddRange(response.Things);

                // If NextMarker is not null, there are more results. Get the next
page of results.
                if (!String.IsNullOrEmpty(response.NextMarker))
                {
                    request.Marker = response.NextMarker;
                    response = await _amazonIoT.ListThingsAsync(request);
                }
```

```
                else
                    hasMoreResults = false;
            }

            _logger.LogInformation($"Retrieved {things.Count} Things");
            return things;
        }
        catch (Amazon.IoT.Model.ThrottlingException ex)
        {
            _logger.LogWarning($"Request throttled, please try again later:
{ex.Message}");
            return new List<ThingAttribute>();
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't list Things. Here's why: {ex.Message}");
            return new List<ThingAttribute>();
        }
    }
```

- API 세부 정보는 API 참조의 ListThingsAWS SDK for .NET 를 참조하세요.

`SearchIndex`

다음 코드 예시는 `SearchIndex`의 사용 방법을 보여 줍니다.

SDK for .NET (v4)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
    /// <summary>
    /// Searches for IoT Things using the search index.
    /// </summary>
    /// <param name="queryString">The search query string.</param>
    /// <returns>List of Things that match the search criteria, or empty list if
search failed.</returns>
    public async Task<List<ThingDocument>> SearchIndexAsync(string queryString)
```

```
    {
        try
        {
            // First, try to perform the search
            var request = new SearchIndexRequest
            {
                QueryString = queryString
            };

            var response = await _amazonIoT.SearchIndexAsync(request);
            _logger.LogInformation($"Search found {response.Things.Count} Things");
            return response.Things;
        }
        catch (Amazon.IoT.Model.IndexNotReadyException ex)
        {
            _logger.LogWarning($"Search index not ready, setting up indexing
configuration: {ex.Message}");
            return await SetupIndexAndRetrySearchAsync(queryString);
        }
        catch (Amazon.IoT.Model.ResourceNotFoundException ex) when
(ex.Message.Contains("index") || ex.Message.Contains("Index"))
        {
            _logger.LogWarning($"Search index not configured, setting up indexing
configuration: {ex.Message}");
            return await SetupIndexAndRetrySearchAsync(queryString);
        }
        catch (Amazon.IoT.Model.ThrottlingException ex)
        {
            _logger.LogWarning($"Request throttled, please try again later:
{ex.Message}");
            return new List<ThingDocument>();
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't search index. Here's why: {ex.Message}");
            return new List<ThingDocument>();
        }
    }

    /// <summary>
    /// Sets up the indexing configuration and retries the search after waiting for
the index to be ready.
    /// </summary>
    /// <param name="queryString">The search query string.</param>
```

```
   /// <returns>List of Things that match the search criteria, or empty list if
setup/search failed.</returns>
   private async Task<List<ThingDocument>> SetupIndexAndRetrySearchAsync(string
queryString)
   {
       try
       {
           // Update indexing configuration to REGISTRY mode
           _logger.LogInformation("Setting up IoT search indexing
configuration...");
           await _amazonIoT.UpdateIndexingConfigurationAsync(
               new UpdateIndexingConfigurationRequest()
               {
                   ThingIndexingConfiguration = new ThingIndexingConfiguration()
                   {
                       ThingIndexingMode = ThingIndexingMode.REGISTRY
                   }
               });

           _logger.LogInformation("Indexing configuration updated. Waiting for
index to be ready...");

           // Wait for the index to be set up - this can take some time
           const int maxRetries = 10;
           const int retryDelaySeconds = 10;

           for (int attempt = 1; attempt <= maxRetries; attempt++)
           {
               try
               {
                   _logger.LogInformation($"Waiting for index to be ready (attempt
{attempt}/{maxRetries})...");
                   await Task.Delay(TimeSpan.FromSeconds(retryDelaySeconds));

                   // Try to get the current indexing configuration to see if it's
ready
                   var configResponse = await
_amazonIoT.GetIndexingConfigurationAsync(new GetIndexingConfigurationRequest());
                   if (configResponse.ThingIndexingConfiguration?.ThingIndexingMode
== ThingIndexingMode.REGISTRY)
                   {
                       // Try the search again
                       var request = new SearchIndexRequest
                       {
```

```
                        QueryString = queryString
                    };

                    var response = await _amazonIoT.SearchIndexAsync(request);
                    _logger.LogInformation($"Search found
{response.Things.Count} Things after index setup");
                    return response.Things;
                }
            }
            catch (Amazon.IoT.Model.IndexNotReadyException)
            {
                // Index still not ready, continue waiting
                _logger.LogInformation("Index still not ready, continuing to
wait...");
                continue;
            }
            catch (Amazon.IoT.Model.InvalidRequestException ex) when
(ex.Message.Contains("index") || ex.Message.Contains("Index"))
            {
                // Index still not ready, continue waiting
                _logger.LogInformation("Index still not ready, continuing to
wait...");
                continue;
            }
        }

        _logger.LogWarning("Timeout waiting for search index to be ready after
configuration update");
        return new List<ThingDocument>();
    }
    catch (Exception ex)
    {
        _logger.LogError($"Couldn't set up search index configuration. Here's
why: {ex.Message}");
        return new List<ThingDocument>();
    }
}
```

- API 세부 정보는 AWS SDK for .NET API 참조의 SearchIndex를 참조하세요.

UpdateThing

다음 코드 예시는 UpdateThing의 사용 방법을 보여 줍니다.

SDK for .NET (v4)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
   /// <summary>
   /// Updates an IoT Thing with attributes.
   /// </summary>
   /// <param name="thingName">The name of the Thing to update.</param>
   /// <param name="attributes">Dictionary of attributes to add.</param>
   /// <returns>True if successful, false otherwise.</returns>
   public async Task<bool> UpdateThingAsync(string thingName, Dictionary<string,
string> attributes)
   {
       try
       {
           var request = new UpdateThingRequest
           {
               ThingName = thingName,
               AttributePayload = new AttributePayload
               {
                   Attributes = attributes,
                   Merge = true
               }
           };

           await _amazonIoT.UpdateThingAsync(request);
           _logger.LogInformation($"Updated Thing {thingName} with attributes");
           return true;
       }
       catch (Amazon.IoT.Model.ResourceNotFoundException ex)
       {
           _logger.LogError($"Cannot update Thing - resource not found:
{ex.Message}");
           return false;
       }
```

```
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't update Thing attributes. Here's why:
{ex.Message}");
            return false;
        }
    }
```

- API 세부 정보는 AWS SDK for .NET API 참조의 UpdateThing을 참조하세요.

AWS IoT data SDK for .NET (v4)를 사용한 예제

다음 코드 예제에서는 AWS SDK for .NET (v4)를와 함께 사용하여 작업을 수행하고 일반적인 시나리오를 구현하는 방법을 보여줍니다 AWS IoT data.

작업은 대규모 프로그램에서 발췌한 코드이며 컨텍스트에 맞춰 실행해야 합니다. 작업은 개별 서비스 함수를 직접적으로 호출하는 방법을 보여주며 관련 시나리오의 컨텍스트에 맞는 작업을 볼 수 있습니다.

각 예시에는 전체 소스 코드에 대한 링크가 포함되어 있으며, 여기에서 컨텍스트에 맞춰 코드를 설정하고 실행하는 방법에 대한 지침을 찾을 수 있습니다.

주제

- 작업

작업

`GetThingShadow`

다음 코드 예시는 `GetThingShadow`의 사용 방법을 보여 줍니다.

SDK for .NET (v4)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
    /// <summary>
    /// Gets the Thing's shadow information.
    /// </summary>
    /// <param name="thingName">The name of the Thing.</param>
    /// <returns>The shadow data as a string, or null if retrieval failed.</returns>
    public async Task<string?> GetThingShadowAsync(string thingName)
    {
        try
        {
            var request = new GetThingShadowRequest
            {
                ThingName = thingName
            };

            var response = await _amazonIotData.GetThingShadowAsync(request);
            using var reader = new StreamReader(response.Payload);
            var shadowData = await reader.ReadToEndAsync();

            _logger.LogInformation($"Retrieved shadow for Thing {thingName}");
            return shadowData;
        }
        catch (Amazon.IotData.Model.ResourceNotFoundException ex)
        {
            _logger.LogError($"Cannot get Thing shadow - resource not found:
{ex.Message}");
            return null;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't get Thing shadow. Here's why:
{ex.Message}");
            return null;
        }
    }
```

- API 세부 정보는 AWS SDK for .NET API 참조의 GetThingShadow를 참조하세요.

`UpdateThingShadow`

다음 코드 예시는 `UpdateThingShadow`의 사용 방법을 보여 줍니다.

SDK for .NET (v4)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
   /// <summary>
   /// Updates the Thing's shadow with new state information.
   /// </summary>
   /// <param name="thingName">The name of the Thing.</param>
   /// <param name="shadowPayload">The shadow payload in JSON format.</param>
   /// <returns>True if successful, false otherwise.</returns>
   public async Task<bool> UpdateThingShadowAsync(string thingName, string
shadowPayload)
   {
       try
       {
           var request = new UpdateThingShadowRequest
           {
               ThingName = thingName,
               Payload = new
MemoryStream(System.Text.Encoding.UTF8.GetBytes(shadowPayload))
           };

           await _amazonIotData.UpdateThingShadowAsync(request);
           _logger.LogInformation($"Updated shadow for Thing {thingName}");
           return true;
       }
       catch (Amazon.IotData.Model.ResourceNotFoundException ex)
       {
           _logger.LogError($"Cannot update Thing shadow - resource not found:
{ex.Message}");
           return false;
       }
       catch (Exception ex)
       {
           _logger.LogError($"Couldn't update Thing shadow. Here's why:
{ex.Message}");
           return false;
       }
```

```
    }
```

- API 세부 정보는 AWS SDK for .NET API 참조의 UpdateThingShadow를 참조하세요.

SDK for .NET (v4)를 사용한 Amazon Redshift 예제

다음 코드 예제에서는 Amazon Redshift에서 AWS SDK for .NET (v4)를 사용하여 작업을 수행하고 일반적인 시나리오를 구현하는 방법을 보여줍니다.

기본 사항은 서비스 내에서 필수 작업을 수행하는 방법을 보여주는 코드 예제입니다.

작업은 대규모 프로그램에서 발췌한 코드이며 컨텍스트에 맞춰 실행해야 합니다. 작업은 개별 서비스 함수를 직접적으로 호출하는 방법을 보여주며 관련 시나리오의 컨텍스트에 맞는 작업을 볼 수 있습니다.

각 예시에는 전체 소스 코드에 대한 링크가 포함되어 있으며, 여기에서 컨텍스트에 맞춰 코드를 설정하고 실행하는 방법에 대한 지침을 찾을 수 있습니다.

주제

- 시작하기
- 기본 사항
- 작업

시작하기

Hello Amazon Redshift

다음 코드 예시에서는 Amazon Redshift 사용을 시작하는 방법을 보여줍니다.

SDK for .NET (v4)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
    /// <summary>
```

```
    /// Main method to run the Hello Amazon Redshift example.
    /// </summary>
    /// <param name="args">Command line arguments (not used).</param>
    public static async Task Main(string[] args)
    {
        var redshiftClient = new AmazonRedshiftClient();

        Console.WriteLine("Hello, Amazon Redshift! Let's list available clusters:");

        var clusters = new List<Cluster>();

        try
        {
            // Use pagination to retrieve all clusters.
            var clustersPaginator = redshiftClient.Paginators.DescribeClusters(new
DescribeClustersRequest());

            await foreach (var response in clustersPaginator.Responses)
            {
                if (response.Clusters != null)
                    clusters.AddRange(response.Clusters);
            }

            Console.WriteLine($"{clusters.Count} cluster(s) retrieved.");

            foreach (var cluster in clusters)
            {
                Console.WriteLine($"\t{cluster.ClusterIdentifier} (Status:
{cluster.ClusterStatus})");
            }
        }
        catch (AmazonRedshiftException ex)
        {
            Console.WriteLine($"Couldn't list clusters. Here's why: {ex.Message}");
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred: {ex.Message}");
        }
    }
```

- API에 대한 세부 정보는 AWS SDK for .NET API 참조의 DescribeClusters를 참조하세요.

기본 사항

기본 사항 알아보기

다음 코드 예제에서는 다음과 같은 작업을 수행하는 방법을 보여줍니다.

- Redshift 클러스터를 생성합니다.
- 클러스터의 데이터베이스를 나열합니다.
- Movies라는 테이블을 생성합니다.
- Movies 테이블을 채웁니다.
- Movies 테이블을 연도별로 쿼리합니다.
- Redshift 클러스터를 수정합니다.
- Amazon Redshift 클러스터를 삭제합니다.

SDK for .NET (v4)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

Redshift 래퍼 클래스를 생성하여 작업을 관리합니다.

```
/// <summary>
/// Wrapper class for Amazon Redshift operations.
/// </summary>
public class RedshiftWrapper
{
    private readonly IAmazonRedshift _redshiftClient;
    private readonly IAmazonRedshiftDataAPIService _redshiftDataClient;

    /// <summary>
    /// Constructor for RedshiftWrapper.
    /// </summary>
    /// <param name="redshiftClient">Amazon Redshift client.</param>
    /// <param name="redshiftDataClient">Amazon Redshift Data API client.</param>
    public RedshiftWrapper(IAmazonRedshift redshiftClient,
 IAmazonRedshiftDataAPIService redshiftDataClient)
```

```
    {
        _redshiftClient = redshiftClient;
        _redshiftDataClient = redshiftDataClient;
    }

    /// <summary>
    /// Create a new Amazon Redshift cluster.
    /// </summary>
    /// <param name="clusterIdentifier">The identifier for the cluster.</param>
    /// <param name="databaseName">The name of the database.</param>
    /// <param name="masterUsername">The master username.</param>
    /// <param name="masterUserPassword">The master user password.</param>
    /// <param name="nodeType">The node type for the cluster.</param>
    /// <returns>The cluster that was created.</returns>
    public async Task<Cluster> CreateClusterAsync(string clusterIdentifier, string
databaseName,
        string masterUsername, string masterUserPassword, string nodeType =
"ra3.large")
    {
        try
        {
            var request = new CreateClusterRequest
            {
                ClusterIdentifier = clusterIdentifier,
                DBName = databaseName,
                MasterUsername = masterUsername,
                MasterUserPassword = masterUserPassword,
                NodeType = nodeType,
                NumberOfNodes = 1,
                ClusterType = "single-node"
            };

            var response = await _redshiftClient.CreateClusterAsync(request);
            Console.WriteLine($"Created cluster {clusterIdentifier}");
            return response.Cluster;
        }
        catch (ClusterAlreadyExistsException ex)
        {
            Console.WriteLine($"Cluster already exists: {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Couldn't create cluster. Here's why: {ex.Message}");
```

```
            throw;
        }
    }

    /// <summary>
    /// Describe Amazon Redshift clusters.
    /// </summary>
    /// <param name="clusterIdentifier">Optional cluster identifier to describe a
specific cluster.</param>
    /// <returns>A list of clusters.</returns>
    public async Task<List<Cluster>> DescribeClustersAsync(string? clusterIdentifier
= null)
    {
        try
        {
            var clusters = new List<Cluster>();
            var request = new DescribeClustersRequest();
            if (!string.IsNullOrEmpty(clusterIdentifier))
            {
                request.ClusterIdentifier = clusterIdentifier;
            }

            var clustersPaginator =
_redshiftClient.Paginators.DescribeClusters(request);
            await foreach (var response in clustersPaginator.Responses)
            {
                if (response.Clusters != null)
                    clusters.AddRange(response.Clusters);
            }

            Console.WriteLine($"{clusters.Count} cluster(s) retrieved.");
            foreach (var cluster in clusters)
            {
                Console.WriteLine($"\t{cluster.ClusterIdentifier} (Status:
{cluster.ClusterStatus})");
            }

            return clusters;
        }
        catch (ClusterNotFoundException ex)
        {
            Console.WriteLine($"Cluster {clusterIdentifier} not found:
{ex.Message}");
            throw;
```

```
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Couldn't describe clusters. Here's why:
 {ex.Message}");
            throw;
        }
    }

    /// <summary>
    /// Modify an Amazon Redshift cluster.
    /// </summary>
    /// <param name="clusterIdentifier">The identifier for the cluster.</param>
    /// <param name="preferredMaintenanceWindow">The preferred maintenance window.</
param>
    /// <returns>True if successful.</returns>
    public async Task<bool> ModifyClusterAsync(string clusterIdentifier, string
 preferredMaintenanceWindow)
    {
        try
        {
            var request = new ModifyClusterRequest
            {
                ClusterIdentifier = clusterIdentifier,
                PreferredMaintenanceWindow = preferredMaintenanceWindow
            };

            var response = await _redshiftClient.ModifyClusterAsync(request);
            Console.WriteLine($"The modified cluster was successfully modified and
 has {response.Cluster.PreferredMaintenanceWindow} as the maintenance window");
            return true;
        }
        catch (ClusterNotFoundException ex)
        {
            Console.WriteLine($"Cluster {clusterIdentifier} not found:
 {ex.Message}");
            return false;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Couldn't modify cluster. Here's why: {ex.Message}");
            return false;
        }
    }
```

```
    /// <summary>
    /// Delete an Amazon Redshift cluster without a final snapshot.
    /// </summary>
    /// <param name="clusterIdentifier">The identifier for the cluster.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DeleteClusterWithoutSnapshotAsync(string
clusterIdentifier)
    {
        try
        {
            var request = new DeleteClusterRequest
            {
                ClusterIdentifier = clusterIdentifier,
                SkipFinalClusterSnapshot = true
            };

            var response = await _redshiftClient.DeleteClusterAsync(request);
            Console.WriteLine($"The {clusterIdentifier} was deleted");
            return true;
        }
        catch (ClusterNotFoundException ex)
        {
            Console.WriteLine($"Cluster not found: {ex.Message}");
            return false;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Couldn't delete cluster. Here's why: {ex.Message}");
            return false;
        }
    }

    /// <summary>
    /// List databases in a Redshift cluster.
    /// </summary>
    /// <param name="clusterIdentifier">The cluster identifier.</param>
    /// <param name="dbUser">The database user.</param>
    /// <param name="dbUser">The database name for authentication.</param>
    /// <returns>A list of database names.</returns>
    public async Task<List<string>> ListDatabasesAsync(string clusterIdentifier,
string dbUser, string databaseName)
    {
        try
```

```
        {
            var request = new ListDatabasesRequest
            {
                ClusterIdentifier = clusterIdentifier,
                DbUser = dbUser,
                Database = databaseName
            };

            var response = await _redshiftDataClient.ListDatabasesAsync(request);
            var databases = new List<string>();

            foreach (var database in response.Databases)
            {
                Console.WriteLine($"The database name is : {database}");
                databases.Add(database);
            }

            return databases;
        }
        catch (Amazon.RedshiftDataAPIService.Model.ValidationException ex)
        {
            Console.WriteLine($"Validation error: {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Couldn't list databases. Here's why: {ex.Message}");
            throw;
        }
    }

    /// <summary>
    /// Create a table in the Redshift database.
    /// </summary>
    /// <param name="clusterIdentifier">The cluster identifier.</param>
    /// <param name="database">The database name.</param>
    /// <param name="dbUser">The database user.</param>
    /// <returns>The statement ID.</returns>
    public async Task<string> CreateTableAsync(string clusterIdentifier, string
database, string dbUser)
    {
        try
        {
            var sqlStatement = @"
```

```
                CREATE TABLE Movies (
                    id INTEGER PRIMARY KEY,
                    title VARCHAR(250) NOT NULL,
                    year INTEGER NOT NULL
                )";

            var request = new ExecuteStatementRequest
            {
                ClusterIdentifier = clusterIdentifier,
                Database = database,
                DbUser = dbUser,
                Sql = sqlStatement
            };

            var response = await _redshiftDataClient.ExecuteStatementAsync(request);
            await WaitForStatementToCompleteAsync(response.Id);
            Console.WriteLine("Table created: Movies");
            return response.Id;
        }
        catch (Amazon.RedshiftDataAPIService.Model.ValidationException ex)
        {
            Console.WriteLine($"Validation error: {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Couldn't create table. Here's why: {ex.Message}");
            throw;
        }
    }

    /// <summary>
    /// Insert a record into the Movies table using parameterized query.
    /// </summary>
    /// <param name="clusterIdentifier">The cluster identifier.</param>
    /// <param name="database">The database name.</param>
    /// <param name="dbUser">The database user.</param>
    /// <param name="id">The movie ID.</param>
    /// <param name="title">The movie title.</param>
    /// <param name="year">The movie year.</param>
    /// <returns>The statement ID.</returns>
    public async Task<string> InsertMovieAsync(string clusterIdentifier, string
database, string dbUser,
        int id, string title, int year)
```

```
    {
        try
        {
            var sqlStatement = "INSERT INTO Movies (id, title, year) VALUES
(:id, :title, :year)";

            var request = new ExecuteStatementRequest
            {
                ClusterIdentifier = clusterIdentifier,
                Database = database,
                DbUser = dbUser,
                Sql = sqlStatement,
                Parameters = new List<SqlParameter>
                {
                    new SqlParameter { Name = "id", Value = id.ToString() },
                    new SqlParameter { Name = "title", Value = title },
                    new SqlParameter { Name = "year", Value = year.ToString() }
                }
            };

            var response = await _redshiftDataClient.ExecuteStatementAsync(request);
            await WaitForStatementToCompleteAsync(response.Id);
            Console.WriteLine($"Inserted: {title} ({year})");
            return response.Id;
        }
        catch (Amazon.RedshiftDataAPIService.Model.ValidationException ex)
        {
            Console.WriteLine($"Validation error: {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Couldn't insert movie. Here's why: {ex.Message}");
            throw;
        }
    }

    /// <summary>
    /// Query movies by year using parameterized query.
    /// </summary>
    /// <param name="clusterIdentifier">The cluster identifier.</param>
    /// <param name="database">The database name.</param>
    /// <param name="dbUser">The database user.</param>
    /// <param name="year">The year to query.</param>
```

```
    /// <returns>A list of movie titles.</returns>
    public async Task<List<string>> QueryMoviesByYearAsync(string clusterIdentifier,
string database,
        string dbUser, int year)
    {
        try
        {
            var sqlStatement = "SELECT title FROM Movies WHERE year = :year";

            var request = new ExecuteStatementRequest
            {
                ClusterIdentifier = clusterIdentifier,
                Database = database,
                DbUser = dbUser,
                Sql = sqlStatement,
                Parameters = new List<SqlParameter>
                {
                    new SqlParameter { Name = "year", Value = year.ToString() }
                }
            };

            var response = await _redshiftDataClient.ExecuteStatementAsync(request);
            Console.WriteLine($"The identifier of the statement is {response.Id}");

            await WaitForStatementToCompleteAsync(response.Id);

            var results = await GetStatementResultAsync(response.Id);
            var movieTitles = new List<string>();

            foreach (var row in results)
            {
                if (row.Count > 0)
                {
                    var title = row[0].StringValue;
                    Console.WriteLine($"The Movie title field is {title}");
                    movieTitles.Add(title);
                }
            }

            return movieTitles;
        }
        catch (Amazon.RedshiftDataAPIService.Model.ValidationException ex)
        {
            Console.WriteLine($"Validation error: {ex.Message}");
```

```
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Couldn't query movies. Here's why: {ex.Message}");
            throw;
        }
    }

    /// <summary>
    /// Describe a statement execution.
    /// </summary>
    /// <param name="statementId">The statement ID.</param>
    /// <returns>The statement description.</returns>
    public async Task<DescribeStatementResponse> DescribeStatementAsync(string
statementId)
    {
        try
        {
            var request = new DescribeStatementRequest
            {
                Id = statementId
            };

            var response = await
_redshiftDataClient.DescribeStatementAsync(request);
            return response;
        }
        catch (Amazon.RedshiftDataAPIService.Model.ResourceNotFoundException ex)
        {
            Console.WriteLine($"Statement not found: {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Couldn't describe statement. Here's why:
{ex.Message}");
            throw;
        }
    }

    /// <summary>
    /// Get the results of a statement execution.
    /// </summary>
```

```
    /// <param name="statementId">The statement ID.</param>
    /// <returns>A list of result rows.</returns>
    public async Task<List<List<Field>>> GetStatementResultAsync(string statementId)
    {
        try
        {
            var request = new GetStatementResultRequest
            {
                Id = statementId
            };

            var response = await
_redshiftDataClient.GetStatementResultAsync(request);
            return response.Records;
        }
        catch (Amazon.RedshiftDataAPIService.Model.ResourceNotFoundException ex)
        {
            Console.WriteLine($"Statement not found: {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Couldn't get statement result. Here's why:
{ex.Message}");
            throw;
        }
    }

    /// <summary>
    /// Wait for a statement to complete execution.
    /// </summary>
    /// <param name="statementId">The statement ID.</param>
    /// <returns>A task representing the asynchronous operation.</returns>
    private async Task WaitForStatementToCompleteAsync(string statementId)
    {
        var status = StatusString.SUBMITTED;
        DescribeStatementResponse? response = null;

        while (status == StatusString.SUBMITTED || status == StatusString.PICKED ||
status == StatusString.STARTED)
        {
            await Task.Delay(1000); // Wait 1 second
            response = await DescribeStatementAsync(statementId);
            status = response.Status;
```

```
            Console.WriteLine($"...{status}");
        }

        if (status == StatusString.FINISHED)
        {
            Console.WriteLine("The statement is finished!");
        }
        else
        {
            var errorMessage = response?.Error ?? "Unknown error";
            Console.WriteLine($"The statement failed with status: {status}");
            Console.WriteLine($"Error message: {errorMessage}");
        }
    }

    /// <summary>
    /// Wait for a cluster to become available.
    /// </summary>
    /// <param name="clusterIdentifier">The cluster identifier.</param>
    /// <returns>A task representing the asynchronous operation.</returns>
    public async Task WaitForClusterAvailableAsync(string clusterIdentifier)
    {
        Console.WriteLine($"Wait until {clusterIdentifier} is available. This may
 take a few minutes.");

        var startTime = DateTime.Now;
        var clusters = await DescribeClustersAsync(clusterIdentifier);

        while (clusters[0].ClusterStatus != "available")
        {
            var elapsed = DateTime.Now - startTime;
            Console.WriteLine($"Elapsed Time: {elapsed:mm\\:ss} - Waiting for
 cluster...");

            await Task.Delay(5000); // Wait 5 seconds
            clusters = await DescribeClustersAsync(clusterIdentifier);
        }

        var totalElapsed = DateTime.Now - startTime;
        Console.WriteLine($"Cluster is available! Total Elapsed Time:
 {totalElapsed:mm\\:ss}");
    }
}
```

Redshift 기본 사항을 시연하는 대화형 시나리오를 실행합니다.

```
/// <summary>
/// Amazon Redshift Getting Started Scenario.
/// </summary>
public class RedshiftBasics
{
    public static bool IsInteractive = true;
    public static RedshiftWrapper? Wrapper = null;
    public static ILogger logger = null!;
    private static readonly string _moviesFilePath = "../../../../../../resources/
sample_files/movies.json";

    /// <summary>
    /// Main method for the Amazon Redshift Getting Started scenario.
    /// </summary>
    /// <param name="args">Command line arguments.</param>
    public static async Task Main(string[] args)
    {
        using var host = Host.CreateDefaultBuilder(args)
            .ConfigureServices((_, services) =>
                services.AddAWSService<IAmazonRedshift>()
                    .AddAWSService<IAmazonRedshiftDataAPIService>()
                    .AddTransient<RedshiftWrapper>()
            )
            .Build();

        logger = LoggerFactory.Create(builder => { builder.AddConsole(); })
            .CreateLogger<RedshiftBasics>();

        Wrapper = host.Services.GetRequiredService<RedshiftWrapper>();

        await RunScenarioAsync();
    }

    /// <summary>
    /// Run the complete Amazon Redshift scenario.
    /// </summary>
    public static async Task RunScenarioAsync()
    {
        // Set all variables to default values
```

```
        string userName = "awsuser";
        string userPassword = "AwsUser1000";
        string clusterIdentifier = "redshift-cluster-movies";
        var databaseName = "dev";
        int recordCount = 50;
        int year = 2013;
        try
        {
            Console.WriteLine(

"================================================================================");
            Console.WriteLine("Welcome to the Amazon Redshift SDK Getting Started
scenario.");
            Console.WriteLine(
                "This .NET program demonstrates how to interact with Amazon Redshift
by using the AWS SDK for .NET.");
            Console.WriteLine("Let's get started...");
            Console.WriteLine(

"================================================================================");

            // Step 1: Get user credentials (if interactive)
            if (IsInteractive)
            {
                Console.WriteLine("Please enter a user name for the cluster (default
is awsuser):");
                var userInput = Console.ReadLine();
                if (!string.IsNullOrEmpty(userInput))
                    userName = userInput;


Console.WriteLine("================================================================================
                Console.WriteLine("Please enter a user password for the cluster
(default is AwsUser1000):");
                var passwordInput = Console.ReadLine();
                if (!string.IsNullOrEmpty(passwordInput))
                    userPassword = passwordInput;


Console.WriteLine("================================================================================

                // Step 2: Get cluster identifier
                Console.WriteLine("Enter a cluster id value (default is redshift-
cluster-movies):");
```

```
                var clusterInput = Console.ReadLine();
                if (!string.IsNullOrEmpty(clusterInput))
                    clusterIdentifier = clusterInput;
            }
            else
            {
                Console.WriteLine($"Using default values: userName={userName},
clusterIdentifier={clusterIdentifier}");
            }

            // Step 3: Create Redshift cluster
            await Wrapper!.CreateClusterAsync(clusterIdentifier, databaseName,
userName, userPassword);

Console.WriteLine("=============================================================================

            // Step 4: Wait for cluster to become available

Console.WriteLine("=============================================================================
            await Wrapper.WaitForClusterAvailableAsync(clusterIdentifier);

Console.WriteLine("=============================================================================

            // Step 5: List databases

Console.WriteLine("=============================================================================
            Console.WriteLine($" When you created {clusterIdentifier}, the dev
database is created by default and used in this scenario.");
            Console.WriteLine(" To create a custom database, you need to have a
CREATEDB privilege.");
            Console.WriteLine(" For more information, see the documentation here:
https://docs.aws.amazon.com/redshift/latest/dg/r_CREATE_DATABASE.html.");
            if (IsInteractive)
            {
                Console.WriteLine("Press Enter to continue...");
                Console.ReadLine();
            }

Console.WriteLine("=============================================================================


Console.WriteLine("=============================================================================
            Console.WriteLine($"List databases in {clusterIdentifier}");
            if (IsInteractive)
```

```
            {
                Console.WriteLine("Press Enter to continue...");
                Console.ReadLine();
            }
            await Wrapper.ListDatabasesAsync(clusterIdentifier, userName,
databaseName);

Console.WriteLine("========================================================================

            // Step 6: Create Movies table

Console.WriteLine("========================================================================
            Console.WriteLine("Now you will create a table named Movies.");
            if (IsInteractive)
            {
                Console.WriteLine("Press Enter to continue...");
                Console.ReadLine();
            }
            await Wrapper.CreateTableAsync(clusterIdentifier, databaseName,
userName);

Console.WriteLine("========================================================================

            // Step 7: Populate the Movies table

Console.WriteLine("========================================================================
            Console.WriteLine("Populate the Movies table using the Movies.json
file.");

            if (IsInteractive)
            {
                Console.WriteLine("Specify the number of records you would like to
add to the Movies Table.");
                Console.WriteLine("Please enter a value between 50 and 200.");
                Console.Write("Enter a value: ");

                var recordCountInput = Console.ReadLine();
                if (int.TryParse(recordCountInput, out var inputCount) && inputCount
is >= 50 and <= 200)
                {
                    recordCount = inputCount;
                }
                else
                {
```

```
                    Console.WriteLine($"Invalid input. Using default value of
{recordCount}.");
                }
            }
            else
            {
                Console.WriteLine($"Using default record count: {recordCount}");
            }

            await PopulateMoviesTableAsync(clusterIdentifier, databaseName,
userName, recordCount);
            Console.WriteLine($"{recordCount} records were added to the Movies
table.");

Console.WriteLine("================================================================================

            // Step 8 & 9: Query movies by year

Console.WriteLine("================================================================================
            Console.WriteLine("Query the Movies table by year. Enter a value between
2012-2014.");

            if (IsInteractive)
            {
                Console.Write("Enter a year: ");
                var yearInput = Console.ReadLine();
                if (int.TryParse(yearInput, out var inputYear) && inputYear is >=
2012 and <= 2014)
                {
                    year = inputYear;
                }
                else
                {
                    Console.WriteLine($"Invalid input. Using default value of
{year}.");
                }
            }
            else
            {
                Console.WriteLine($"Using default year: {year}");
            }

            await Wrapper.QueryMoviesByYearAsync(clusterIdentifier, databaseName,
userName, year);
```

```
 Console.WriteLine("=============================================================================

            // Step 10: Modify the cluster

 Console.WriteLine("=============================================================================
            Console.WriteLine("Now you will modify the Redshift cluster.");
            if (IsInteractive)
            {
                Console.WriteLine("Press Enter to continue...");
                Console.ReadLine();
            }
            await Wrapper.ModifyClusterAsync(clusterIdentifier, "wed:07:30-
wed:08:00");

 Console.WriteLine("=============================================================================

            // Step 11 & 12: Delete cluster confirmation

 Console.WriteLine("=============================================================================
            if (IsInteractive)
            {
                Console.WriteLine("Would you like to delete the Amazon Redshift
 cluster? (y/n)");
                var deleteResponse = Console.ReadLine();
                if (deleteResponse?.ToLower() == "y")
                {
                    await
 Wrapper.DeleteClusterWithoutSnapshotAsync(clusterIdentifier);
                }
            }
            else
            {
                Console.WriteLine("Deleting the Amazon Redshift cluster...");
                await Wrapper.DeleteClusterWithoutSnapshotAsync(clusterIdentifier);
            }

 Console.WriteLine("=============================================================================


 Console.WriteLine("=============================================================================
            Console.WriteLine("This concludes the Amazon Redshift SDK Getting
 Started scenario.");
```

```
Console.WriteLine("=============================================================================
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred during the scenario:
{ex.Message}");
            Console.WriteLine("Deleting the Amazon Redshift cluster...");
            await Wrapper!.DeleteClusterWithoutSnapshotAsync(clusterIdentifier);
            throw;
        }
    }

    /// <summary>
    /// Populate the Movies table with data from the JSON file.
    /// </summary>
    /// <param name="clusterIdentifier">The cluster identifier.</param>
    /// <param name="database">The database name.</param>
    /// <param name="dbUser">The database user.</param>
    /// <param name="recordCount">Number of records to insert.</param>
    private static async Task PopulateMoviesTableAsync(string clusterIdentifier,
string database, string dbUser, int recordCount)
    {
        if (!File.Exists(_moviesFilePath))
        {
            throw new FileNotFoundException($"Required movies data file not found
at: {_moviesFilePath}");
        }

        var jsonContent = await File.ReadAllTextAsync(_moviesFilePath);
        var options = new JsonSerializerOptions
        {
            PropertyNameCaseInsensitive = true
        };
        var movies = JsonSerializer.Deserialize<List<Movie>>(jsonContent, options);

        if (movies == null || movies.Count == 0)
        {
            throw new InvalidOperationException("Failed to parse movies JSON file or
file is empty.");
        }

        var insertCount = Math.Min(recordCount, movies.Count);
```

```
        for (int i = 0; i < insertCount; i++)
        {
            var movie = movies[i];
            await Wrapper!.InsertMovieAsync(clusterIdentifier, database, dbUser, i,
 movie.Title, movie.Year);
        }
    }

    /// <summary>
    /// Movie data model.
    /// </summary>
    private class Movie
    {
        public string Title { get; set; } = string.Empty;
        public int Year { get; set; }
    }
}
```

- API 세부 정보는 AWS SDK for .NET API 참조의 다음 주제를 참조하세요.
  - CreateCluster
  - DescribeClusters
  - DescribeStatement
  - ExecuteStatement
  - GetStatementResult
  - ListDatabasesPaginator
  - ModifyCluster

작업

`CreateCluster`

다음 코드 예시는 `CreateCluster`의 사용 방법을 보여 줍니다.

SDK for .NET (v4)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
    /// <summary>
    /// Create a new Amazon Redshift cluster.
    /// </summary>
    /// <param name="clusterIdentifier">The identifier for the cluster.</param>
    /// <param name="databaseName">The name of the database.</param>
    /// <param name="masterUsername">The master username.</param>
    /// <param name="masterUserPassword">The master user password.</param>
    /// <param name="nodeType">The node type for the cluster.</param>
    /// <returns>The cluster that was created.</returns>
    public async Task<Cluster> CreateClusterAsync(string clusterIdentifier, string
databaseName,
        string masterUsername, string masterUserPassword, string nodeType =
"ra3.large")
    {
        try
        {
            var request = new CreateClusterRequest
            {
                ClusterIdentifier = clusterIdentifier,
                DBName = databaseName,
                MasterUsername = masterUsername,
                MasterUserPassword = masterUserPassword,
                NodeType = nodeType,
                NumberOfNodes = 1,
                ClusterType = "single-node"
            };

            var response = await _redshiftClient.CreateClusterAsync(request);
            Console.WriteLine($"Created cluster {clusterIdentifier}");
            return response.Cluster;
        }
        catch (ClusterAlreadyExistsException ex)
        {
            Console.WriteLine($"Cluster already exists: {ex.Message}");
```

```
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Couldn't create cluster. Here's why: {ex.Message}");
            throw;
        }
    }
```

- API에 대한 세부 정보는 AWS SDK for .NET API 참조의 CreateCluster를 참조하세요.

`DeleteCluster`

다음 코드 예시는 `DeleteCluster`의 사용 방법을 보여 줍니다.

SDK for .NET (v4)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
    /// <summary>
    /// Delete an Amazon Redshift cluster without a final snapshot.
    /// </summary>
    /// <param name="clusterIdentifier">The identifier for the cluster.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DeleteClusterWithoutSnapshotAsync(string
clusterIdentifier)
    {
        try
        {
            var request = new DeleteClusterRequest
            {
                ClusterIdentifier = clusterIdentifier,
                SkipFinalClusterSnapshot = true
            };

            var response = await _redshiftClient.DeleteClusterAsync(request);
```

```
            Console.WriteLine($"The {clusterIdentifier} was deleted");
            return true;
        }
        catch (ClusterNotFoundException ex)
        {
            Console.WriteLine($"Cluster not found: {ex.Message}");
            return false;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Couldn't delete cluster. Here's why: {ex.Message}");
            return false;
        }
    }
```

- API 세부 정보는 AWS SDK for .NET API 참조의 DeleteCluster를 참조하세요.

`DescribeClusters`

다음 코드 예시는 `DescribeClusters`의 사용 방법을 보여 줍니다.

SDK for .NET (v4)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
    /// <summary>
    /// Describe Amazon Redshift clusters.
    /// </summary>
    /// <param name="clusterIdentifier">Optional cluster identifier to describe a
specific cluster.</param>
    /// <returns>A list of clusters.</returns>
    public async Task<List<Cluster>> DescribeClustersAsync(string? clusterIdentifier
= null)
    {
        try
        {
            var clusters = new List<Cluster>();
```

```
            var request = new DescribeClustersRequest();
            if (!string.IsNullOrEmpty(clusterIdentifier))
            {
                request.ClusterIdentifier = clusterIdentifier;
            }

            var clustersPaginator =
_redshiftClient.Paginators.DescribeClusters(request);
            await foreach (var response in clustersPaginator.Responses)
            {
                if (response.Clusters != null)
                    clusters.AddRange(response.Clusters);
            }

            Console.WriteLine($"{clusters.Count} cluster(s) retrieved.");
            foreach (var cluster in clusters)
            {
                Console.WriteLine($"\t{cluster.ClusterIdentifier} (Status:
{cluster.ClusterStatus})");
            }

            return clusters;
        }
        catch (ClusterNotFoundException ex)
        {
            Console.WriteLine($"Cluster {clusterIdentifier} not found:
{ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Couldn't describe clusters. Here's why:
{ex.Message}");
            throw;
        }
    }
```

- API 세부 정보는 AWS SDK for .NET API 참조의 DescribeClusters를 참조하세요.

`DescribeStatement`

다음 코드 예시는 `DescribeStatement`의 사용 방법을 보여 줍니다.

SDK for .NET (v4)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
    /// <summary>
    /// Describe a statement execution.
    /// </summary>
    /// <param name="statementId">The statement ID.</param>
    /// <returns>The statement description.</returns>
    public async Task<DescribeStatementResponse> DescribeStatementAsync(string
statementId)
    {
        try
        {
            var request = new DescribeStatementRequest
            {
                Id = statementId
            };

            var response = await
_redshiftDataClient.DescribeStatementAsync(request);
            return response;
        }
        catch (Amazon.RedshiftDataAPIService.Model.ResourceNotFoundException ex)
        {
            Console.WriteLine($"Statement not found: {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Couldn't describe statement. Here's why:
{ex.Message}");
            throw;
        }
    }
```

- API 세부 정보는 AWS SDK for .NET API 참조의 DescribeStatement를 참조하세요.

GetStatementResult

다음 코드 예시는 GetStatementResult의 사용 방법을 보여 줍니다.

SDK for .NET (v4)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
    /// <summary>
    /// Get the results of a statement execution.
    /// </summary>
    /// <param name="statementId">The statement ID.</param>
    /// <returns>A list of result rows.</returns>
    public async Task<List<List<Field>>> GetStatementResultAsync(string statementId)
    {
        try
        {
            var request = new GetStatementResultRequest
            {
                Id = statementId
            };

            var response = await
_redshiftDataClient.GetStatementResultAsync(request);
            return response.Records;
        }
        catch (Amazon.RedshiftDataAPIService.Model.ResourceNotFoundException ex)
        {
            Console.WriteLine($"Statement not found: {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Couldn't get statement result. Here's why:
{ex.Message}");
            throw;
        }
    }
```

- API에 대한 세부 정보는 AWS SDK for .NET API 참조의 GetStatementResult를 참조하세요.

`ListDatabases`

다음 코드 예시는 `ListDatabases`의 사용 방법을 보여 줍니다.

SDK for .NET (v4)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
    /// <summary>
    /// List databases in a Redshift cluster.
    /// </summary>
    /// <param name="clusterIdentifier">The cluster identifier.</param>
    /// <param name="dbUser">The database user.</param>
    /// <param name="dbUser">The database name for authentication.</param>
    /// <returns>A list of database names.</returns>
    public async Task<List<string>> ListDatabasesAsync(string clusterIdentifier,
string dbUser, string databaseName)
    {
        try
        {
            var request = new ListDatabasesRequest
            {
                ClusterIdentifier = clusterIdentifier,
                DbUser = dbUser,
                Database = databaseName
            };

            var response = await _redshiftDataClient.ListDatabasesAsync(request);
            var databases = new List<string>();

            foreach (var database in response.Databases)
            {
                Console.WriteLine($"The database name is : {database}");
```

```
                databases.Add(database);
            }

            return databases;
        }
        catch (Amazon.RedshiftDataAPIService.Model.ValidationException ex)
        {
            Console.WriteLine($"Validation error: {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Couldn't list databases. Here's why: {ex.Message}");
            throw;
        }
    }
```

- API 세부 정보는 AWS SDK for .NET API 참조의 ListDatabases를 참조하시기 바랍니다.

`ModifyCluster`

다음 코드 예시는 `ModifyCluster`의 사용 방법을 보여 줍니다.

SDK for .NET (v4)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
    /// <summary>
    /// Modify an Amazon Redshift cluster.
    /// </summary>
    /// <param name="clusterIdentifier">The identifier for the cluster.</param>
    /// <param name="preferredMaintenanceWindow">The preferred maintenance window.</
param>
    /// <returns>True if successful.</returns>
    public async Task<bool> ModifyClusterAsync(string clusterIdentifier, string
 preferredMaintenanceWindow)
```

```
    {
        try
        {
            var request = new ModifyClusterRequest
            {
                ClusterIdentifier = clusterIdentifier,
                PreferredMaintenanceWindow = preferredMaintenanceWindow
            };

            var response = await _redshiftClient.ModifyClusterAsync(request);
            Console.WriteLine($"The modified cluster was successfully modified and
has {response.Cluster.PreferredMaintenanceWindow} as the maintenance window");
            return true;
        }
        catch (ClusterNotFoundException ex)
        {
            Console.WriteLine($"Cluster {clusterIdentifier} not found:
{ex.Message}");
            return false;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Couldn't modify cluster. Here's why: {ex.Message}");
            return false;
        }
    }
```

- API에 대한 세부 정보는 AWS SDK for .NET API 참조의 ModifyCluster를 참조하세요.

SDK for .NET (v4)를 사용한 Amazon S3 예제

다음 코드 예제에서는 Amazon S3에서 AWS SDK for .NET (v4)를 사용하여 작업을 수행하고 일반적인 시나리오를 구현하는 방법을 보여줍니다.

기본 사항은 서비스 내에서 필수 작업을 수행하는 방법을 보여주는 코드 예제입니다.

작업은 대규모 프로그램에서 발췌한 코드이며 컨텍스트에 맞춰 실행해야 합니다. 작업은 개별 서비스 함수를 직접 호출하는 방법을 보여주며, 관련 시나리오의 컨텍스트에 맞는 작업을 볼 수 있습니다.

시나리오는 동일한 서비스 내에서 또는 다른 AWS 서비스와 결합된 상태에서 여러 함수를 직접적으로 호출하여 특정 태스크를 수행하는 방법을 보여주는 코드 예제입니다.

각 예시에는 전체 소스 코드에 대한 링크가 포함되어 있으며, 여기에서 컨텍스트에 맞춰 코드를 설정하고 실행하는 방법에 대한 지침을 찾을 수 있습니다.

주제

- 시작하기
- 기본 사항
- 작업
- 시나리오

시작하기

Hello Amazon S3

다음 코드 예제에서는 Amazon S3 사용을 시작하는 방법을 보여줍니다.

SDK for .NET (v4)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
/// <summary>
/// Hello Amazon Simple Storage Service
// (Amazon S3) example.
/// </summary>
public class HelloS3
{
    /// <summary>
    /// Main method to run the Hello S3 example.
    /// </summary>
    /// <param name="args">Command line arguments.</param>
    /// <returns>A Task object.</returns>
    public static async Task Main(string[] args)
    {
        var s3Client = new AmazonS3Client();

        try
        {
```

```
            Console.WriteLine("Hello Amazon S3! Let's list your buckets:");
            Console.WriteLine(new string('-', 80));

            // Use the built-in paginator to list buckets
            var request = new ListBucketsRequest();
            var paginator = s3Client.Paginators.ListBuckets(request);

            var buckets = new List<S3Bucket>();

            await foreach (var response in paginator.Responses)
            {
                buckets.AddRange(response.Buckets);
            }

            if (buckets.Any())
            {
                Console.WriteLine($"Found {buckets.Count} S3 buckets:");
                Console.WriteLine();

                foreach (var bucket in buckets)
                {
                    Console.WriteLine($"- Bucket Name: {bucket.BucketName}");
                    Console.WriteLine($"  Creation Date: {bucket.CreationDate:yyyy-
MM-dd HH:mm:ss UTC}");
                    Console.WriteLine();
                }
            }
            else
            {
                Console.WriteLine("No S3 buckets found in your account.");
            }

            Console.WriteLine("Hello S3 completed successfully.");
        }
        catch (AmazonS3Exception ex)
        {
            Console.WriteLine($"S3 service error occurred: {ex.Message}");
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Couldn't list S3 buckets. Here's why:
 {ex.Message}");
        }
    }
```

```
}
```

- API 세부 정보는 AWS SDK for .NET API 참조의 ListBuckets를 참조하세요.

기본 사항

기본 사항 알아보기

다음 코드 예제에서는 다음과 같은 작업을 수행하는 방법을 보여줍니다.

- 버킷을 만들고 버킷에 파일을 업로드합니다.
- 버킷에서 객체를 다운로드합니다.
- 버킷의 하위 폴더에 객체를 복사합니다.
- 버킷의 객체를 나열합니다.
- 버킷 객체와 버킷을 삭제합니다.

SDK for .NET (v4)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

Amazon S3 기능을 보여주는 대화형 시나리오를 실행합니다.

```
public class S3_Basics
{
    public static bool IsInteractive = true;
    public static string BucketName = null!;
    public static string TempFilePath = null!;
    public static S3Wrapper _s3Wrapper = null!;
    public static ILogger<S3_Basics> _logger = null!;

    public static async Task Main(string[] args)
    {
        // Set up dependency injection for the Amazon service.
        using var host = Host.CreateDefaultBuilder(args)
```

```
            .ConfigureServices((_, services) =>
                services.AddAWSService<IAmazonS3>()
                    .AddTransient<S3Wrapper>()
                    .AddLogging(builder => builder.AddConsole()))
            .Build();

        _logger = LoggerFactory.Create(builder => builder.AddConsole())
            .CreateLogger<S3_Basics>();

        _s3Wrapper = host.Services.GetRequiredService<S3Wrapper>();

        var sepBar = new string('-', 45);

        Console.WriteLine(sepBar);
        Console.WriteLine("Amazon Simple Storage Service (Amazon S3) basic");
        Console.WriteLine("procedures. This application will:");
        Console.WriteLine("\n\t1. Create a bucket");
        Console.WriteLine("\n\t2. Upload an object to the new bucket");
        Console.WriteLine("\n\t3. Copy the uploaded object to a folder in the
bucket");
        Console.WriteLine("\n\t4. List the items in the new bucket");
        Console.WriteLine("\n\t5. Delete all the items in the bucket");
        Console.WriteLine("\n\t6. Delete the bucket");
        Console.WriteLine(sepBar);

        await RunScenario(_s3Wrapper, _logger);

        Console.WriteLine(sepBar);
        Console.WriteLine("The Amazon S3 scenario has successfully completed.");
        Console.WriteLine(sepBar);
    }

    /// <summary>
    /// Run the S3 Basics scenario with injected dependencies.
    /// </summary>
    /// <param name="s3Wrapper">The S3 wrapper instance.</param>
    /// <param name="scenarioLogger">The logger instance.</param>
    /// <returns>A Task object.</returns>
    public static async Task RunScenario(S3Wrapper s3Wrapper, ILogger<S3_Basics>
scenarioLogger)
    {
        string bucketName = BucketName;
        string filePath = TempFilePath;
        string keyName = string.Empty;
```

```
        var sepBar = new string('-', 45);

        try
        {
            // Create a bucket.
            Console.WriteLine($"\n{sepBar}");
            Console.WriteLine("\nCreate a new Amazon S3 bucket.\n");
            Console.WriteLine(sepBar);

            if (IsInteractive)
            {
                Console.Write("Please enter a name for the new bucket: ");
                bucketName = Console.ReadLine();
            }
            else
            {
                Console.WriteLine($"Using bucket name: {bucketName}");
            }

            var success = await s3Wrapper.CreateBucketAsync(bucketName);
            if (success)
            {
                Console.WriteLine($"Successfully created bucket: {bucketName}.\n");
            }
            else
            {
                Console.WriteLine($"Could not create bucket: {bucketName}.\n");
            }

            Console.WriteLine(sepBar);
            Console.WriteLine("Upload a file to the new bucket.");
            Console.WriteLine(sepBar);

            if (IsInteractive)
            {
                // Get the local path and filename for the file to upload.
                while (string.IsNullOrEmpty(filePath))
                {
                    Console.Write("Please enter the path and filename of the file to
upload: ");
                    filePath = Console.ReadLine();

                    // Confirm that the file exists on the local computer.
```

```
                    if (!File.Exists(filePath))
                    {
                        Console.WriteLine($"Couldn't find {filePath}. Try again.
\n");
                        filePath = string.Empty;
                    }
                }
            }
            else
            {
                // Use the public variable if set, otherwise create a temp file
                if (!string.IsNullOrEmpty(TempFilePath))
                {
                    filePath = TempFilePath;
                    Console.WriteLine($"Using provided test file: {filePath}");
                }
                else
                {
                    // Create a temporary test file for non-interactive mode
                    filePath = Path.GetTempFileName();
                    var testContent = "This is a test file for S3 basics scenario.
\nGenerated on: " + DateTime.UtcNow.ToString("yyyy-MM-dd HH:mm:ss UTC");
                    await File.WriteAllTextAsync(filePath, testContent);
                    Console.WriteLine($"Created temporary test file: {filePath}");
                }
            }

            // Get the file name from the full path.
            keyName = Path.GetFileName(filePath);

            success = await s3Wrapper.UploadFileAsync(bucketName, keyName,
 filePath);

            if (success)
            {
                Console.WriteLine($"Successfully uploaded {keyName} from {filePath}
 to {bucketName}.\n");
            }
            else
            {
                Console.WriteLine($"Could not upload {keyName}.\n");
            }

            // Set up download path
```

```
            string downloadPath = string.Empty;

            if (IsInteractive)
            {
                // Now get a new location where we can save the file.
                while (string.IsNullOrEmpty(downloadPath))
                {
                    // First get the path to which the file will be downloaded.
                    Console.Write("Please enter the path where the file will be
downloaded: ");
                    downloadPath = Console.ReadLine();

                    // Confirm that the file doesn't already exist on the local
computer.
                    if (File.Exists($"{downloadPath}\\{keyName}"))
                    {
                        Console.WriteLine($"Sorry, the file already exists in that
location.\n");
                        downloadPath = string.Empty;
                    }
                }
            }
            else
            {
                downloadPath = Path.GetTempPath();
                var downloadFile = Path.Combine(downloadPath, keyName);
                if (File.Exists(downloadFile))
                {
                    File.Delete(downloadFile);
                }

                Console.WriteLine($"Using download path: {downloadPath}");
            }

            // Download an object from a bucket.
            success = await s3Wrapper.DownloadObjectFromBucketAsync(bucketName,
keyName, downloadPath);

            if (success)
            {
                Console.WriteLine($"Successfully downloaded {keyName}.\n");
            }
            else
            {
```

```
                Console.WriteLine($"Sorry, could not download {keyName}.\n");
            }

            // Copy the object to a different folder in the bucket.
            string folderName = string.Empty;

            if (IsInteractive)
            {
                while (string.IsNullOrEmpty(folderName))
                {
                    Console.Write("Please enter the name of the folder to copy your
 object to: ");
                    folderName = Console.ReadLine();
                }
            }
            else
            {
                folderName = "test-folder";
                Console.WriteLine($"Using folder name: {folderName}");
            }

            await s3Wrapper.CopyObjectInBucketAsync(bucketName, keyName,
 folderName);

            // List the objects in the bucket.
            await s3Wrapper.ListBucketContentsAsync(bucketName);

            // Delete the contents of the bucket.
            if (IsInteractive)
            {
                Console.WriteLine("Press <Enter> when you are ready to delete the
 bucket contents.");
                _ = Console.ReadLine();
            }

            var deleteContentsSuccess = await
 s3Wrapper.DeleteBucketContentsAsync(bucketName);
            if (deleteContentsSuccess)
            {
                Console.WriteLine($"Successfully deleted contents of {bucketName}.
\n");
            }
            else
            {
```

```
                Console.WriteLine($"Sorry, could not delete contents of
{bucketName}.\n");
            }

            if (IsInteractive)
            {
                // Deleting the bucket too quickly after separately deleting its
contents can
                // cause an error that the bucket isn't empty. To delete contents
and bucket in one
                // operation, use AmazonS3Util.DeleteS3BucketWithObjectsAsync
                Console.WriteLine("Press <Enter> when you are ready to delete the
bucket.");
                _ = Console.ReadLine();
            }
            else
            {
                // Add a small delay for non-interactive mode to ensure objects are
fully deleted.
                Console.WriteLine("Waiting a moment for objects to be fully
deleted...");
                await Task.Delay(2000);
            }

            // Delete the bucket.
            var deleteSuccess = await s3Wrapper.DeleteBucketAsync(bucketName);
            if (deleteSuccess)
            {
                Console.WriteLine($"Successfully deleted {bucketName}.\n");
            }
            else
            {
                Console.WriteLine($"Sorry, could not delete {bucketName}.\n");
            }

            // Clean up temporary files in non-interactive mode
            if (!IsInteractive)
            {
                try
                {
                    if (File.Exists(filePath))
                    {
                        File.Delete(filePath);
                        Console.WriteLine("Cleaned up temporary test file.");
```

```
                    }

                    var downloadFile = Path.Combine(downloadPath, keyName);
                    if (File.Exists(downloadFile))
                    {
                        File.Delete(downloadFile);
                        Console.WriteLine("Cleaned up downloaded test file.");
                    }
                }
                catch (Exception ex)
                {
                    scenarioLogger.LogWarning(ex, "Failed to clean up temporary
files.");
                }
            }
        }
        catch (Exception ex)
        {
            scenarioLogger.LogError(ex, "An error occurred during the S3 scenario
execution.");

            // Clean up on error - delete bucket if it exists
            try
            {
                if (!string.IsNullOrEmpty(bucketName))
                {
                    await s3Wrapper.DeleteBucketContentsAsync(bucketName);
                    await s3Wrapper.DeleteBucketAsync(bucketName);
                }
            }
            catch (Exception cleanupEx)
            {
                scenarioLogger.LogError(cleanupEx, "Error during cleanup.");
            }

            // Clean up temporary files in non-interactive mode
            if (!IsInteractive)
            {
                try
                {
                    if (!string.IsNullOrEmpty(filePath) && File.Exists(filePath))
                    {
                        File.Delete(filePath);
                    }
```

```
                }
                catch (Exception fileCleanupEx)
                {
                    scenarioLogger.LogWarning(fileCleanupEx, "Failed to clean up
 temporary files during error handling.");
                }
            }

            throw;
        }
    }
}
```

Amazon S3 SDK 메서드의 래퍼 클래스입니다.

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

namespace S3_Actions;

/// <summary>
/// This class contains all of the methods for working with Amazon Simple
/// Storage Service (Amazon S3) buckets.
/// </summary>
public class S3Wrapper
{
    private readonly IAmazonS3 _amazonS3;

    /// <summary>
    /// Initializes a new instance of the <see cref="S3Wrapper"/> class.
    /// </summary>
    /// <param name="amazonS3">An initialized Amazon S3 client object.</param>
    public S3Wrapper(IAmazonS3 amazonS3)
    {
        _amazonS3 = amazonS3;
    }


    /// <summary>
    /// Shows how to create a new Amazon S3 bucket.
    /// </summary>
```

```
    /// <param name="bucketName">The name of the bucket to create.</param>
    /// <returns>A boolean value representing the success or failure of
    /// the bucket creation process.</returns>
    public async Task<bool> CreateBucketAsync(string bucketName)
    {
        try
        {
            var request = new PutBucketRequest
            {
                BucketName = bucketName,
                UseClientRegion = true,
            };

            var response = await _amazonS3.PutBucketAsync(request);
            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
        catch (AmazonS3Exception ex)
        {
            Console.WriteLine($"Error creating bucket: '{ex.Message}'");
            return false;
        }
    }



    /// <summary>
    /// Shows how to upload a file from the local computer to an Amazon S3
    /// bucket.
    /// </summary>
    /// <param name="bucketName">The Amazon S3 bucket to which the object
    /// will be uploaded.</param>
    /// <param name="objectName">The object to upload.</param>
    /// <param name="filePath">The path, including file name, of the object
    /// on the local computer to upload.</param>
    /// <returns>A boolean value indicating the success or failure of the
    /// upload procedure.</returns>
    public async Task<bool> UploadFileAsync(
        string bucketName,
        string objectName,
        string filePath)
    {
        try
        {
            var request = new PutObjectRequest
```

```
            {
                BucketName = bucketName,
                Key = objectName,
                FilePath = filePath,
            };

            var response = await _amazonS3.PutObjectAsync(request);
            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
        catch (AmazonS3Exception ex)
        {
            Console.WriteLine($"Error uploading {objectName}: {ex.Message}");
            return false;
        }
    }



    /// <summary>
    /// Shows how to download an object from an Amazon S3 bucket to the
    /// local computer.
    /// </summary>
    /// <param name="bucketName">The name of the bucket where the object is
    /// currently stored.</param>
    /// <param name="objectName">The name of the object to download.</param>
    /// <param name="filePath">The path, including filename, where the
    /// downloaded object will be stored.</param>
    /// <returns>A boolean value indicating the success or failure of the
    /// download process.</returns>
    public async Task<bool> DownloadObjectFromBucketAsync(
        string bucketName,
        string objectName,
        string filePath)
    {
        var request = new GetObjectRequest
        {
            BucketName = bucketName,
            Key = objectName,
        };

        using GetObjectResponse response = await _amazonS3.GetObjectAsync(request);

        try
        {
```

```
            // Save object to local file
            await response.WriteResponseStreamToFileAsync($"{filePath}\
\{objectName}", true, CancellationToken.None);
            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
        catch (AmazonS3Exception ex)
        {
            Console.WriteLine($"Error saving {objectName}: {ex.Message}");
            return false;
        }
    }



    /// <summary>
    /// Copies an object in an Amazon S3 bucket to a folder within the
    /// same bucket.
    /// </summary>
    /// <param name="bucketName">The name of the Amazon S3 bucket where the
    /// object to copy is located.</param>
    /// <param name="objectName">The object to be copied.</param>
    /// <param name="folderName">The folder to which the object will
    /// be copied.</param>
    /// <returns>A boolean value that indicates the success or failure of
    /// the copy operation.</returns>
    public async Task<bool> CopyObjectInBucketAsync(
        string bucketName,
        string objectName,
        string folderName)
    {
        try
        {
            var request = new CopyObjectRequest
            {
                SourceBucket = bucketName,
                SourceKey = objectName,
                DestinationBucket = bucketName,
                DestinationKey = $"{folderName}\\{objectName}",
            };
            var response = await _amazonS3.CopyObjectAsync(request);
            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
        catch (AmazonS3Exception ex)
        {
```

```
            Console.WriteLine($"Error copying object: '{ex.Message}'");
            return false;
        }
    }



    /// <summary>
    /// Shows how to list the objects in an Amazon S3 bucket.
    /// </summary>
    /// <param name="bucketName">The name of the bucket for which to list.
    /// <param name="printList">True to print out the list.
    /// <returns>The collection of objects.</returns>
    public async Task<List<S3Object>?> ListBucketContentsAsync(string bucketName,
bool printList = true)
    {
        try
        {
            var request = new ListObjectsV2Request
            {
                BucketName = bucketName,
                MaxKeys = 5,
            };

            if (printList)
            {
                Console.WriteLine("--------------------------------------");
                Console.WriteLine($"Listing the contents of {bucketName}:");
                Console.WriteLine("--------------------------------------");
            }

            var listObjectsV2Paginator = _amazonS3.Paginators.ListObjectsV2(new
ListObjectsV2Request
            {
                BucketName = bucketName,
            });
            var s3Objects = new List<S3Object>();
            await foreach (var response in listObjectsV2Paginator.Responses)
            {
                if (response.S3Objects != null)
                {
                    s3Objects.AddRange(response.S3Objects);
                }
            }
```

```
            if (printList)
            {
                Console.WriteLine($"Number of Objects: {s3Objects.Count}");
                foreach (var entry in s3Objects)
                {
                    Console.WriteLine($"Key = {entry.Key} Size = {entry.Size}");
                }
            }

            return s3Objects;
        }
        catch (AmazonS3Exception ex)
        {
            Console.WriteLine($"Error encountered on server. Message:'{ex.Message}'
getting list of objects.");
            return null;
        }
    }



    /// <summary>
    /// Delete all of the objects stored in an existing Amazon S3 bucket.
    /// </summary>
    /// <param name="bucketName">The name of the bucket from which the
    /// contents will be deleted.</param>
    /// <returns>A boolean value that represents the success or failure of
    /// deleting all of the objects in the bucket.</returns>
    public async Task<bool> DeleteBucketContentsAsync(string bucketName)
    {
        // Iterate over the contents of the bucket and delete all objects.
        try
        {
            // Delete all objects in the bucket.
            var deleteList = await ListBucketContentsAsync(bucketName, false);
            if (deleteList != null && deleteList.Any())
            {
                await _amazonS3.DeleteObjectsAsync(new DeleteObjectsRequest()
                {
                    BucketName = bucketName,
                    Objects = deleteList.Select(o => new KeyVersion { Key =
o.Key }).ToList(),
                });
```

```
            }

            return true;
        }
        catch (AmazonS3Exception ex)
        {
            Console.WriteLine($"Error deleting objects: {ex.Message}");
            return false;
        }
    }



    /// <summary>
    /// Shows how to delete an Amazon S3 bucket.
    /// </summary>
    /// <param name="bucketName">The name of the Amazon S3 bucket to delete.</param>
    /// <returns>A boolean value that represents the success or failure of
    /// the delete operation.</returns>
    public async Task<bool> DeleteBucketAsync(string bucketName)
    {
        try
        {
            var request = new DeleteBucketRequest { BucketName = bucketName, };

            await _amazonS3.DeleteBucketAsync(request);
            return true;
        }
        catch (AmazonS3Exception ex)
        {
            Console.WriteLine($"Error deleting bucket: {ex.Message}");
            return false;
        }
    }

}
```

- API 세부 정보는 AWS SDK for .NET API 참조의 다음 항목을 참조하세요.
  - CopyObject
  - CreateBucket

- DeleteBucket
- DeleteObjects
- GetObject
- ListObjectsV2
- PutObject

작업

`CopyObject`

다음 코드 예시는 `CopyObject`의 사용 방법을 보여 줍니다.

SDK for .NET (v4)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
/// <summary>
/// Copies an object in an Amazon S3 bucket to a folder within the
/// same bucket.
/// </summary>
/// <param name="bucketName">The name of the Amazon S3 bucket where the
/// object to copy is located.</param>
/// <param name="objectName">The object to be copied.</param>
/// <param name="folderName">The folder to which the object will
/// be copied.</param>
/// <returns>A boolean value that indicates the success or failure of
/// the copy operation.</returns>
public async Task<bool> CopyObjectInBucketAsync(
    string bucketName,
    string objectName,
    string folderName)
{
    try
    {
        var request = new CopyObjectRequest
```

```
            {
                SourceBucket = bucketName,
                SourceKey = objectName,
                DestinationBucket = bucketName,
                DestinationKey = $"{folderName}\\{objectName}",
            };
            var response = await _amazonS3.CopyObjectAsync(request);
            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
        catch (AmazonS3Exception ex)
        {
            Console.WriteLine($"Error copying object: '{ex.Message}'");
            return false;
        }
    }
```

- API 세부 정보는 AWS SDK for .NET API 참조의 CopyObject를 참조하세요.

`CreateBucket`

다음 코드 예시는 `CreateBucket`의 사용 방법을 보여 줍니다.

SDK for .NET (v4)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
    /// <summary>
    /// Shows how to create a new Amazon S3 bucket.
    /// </summary>
    /// <param name="bucketName">The name of the bucket to create.</param>
    /// <returns>A boolean value representing the success or failure of
    /// the bucket creation process.</returns>
    public async Task<bool> CreateBucketAsync(string bucketName)
    {
        try
```

```
        {
            var request = new PutBucketRequest
            {
                BucketName = bucketName,
                UseClientRegion = true,
            };

            var response = await _amazonS3.PutBucketAsync(request);
            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
        catch (AmazonS3Exception ex)
        {
            Console.WriteLine($"Error creating bucket: '{ex.Message}'");
            return false;
        }
    }
```

- API 세부 정보는 AWS SDK for .NET API 참조의 CreateBucket을 참조하세요.

`CreatePresignedPost`

다음 코드 예시는 `CreatePresignedPost`의 사용 방법을 보여 줍니다.

SDK for .NET (v4)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

미리 서명된 POST URL을 생성합니다.

```
    /// <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);
    }
```

- API 세부 정보는 AWS SDK for .NET API 참조의 CreatePresignedPost를 참조하세요.

DeleteBucket

다음 코드 예시는 DeleteBucket의 사용 방법을 보여 줍니다.

SDK for .NET (v4)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
/// <summary>
/// Shows how to delete an Amazon S3 bucket.
/// </summary>
/// <param name="bucketName">The name of the Amazon S3 bucket to delete.</param>
/// <returns>A boolean value that represents the success or failure of
/// the delete operation.</returns>
public async Task<bool> DeleteBucketAsync(string bucketName)
{
    try
    {
        var request = new DeleteBucketRequest { BucketName = bucketName, };

        await _amazonS3.DeleteBucketAsync(request);
        return true;
    }
    catch (AmazonS3Exception ex)
    {
        Console.WriteLine($"Error deleting bucket: {ex.Message}");
        return false;
    }
}
```

- API 세부 정보는 AWS SDK for .NET API 참조의 DeleteBucket을 참조하세요.

DeleteObjects

다음 코드 예시는 DeleteObjects의 사용 방법을 보여 줍니다.

SDK for .NET (v4)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
    /// <summary>
    /// Delete all of the objects stored in an existing Amazon S3 bucket.
    /// </summary>
    /// <param name="bucketName">The name of the bucket from which the
    /// contents will be deleted.</param>
    /// <returns>A boolean value that represents the success or failure of
    /// deleting all of the objects in the bucket.</returns>
    public async Task<bool> DeleteBucketContentsAsync(string bucketName)
    {
        // Iterate over the contents of the bucket and delete all objects.
        try
        {
            // Delete all objects in the bucket.
            var deleteList = await ListBucketContentsAsync(bucketName, false);
            if (deleteList != null && deleteList.Any())
            {
                await _amazonS3.DeleteObjectsAsync(new DeleteObjectsRequest()
                {
                    BucketName = bucketName,
                    Objects = deleteList.Select(o => new KeyVersion { Key =
o.Key }).ToList(),
                });
            }

            return true;
        }
        catch (AmazonS3Exception ex)
        {
            Console.WriteLine($"Error deleting objects: {ex.Message}");
            return false;
        }
    }
```

- API 세부 정보는 AWS SDK for .NET API 참조의 DeleteObjects를 참조하세요.

`GetObject`

다음 코드 예시는 `GetObject`의 사용 방법을 보여 줍니다.

SDK for .NET (v4)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
/// <summary>
/// Shows how to download an object from an Amazon S3 bucket to the
/// local computer.
/// </summary>
/// <param name="bucketName">The name of the bucket where the object is
/// currently stored.</param>
/// <param name="objectName">The name of the object to download.</param>
/// <param name="filePath">The path, including filename, where the
/// downloaded object will be stored.</param>
/// <returns>A boolean value indicating the success or failure of the
/// download process.</returns>
public async Task<bool> DownloadObjectFromBucketAsync(
    string bucketName,
    string objectName,
    string filePath)
{
    var request = new GetObjectRequest
    {
        BucketName = bucketName,
        Key = objectName,
    };

    using GetObjectResponse response = await _amazonS3.GetObjectAsync(request);

    try
```

```
        {
            // Save object to local file
            await response.WriteResponseStreamToFileAsync($"{filePath}\
\{objectName}", true, CancellationToken.None);
            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
        catch (AmazonS3Exception ex)
        {
            Console.WriteLine($"Error saving {objectName}: {ex.Message}");
            return false;
        }
    }
```

- API 세부 정보는 AWS SDK for .NET API 참조의 GetObject를 참조하세요.

`ListObjectsV2`

다음 코드 예시는 `ListObjectsV2`의 사용 방법을 보여 줍니다.

SDK for .NET (v4)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
    /// <summary>
    /// Shows how to list the objects in an Amazon S3 bucket.
    /// </summary>
    /// <param name="bucketName">The name of the bucket for which to list.
    /// <param name="printList">True to print out the list.
    /// <returns>The collection of objects.</returns>
    public async Task<List<S3Object>?> ListBucketContentsAsync(string bucketName,
 bool printList = true)
    {
        try
        {
            var request = new ListObjectsV2Request
```

```
            {
                BucketName = bucketName,
                MaxKeys = 5,
            };

            if (printList)
            {
                Console.WriteLine("--------------------------------------");
                Console.WriteLine($"Listing the contents of {bucketName}:");
                Console.WriteLine("--------------------------------------");
            }

            var listObjectsV2Paginator = _amazonS3.Paginators.ListObjectsV2(new
ListObjectsV2Request
            {
                BucketName = bucketName,
            });
            var s3Objects = new List<S3Object>();
            await foreach (var response in listObjectsV2Paginator.Responses)
            {
                if (response.S3Objects != null)
                {
                    s3Objects.AddRange(response.S3Objects);
                }
            }

            if (printList)
            {
                Console.WriteLine($"Number of Objects: {s3Objects.Count}");
                foreach (var entry in s3Objects)
                {
                    Console.WriteLine($"Key = {entry.Key} Size = {entry.Size}");
                }
            }

            return s3Objects;
        }
        catch (AmazonS3Exception ex)
        {
            Console.WriteLine($"Error encountered on server. Message:'{ex.Message}'
getting list of objects.");
            return null;
        }
    }
```

- API 세부 정보는 AWS SDK for .NET API 참조의 ListObjectsV2를 참조하세요.

`PutObject`

다음 코드 예시는 `PutObject`의 사용 방법을 보여 줍니다.

SDK for .NET (v4)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
/// <summary>
/// Shows how to upload a file from the local computer to an Amazon S3
/// bucket.
/// </summary>
/// <param name="bucketName">The Amazon S3 bucket to which the object
/// will be uploaded.</param>
/// <param name="objectName">The object to upload.</param>
/// <param name="filePath">The path, including file name, of the object
/// on the local computer to upload.</param>
/// <returns>A boolean value indicating the success or failure of the
/// upload procedure.</returns>
public async Task<bool> UploadFileAsync(
    string bucketName,
    string objectName,
    string filePath)
{
    try
    {
        var request = new PutObjectRequest
        {
            BucketName = bucketName,
            Key = objectName,
            FilePath = filePath,
        };
```

```
            var response = await _amazonS3.PutObjectAsync(request);
            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
        catch (AmazonS3Exception ex)
        {
            Console.WriteLine($"Error uploading {objectName}: {ex.Message}");
            return false;
        }
    }
```

- API 세부 정보는 AWS SDK for .NET API 참조의 PutObject를 참조하십시오.

시나리오

미리 서명된 URL 생성

다음 코드 예제는 Amazon S3에 대해 미리 서명된 URL을 생성하고 객체를 업로드하는 방법을 보여줍니다.

SDK for .NET (v4)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

브라우저에서 직접 업로드할 수 있도록 미리 서명된 POST URL을 생성하고 사용합니다.

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

에 대한 프로젝트 마이그레이션 AWS SDK for .NET

이 섹션에서는 버전 4의 마이그레이션 작업에 대한 정보와 이러한 작업을 수행하는 방법에 대한 AWS SDK for .NET지침을 제공합니다. 이전 버전으로 마이그레이션하는 방법(예: V2에서 V3로 또는 한 V3 하위 버전에서 다른 버전으로 마이그레이션)에 대한 자세한 내용은 AWS SDK for .NET 개발자 안내서 (버전 3)의 프로젝트 마이그레이션을 참조하세요.

새로운 기능 및 지원하는 플랫폼도 참조하세요.

주제

- 의 버전 4로 마이그레이션 AWS SDK for .NET
- .NET Standard 1.3에서 마이그레이션

의 버전 4로 마이그레이션 AWS SDK for .NET

AWS SDK for .NET 버전 4(V4)는 SDK 버전 3(V3)에서 크게 변경되었습니다. 이 주제에서는 버전 4의 주요 변경 사항과 V3에서 환경 또는 코드를 마이그레이션하기 위해 수행해야 할 수 있는 작업에 대해 설명합니다. SDK의 기타 주목할 만한 변경 사항에 대한 자세한 내용은 다음 리소스를 참조하세요.

- GitHub의 개발-트래커 문제: https://github.com/aws/aws-sdk-net/issues/3362.
- 블로그 게시물 Preview 1 of AWS SDK for .NET V4.
- 블로그 게시물 Preview 4 of AWS SDK for .NET V4.
- 블로그 게시물 AWS SDK for .NET V4.0의 일반 가용성.

.NET Framework

의 V4에서 .NET Framework 3.5 대상이 제거되었습니다 AWS SDK for .NET. 따라서 SDK는 더 이상 .NET Framework 3.5를 지원하지 않습니다. 이 SDK 버전은 .NET Framework 4.7.2에 대해 컴파일되며 .NET 4.0 런타임에서 실행됩니다. 자세한 내용은 지원되는 플랫폼을 참조하세요.

값 유형

요청 및 응답을 생성하는 데 사용되는 클래스에서 값 유형을 사용하는 속성이 null 가능한 값 유형을 사용하도록 변경되었습니다. 다음 유형의 속성이 변경되었습니다.

- `bool`가 로 변경되었습니다. `bool?`

- double가 로 변경되었습니다. double?
- int가 로 변경되었습니다. int?
- float가 로 변경되었습니다. float?
- long가 로 변경되었습니다. long?
- Datetime가 로 변경되었습니다. Datetime?

이 변경 사항에 대한 자세한 내용은 블로그 게시물 V4의 AWS SDK for .NET 미리 보기 1을 참조하세요.

컬렉션

요청 및 응답을 생성하는 데 사용되는 클래스에서 컬렉션을 사용하는 속성의 기본값은 입니다null. 따라서 코드를 사용하려면 먼저 컬렉션이 null이 아닌지 확인해야 합니다. 예제:

```
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.
    }
}
```

컬렉션을 초기화하는 V3 동작은를 Amazon.AWSConfigs.InitializeCollections로 설정하여 복원할 수 있습니다true. 이 속성은 V4로 업그레이드하기 전에이 동작 변경을 시도하려는 사용자를 위해 V3에도 존재합니다. V4

이 변경 사항에 대한 자세한 내용은 블로그 게시물 V4의 AWS SDK for .NET 미리 보기 1을 참조하세요.

AWS Security Token Service (STS)

- 리전 엔드포인트

  에 의존하는 자격 증명 공급자를 사용하는 경우 호출 AWS STS은 항상 리전 엔드포인트를 사용합니다. 이는 구성된 us-east-1 리전에 관계없이 퍼블릭 파티션에서 실행할 때 기본적으로 리전을 사용한 SDK의 V3와 다릅니다.

- **`StsRegionalEndpointsValue` 열거형**

  열거형`StsRegionalEndpointsValue`이 Amazon.Runtime 네임스페이스에서 제거되었습니다. 해당 열거형을 사용하는 모든 코드는 제거해야 합니다.

- **`STSAssumeRoleAWSCredentials` 클래스**

  더 이상 사용되지 않는 STS 수임 역할 자격 증명 공급자 `STSAssumeRoleAWSCredentials`가 Amazon.SecurityToken 네임스페이스에서 제거되었습니다. 대신 Amazon.Runtime의 AssumeRoleAWSCredentials를 사용합니다.

와 관련된 변경 사항 `ClientConfig`

Amazon.Runtime.ClientConfig 클래스는 AmazonS3Config와 같은 서비스 클라이언트 구성 클래스의 기본 클래스입니다. 이 기본 클래스는 다음과 같이 변경되었습니다.

- 기본 재시도 모드

  `RetryMode` 속성의 기본값은 `Standard` 대신 입니다`Legacy`. 따라서 Amazon.Runtime.RequestRetryMode 열거형에서 `Legacy` 값이 제거되었습니다.

- 기본 구성 모드

  `DefaultConfigurationMode` 속성의 기본값은 `Standard` 대신 입니다`Legacy`. 따라서 Amazon.Runtime.DefaultConfigurationMode 열거형에서 `Legacy` 값이 제거되었습니다.

- **`ReadWriteTimeout` 속성**

  .NET Framework 4.7.2를 제외한 모든 대상에서 더 이상 사용되지 않는 `ReadWriteTimeout` 속성이 제거되었습니다.

AWSSDK.Extensions.NETCore.Setup NuGet 패키지

AWSSDK.Extensions.NETCore.Setup NuGet 패키지는 SDK의 V3에 있는 문제를 완화하고 패키지를 네이티브 AOT에 안전하게 만들기 위해 업데이트되었습니다. 이러한 변경 사항은 아래에 요약되어 있습니다. 자세한 내용은 GitHub의 aws-sdk-net 리포지토리에서 PR 3353을 참조하세요.

- **`DefaultClientConfig` 클래스**

  `DefaultClientConfig` 클래스는 더 이상 서비스 클라이언트 구성 기본 클래스 Amazon.Runtime.ClientConfig에서 상속되지 않습니다. 의 관련 속성은 null 가능한 값 유형을

DefaultClientConfig 사용하여에 복제ClientConfig되었습니다. 이 변경을 통해 서비스 클라이언트에 대해 생성 중인 구성에 값을 복사할 DefaultClientConfig 때 값이 설정된 시기를 감지할 수 있습니다.

이 변경의 특정 결과 중 하나는 V4에서 DefaultClientConfig.HttpClientFactory를 더 이상 사용할 수 없다는 것입니다. 대신 AWSConfigs.HttpClientFactory을 사용하세요. 자세한 내용은 GitHub 문제 3790을 참조하세요.

이 변경의 또 다른 결과는 서비스 구성 객체를 수락한 IConfiguration.GetAWSOptions 확장 메서드의 일반 오버로드가 제거되었다는 것입니다. 일반이 아닌 오버로드는 대신 사용해야 하며 SDK는 채우는 서비스별 설정을 자동으로 처리합니다. 자세한 내용은 GitHub 문제 3866을 참조하세요.

- 네이티브 AOT

  C# 11 정적 인터페이스 메서드를 사용하는 서비스 클라이언트를 생성하는 새로운 메커니즘이 패키지에 추가되었습니다. 이 변경으로 인해 기본 AOT와 호환되지 않는 서비스 클라이언트 유형을 계산하기 위한 서비스 인터페이스 이름의 문자열 조작을 포함하여 어셈블리 유형 로드를 수행하여 서비스 클라이언트의 인스턴스를 생성할 필요가 없습니다. 이 변경 사항은 .NET 8 이상에서만 사용할 수 있으며 이전 버전에서는 여전히 원래 메커니즘을 사용합니다.

이 가이드에서이 패키지에 대한 자세한 내용은 섹션을 참조하세요AWSSDK.Extensions.NETCore.Setup 및 IConfiguration. 이 패키지의 소스 코드는 https://github.com/aws/aws-sdk-net/tree/main/extensions/src/AWSSDK.Extensions.NETCore.Setup의 GitHub에 있습니다.

CookieSigner 및 UrlSigner

Amazon CloudFront의 CookieSigner 및 UrlSigner 확장은 AWSSDK.Extensions.CloudFront.Signers라는 별도의 확장 패키지로 이동되었습니다. 이 변경 사항은 OpenSSL 3을 지원하고 BouncyCastle.Cryptography에 종속되도록 하기 위한 것입니다.

이 패키지의 소스 코드는 https://github.com/aws/aws-sdk-net/tree/main/extensions/src/AWSSDK.Extensions.CloudFront.Signers의 GitHub에 있습니다.

DateTime과 UTC DateTime 비교

일부 V3 클래스에는 "사용되지 않음" 또는 "사용되지 않음"으로 표시된 DateTime 속성과 대체 UTC DateTime 속성이 있습니다. 이러한 클래스에서는 더 이상 사용되지 않는 DateTime 속성이 제거되고 UTC DateTime 속성의 이름이 DateTime 속성의 원래 이름으로 변경되었습니다.

다음은이 변경이 구현된 클래스의 몇 가지 예입니다.

- DescribeSpotPriceHistoryRequest:
  - 더 이상 사용되지 않는 `StartTime` 속성이 제거되었으며 `StartTimeUtc` 속성 이름이 "StartTime"으로 변경되었습니다.
  - 더 이상 사용되지 않는 `EndTime` 속성이 제거되었으며 `EndTimeUtc` 속성 이름이 "EndTime"으로 변경되었습니다.
- CreateFleetRequest
  - 더 이상 사용되지 않는 `ValidFrom` 속성이 제거되었으며 `ValidFromUtc` 속성 이름이 "ValidFrom"으로 변경되었습니다.
  - 더 이상 사용되지 않는 `ValidUntil` 속성이 제거되었으며 `ValidUntilUtc` 속성 이름이 "ValidUntil"로 변경되었습니다.

이 변경으로 인해 애플리케이션이 더 이상 사용되지 않는 원래 DateTime 속성을 사용하는 경우 오프셋 시간이 발생할 수 있습니다. UTC DateTime 속성을 사용하는 코드에 대해 컴파일 시간 오류가 발생합니다.

DateTime 구문 분석

DateTimeUnmarshaller 클래스가 업데이트되었습니다. 이 클래스는 DateTime 문자열을 현지 시간으로 구문 분석하고 반환했습니다. 경우에 따라 이러한 값은 이전 업데이트로 인해 UTC로 다시 변환되지만 항상 그런 것은 아닙니다. 이제 마샬링되지 않은 DateTime 문자열은 UTC로 간주되고 UTC로 지정되고 마샬링되지 않습니다. 이 업데이트에는 다음과 같은 동작 변경 사항이 포함됩니다.

DateTime 클래스를 기반으로 하는 특정 타임스탬프 속성이 현지 시간으로 구문 분석되고 있었습니다. 여기에는 타임스탬프에 대한 응답 마샬러와 형식 `TimestampFormat.ISO8601` 및에 대한 목록 타임스탬프가 포함되었습니다`TimestampFormat.RFC822`. 대신 UTC 시간을 반환하도록 DateTime 구문 분석이 업데이트되었습니다.

`ConvertFromUnixEpochSeconds` 및 `ConvertFromUnixEpochMilliseconds`

Unix 에포크 초를 DateTime 구조로 변환하는 ConvertFromUnixEpochSeconds 및 ConvertFromUnixEpochMilliseconds 메서드는 Unix 에포크 시간을 UTC 시간이 아닌 현지 시간으로 반환했습니다. DateTime 이제 이러한 메서드는 UTC 시간을 반환합니다.

로깅

SDK에서 로깅을 활성화하는 방법이 V4에 대해 업데이트되었습니다. 콘솔 및 시스템 진단에 대한 로깅은 V3와 동일하게 작동합니다. 즉, AWSConfigs 클래스의 `LoggingConfig.LogTo` 속성을 `LoggingOptions.Console` 또는 로 설정하여 작동합니다`LoggingOptions.SystemDiagnostics`. 에 대한 LoggingOptions 옵션이 반사를 사용하여의 인메모리 인스턴스에 연결하기 위한 SDK의 내부 로직과 함께 제거`log4net`되었습니다`log4net`.

SDK의 로깅을 로깅 프레임워크에 포함하기 위해 별도의 어댑터 패키지를 사용하여 SDK를 로깅 프레임워크와 연결합니다. 용 AWSSDK.Extensions.Logging.Log4NetAdaptor 패키지`log4net`와 용 AWSSDK.Extensions.Logging.ILoggerAdaptor 패키지를 사용합니다`Microsoft.Extensions.Logging`. 다음 코드 예제에서는이 두 경우에 로깅을 구성하는 방법을 보여줍니다.

log4net에 대한 구성 예제

`AWSSDK.Extensions.Logging.Log4NetAdaptor` NuGet 패키지를 추가하고에서 정적 `ConfigureAWSSDKLogging` 메서드를 호출합니다`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.Logging에 대한 구성 예제

`AWSSDK.Extensions.Logging.ILoggerAdaptor` NuGet 패키지를 추가하고 `ILoggerFactory` 인터페이스에서 `ConfigureAWSSDKLogging` 확장 메서드를 호출합니다.

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

var app = builder.Build();

app.Services.GetRequiredService<ILoggerFactory>()
    .ConfigureAWSSDKLogging();
```

HTTP 2 지원

양방향 스트리밍을 활성화하기 위해 HTTP 2에 대한 지원이 추가되었습니다. 자세한 내용은 HTTP 2 지원을 참조하세요.

Single Sign-On

SSOAWSCredentialsOptions 클래스 SupportsGettingNewToken 속성의 기본값이에서 true로 변경되었습니다false. SSOAWSCredentials 클래스를 사용하여 SSO 자격 증명을 가져오는 애플리케이션이 있는 경우 Options.SupportsGettingNewToken 속성을 로 설정해야 할 수 있습니다true. 이 구성의 예는의 코드 예제를 참조하세요.NET 애플리케이션만 사용하는 SSO 자습서. 자세한 내용은 aws-sdk-net GitHub 리포지토리의 PR 3737을 참조하세요.

DynamoDB 관련 변경 사항

다음 변경 사항은 Amazon DynamoDB에만 적용됩니다. 이들 중 다수는 주요 변경 사항입니다.

V4의 DynamoDB 변경 사항에 대한 자세한 내용은 블로그 게시물 V4의 AWS SDK for .NET V4를 AWS SDK for .NET참조하세요.

항목을 보려면 열기

SDK for DynamoDB의 V4 변경 사항은 테스트 가능성과 관련된 몇 가지 문제를 해결하지만 주로 상위 수준 라이브러리를 중심으로 합니다.

- 코드에서 DocumentModel이라는 .NET 문서 모델입니다.
- 코드에서 DataModel이라는 .NET 객체 지속성 모델입니다.

이러한 프로그래밍 모드에 대한 자세한 내용은이 설명서DynamoDB의 섹션을 참조하세요.

문서 모델: 모의 **IAmazonDynamoDB** 인터페이스에 대한 예외 업데이트

SDK V4 이전의 문서 모델에서 테이블이 모의 IAmazonDynamoDB 인터페이스로 초기화된 경우를 반환합니다NullReferenceException. 대신 SDK의 V4가를 반환InvalidOperationException합니다. 비동기 Table 메서드는 모의 클라이언트에서 작동해야 하지만에서 동기 메서드를 호출할 때 예외가 계속 표시될 수 있습니다.NET/Core/Standard.

이 변경 사항에 대한 자세한 내용은 GitHub의 PR 3388을 참조하세요.

문서 모델: **FromJson** 및 **ToJson** 메서드

문서 클래스의 FromJson 및 ToJson 메서드는 이제 직렬화에 LitJson System.Text.Json 대신를 사용하며, LitJson은 SDK의 V4에서 제거되었습니다. 를 사용하면이 구문 분석기System.Text.Json가 숫자 부동 소수점 속성에 대해 더 높은 정밀도를 지원하는 .NET Decimal 유형 사용을 지원한다는 이점이 있습니다.

객체 지속성 모델: **DynamoDBOperationConfig** 클래스

객체 지속성 모델에서 공유 DynamoDBOperationConfig 클래스가 다음과 같이 변경되었습니다.

- 클래스는 SaveConfig, LoadConfig 및 QueryConfig와 같은 새로운 작업별 클래스로 분리되었습니다. 를 사용하는 메서DynamoDBOperationConfig드는 더 이상 사용되지 않는 것으로 표시되었으며 향후 제거될 수 있습니다.

  이 변경 사항에 대한 자세한 내용은 GitHub의 PR 3421을 참조하세요.
- MetadataCachingMode 및 DisableFetchingTableMetadata 속성이 클래스에서 제거되었습니다. 이러한 속성은 앞서 언급한 새로운 작업별 클래스에 포함되지 않았습니다. 제거된 속성은 AWSConfigsDynamoDB 클래스의 전역 Context 속성 또는 DynamoDBContextConfig 클래스에 지정해야 하는 테이블 수준 설정입니다.

  이 변경 사항에 대한 자세한 내용은 GitHub의 PR 3422를 참조하세요.
- 클래스는 더 이상 DynamoDBContextConfig 클래스에서 상속되지 않습니다. 이렇게 하면 작업별 구성의 일부 속성(예: OverrideTableName)이 적용되지 않는 DynamoDBContext의 생성자에 DynamoDBOperationConfig 객체를 전달할 수 없습니다.

  이 변경 사항에 대한 자세한 내용은 GitHub의 PR 3422를 참조하세요.

객체 지속성 모델: 다형성

DynamoDBPolymorphicTypeAttribute 클래스가 객체 지속성 모델에 추가되었습니다. 이 클래스를 사용하면 다형성 유형의 직렬화 및 역직렬화를 지원할 수 있습니다. 자세한 내용은 GitHub의 PR 3643을 참조하세요.

문서 모델 및 객체 지속성 모델: 모의 작업

고객이 DynamoDB 작업을 모의할 수 있는 새로운 작업별 인터페이스가 추가되었습니다. IDynamoDBContext 인터페이스의 팩토리 메서드가 새 인터페이스를 반환하도록 업데이트되었습니다.

이 변경 사항에 대한 자세한 내용은 GitHub의 PR 3450을 참조하세요.

- 객체 지속성 모델
  - `IBatchGet` 및 `IMultiTableBatchGet` 인터페이스를 통한 모의 `BatchGet` 작업.
  - `IBatchWrite` 및 `IMultiTableBatchWrite` 인터페이스를 통한 모의 `BatchWrite` 작업.
  - `ITransactGet` 및 `IMultiTableTransactGet` 인터페이스를 통한 모의 `TransactGet` 작업.
  - `ITransactWrite` 및 `IMultiTableTransactWrite` 인터페이스를 통한 모의 `TransactWrite` 작업.
  - `IAsyncSearch` 인터페이스를 통한 모의 `Scan` 및 `Query` 작업.
- 문서 모델
  - `ITable` 인터페이스를 통한 모의 `Table` 작업.
  - `ISearch` 인터페이스를 통한 모의 `Scan` 및 `Query` 작업.
  - `IDocumentTransactWrite` 및 `IMultiTableDocumentTransactWrite` 인터페이스를 통한 모의 `TransactWrite` 작업.
  - `IDocumentTransactGet` 및 `IMultiTableDocumentTransactGet` 인터페이스를 통한 모의 `TransactGet` 작업.
  - `IDocumentBatchWrite` 및 `IMultiTableDocumentBatchWrite` 인터페이스를 통한 모의 `BatchWrite` 작업.
  - `IDocumentBatchGet` 및 `IMultiTableDocumentBatchGet` 인터페이스를 통한 모의 `BatchGet` 작업.

문서 모델 및 객체 지속성 모델: 네이티브 AOT 지원

기본 AOT의 제한 사항은 중첩된 .NET 유형을 지원하는 것입니다. 경우에 따라 이러한 중첩된 유형은 .NET 컴파일러의 트리밍 구성 요소에서 눈에 띄지 않을 수 있습니다. 이 경우 다음과 같은 예외가 발생할 수 있습니다. `"System.InvalidOperationException: Type <type> is unsupported, it cannot be instantiated."`

트리머에게 하위 유형에 대한 종속성을 알려주는 코드 경로의 `DynamicDependency` 어딘가에를 추가하여이 제한을 해결할 수 있습니다. 저장 중인 최상위 .NET 유형의 생성자가 가능성이 높습니다. 다음 코드 예제에서는 `DynamicDependency` 속성을 사용하는 방법을 보여줍니다.

```
[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

DynamoDBStreams는 AWSSDK.DynamoDB NuGet 패키지에서 제거되었으며 자체 패키지인 AWSSDK.DynamoDBStreams에서 사용할 수 있으며 자체 네임스페이스인가 있습니다Amazon.DynamoDBStreams.

`TableNamePrefix` 값 제거 허용

이제 개별 작업 수준에서 DynamoDBContextConfig 클래스의 `TableNamePrefix` 속성 값을 제거할 수 있습니다. 이 변경 사항에 대한 자세한 내용은 GitHub의 PR 3476을 참조하세요.

`RetrieveDateTimeInUtc` 속성

DynamoDBContextConfig 클래스의 경우 `RetrieveDateTimeInUtc` 속성의 기본값이 로 변경되었습니다`true`.

`DynamoDBContextTableNamePrefix` 속성

AWSConfigsDynamoDB 클래스에서 `DynamoDBContextTableNamePrefix` 속성을 제거했습니다. 사용자는 `AWSConfigsDynamoDB.Context.TableNamePrefix` 대신를 호출해야 합니다.

EC2 관련 변경 사항

다음 변경 사항은 Amazon EC2에만 적용됩니다. 이들 중 대부분 또는 전부가 주요 변경 사항입니다.

항목을 보려면 열기

`GetDecryptedPassword`

Amazon EC2의 `GetDecryptedPassword` 확장이 AWSSDK.Extensions.EC2.DecryptPassword라는 별도의 확장 패키지로 이동되었습니다. 이 변경 사항은 OpenSSL 3을 지원하고 BouncyCastle.Cryptography에 종속되도록 하기 위한 것입니다.

이 패키지의 소스 코드는 https://github.com/aws/aws-sdk-net/tree/main/extensions/src/AWSSDK.Extensions.EC2.DecryptPassword의 GitHub에 있습니다.

Amazon EC2 IMDSv1 지원

인스턴스 메타데이터 서비스 버전 1(IMDSv1)에 대한 지원이 제거되었습니다. SDK의 V4는 IMDS에서 자격 증명 및 기타 메타데이터를 가져올 때 항상 인스턴스 메타데이터 서비스 버전 2(IMDSv2)를 사용합니다. IMDS에 대한 자세한 내용은 Amazon EC2 사용 설명서의 IMDS 사용을 참조하세요.

변경 또는 제거된 프로그래밍 요소

- 전체 `Amazon.EC2.Import` 네임스페이스와 코드가 제거되었습니다.
- Windows용 EC2 AMI를 조회하는 데 사용된 AMIs 유틸리티를 포함하는 전체 `Amazon.EC2.Util` 네임스페이스와 코드가 제거되었습니다.
- 더 이상 사용되지 않는 `IpRanges` 속성이 IpPermission 클래스에서 제거되었습니다. 대신 `Ipv4Ranges` 또는 `Ipv6Ranges` 속성을 사용합니다.
- EC2InstanceMetadata 클래스에서 더 이상 사용되지 않는 필드 `EC2_METADATA_SVC`, `EC2_METADATA_ROOT`, `EC2_USERDATA_ROOT`, 및 `EC2_DYNAMICDATA_ROOT`가 제거되었습니다`EC2_APITOKEN_URL`.

S3 관련 변경 사항

다음 변경 사항은 Amazon S3에만 적용됩니다. 이들 중 대부분 또는 전부가 주요 변경 사항입니다.

항목을 보려면 열기

AWS 리전 us-east-1

`us-east-1` 리전에 대해 구성된 Amazon S3 서비스 클라이언트는 더 이상 다른 리전의 버킷에 액세스할 수 없습니다. 버킷은 버킷이 있는 리전에 대해 구성된 S3 서비스 클라이언트를 사용하여 액세스해야 합니다.

이 변경 사항에 대한 자세한 내용은 블로그 게시물 V4의 AWS SDK for .NET V4를 참조하세요.

S3 암호화 클라이언트

`Amazon.S3.Encryption` 네임스페이스에 정의된 Amazon S3 암호화 클라이언트가 AWSSDK.S3 패키지에서 제거되었습니다. 이 클라이언트는 Amazon.Extensions.S3.Encryption이라는 자체 패키지로 이동되었으며 이에 대한 설명서는에 있습니다https://aws.github.io/amazon-s3-encryption-client-dotnet/api/Amazon.Extensions.S3.Encryption.html. 마이그레이션에 대한 내용은 the section called "S3 암호화 클라이언트 마이그레이션(V1에서 V2로)"을 참조하세요. S3 암호화에 대한 자세한 내용은 Amazon S3 암호화 클라이언트 개발자 안내서의 지원되는 암호화 알고리즘을 참조하세요.

에 대한 S3 태그 지정 지시문 **`CopyObject`**

`TaggingDirective` 속성은 `AmazonS3Client.CopyObject` 메서드에서 사용하는 CopyObjectRequest 클래스의 퍼블릭 속성으로 노출되었습니다. 이 속성은 CopyObject 작업에 정의된 Amazon S3 `x-amz-tagging-directive` 파라미터에 해당합니다.

태그 지정 지시문은 더 이상 COPY로 자동 설정되지 않습니다. 개발자가 태그 지정 지시문을 지정하지 않으면 S3 백엔드는 자동으로 COPY라고 가정하지만, 개발자가 속성을 null로 명시적으로 설정하면 값이 전혀 설정되지 않습니다.

S3 구성의 **`UseArnRegion`** 속성

`AWS_S3_USE_ARN_REGION` 환경 변수가 공유 AWS `config` 파일의 `s3_use_arn_region` 설정보다 우선하도록 Amazon.S3.AmazonS3Config 클래스의 `UseArnRegion` 속성이 업데이트되었습니다. 이러한 변수 및 설정에 대한 자세한 내용은 SDK 및 도구 참조 안내서의 설정 참조를 참조하세요. AWS SDKs

`CopyObject` 및 **`CopyPart`** 메서드의 선행 슬래시

Amazon S3 `CopyObject` 및 `CopyPart` 메서드에 대해 선행 슬래시가 더 이상 잘리지 않습니다. CopyObjectRequest 및 CopyPartRequest CopyPartRequest 클래스에서 `DisableTrimmingLeadingSlash` 속성이 제거되었습니다.

`DoesS3BucketExist...` 메서드

더 이상 사용되지 않는 `DoesS3BucketExist` 및 `DoesS3BucketExistAsync` 메서드는 ICoreAmazonS33 인터페이스를 구현하는 AmazonS3Util 클래스에서 제거되었습니다. 이러한 메서드는 항상 HTTP를 사용하기 때문에 제거되었습니다. 대신 DoesS3BucketExistV2 및 DoesS3BucketExistV2Async를 사용합니다.

SDK는 항상 SigV4를 사용합니다.

의 버전 4는 요청에 서명하기 위해 AWS SDK for .NET 항상 AWS 서명 버전 4(SigV4)를 사용합니다. 이 변경으로 인해 다음과 같은 관련 변경 사항이 발생합니다.

- AWSConfigsS3 클래스의 `UseSignatureVersion4` 속성이 제거되었습니다.
- Amazon.Runtime.ClientConfig 클래스의 `SignatureVersion` 속성이 제거되었습니다. 이 속성은 이전 버전과의 호환성을 위해 Amazon S3에서만 사용되었습니다.
- `RegionEndpoint.Endpoint` 클래스가 제거되었습니다. 여기에는 Amazon S3의 서명 버전을 재정의하는 데 사용된 `SignatureVersionOverride` 속성이 포함됩니다. 대신 서비스별 `client.DetermineServiceOperationEndPoint()` 메서드를 사용합니다.
- SigV4를 사용하도록 AmazonS3Util.PostUpload 및 SS3PostUploadSignedPolicy.GetSignedPolicy 메서드를 업데이트했습니다. 따라서가 `GetSignedPolicy` 이제 동일한 함수를 수행하기 때문에 `S3PostUploadSignedPolicy.GetSignedPolicyV4` 메서드가 제거되었습니다. 또한 `GetSignedPolicy`에는 리전 엔드포인트에 대한 세 번째 파라미터가 부여되었습니다.

`GetACL` 및 `PutACL` 메서드

AmazonS3Client 클래스의 `GetACL` 및 `PutACL` 메서드가 더 이상 사용되지 않는 것으로 표시되었습니다. 이러한 메서드의 기능에 액세스하려면 대신 , `GetBucketACL`, 및 새 메서드를 사용합니다`PutBucketACLGetObjectACLPutObjectACL`.

더 이상 사용되지 않는 프로그래밍 요소 제거

열거 값, 유형, 메서드, 네임스페이스 등을 포함하여 Amazon S3 구현의 여러 프로그래밍 요소가 SDK의 V4에서 제거되었습니다. 다음은 이전에 다루지 않은 경우 제거를 수용하기 위해 취할 수 있는 잠재적 단계와 함께 아래에 나열되어 있습니다.

- TransferUtilityUploadRequest 클래스에서 `DisableMD5Stream` 속성이 제거되었습니다. 대신 `DisableDefaultChecksumValidation` 속성을 사용합니다.

  또한 `CalculateContentMD5Header` 속성이 `TransferUtilityUploadRequest` 클래스에서 제거되었습니다. SDK는 기본적으로 체크섬을 계산하므로이 속성은 더 이상 필요하지 않습니다.
- CopyPartRequest 클래스에서 `ServerSideEncryptionMethod` 및 `ServerSideEncryptionKeyManagementServiceKeyId` 속성이 제거되었습니다. 대신 InitiateMultipartUploadRequest 클래스에서 이름이 동일한 속성을 사용합니다.이 속성은 AmazonS3Client 클래스의 일부 `InitiateMultipartUpload...` 메서드에서 사용됩니다.

- GetObjectResponse 클래스에서 Expires 속성이 제거되었습니다. 대신 ExpiresString 속성을 사용합니다. 문자열이 유효한 타임스탬프 형식이 아닐 수 있으므로 로 변환할 때 코드가 TryParse 메서드를 사용해야 합니다DateTime.
- SS3Region 열거에서 더 이상 사용되지 않는 AWS 리전 식별자가 제거되었습니다.
- Prefix 속성이 LifecycleRule 클래스에서 제거되었습니다. 대신 Filter 속성을 사용합니다.

  또한 NoncurrentVersionTransition 및 Transition 속성이 LifecycleRule 클래스에서 제거되었습니다. 대신 NoncurrentVersionTransitions 및 Transitions 컬렉션을 사용합니다.
- TopicConfiguration 클래스에서 Event 속성이 제거되었습니다. 대신 Events 컬렉션을 사용합니다.
- CalculateContentMD5Header 속성입니다. SDK는 기본적으로 체크섬을 계산하므로이 속성을 더 이상 설정할 필요가 없습니다.
- Bucket 속성이 SelectObjectContentRequest 클래스에서 제거되었습니다. 대신 BucketName 속성을 사용합니다.
- TransferUtilityConfig 클래스에서 NumberOfUploadThreads 속성이 제거되었습니다. 대신 ConcurrentServiceRequests 속성을 사용합니다.

제거된 프로그래밍 요소

열거 값, 유형, 메서드, 네임스페이스 등을 포함하여 SDK의 V4에서 여러 프로그래밍 요소가 제거되었습니다. 다음은 이전에 다루지 않은 경우 제거를 수용하기 위해 취할 수 있는 잠재적 단계와 함께 아래에 나열되어 있습니다.

항목을 보려면 열기

Amazon.Auth.AccessControlPolicy.ActionIdentifiers 네임스페이스

Amazon.Auth.AccessControlPolicy.ActionIdentifiers 네임스페이스가 제거되었습니다. 여기에는 IdentityandAccessManagementActionIdentifiers 클래스에 정의된 IAM 작업 식별자가 포함됩니다. 이러한 작업 식별자를 사용하는 코드는 작업 이름의 문자열 값을 사용하도록 변경해야 합니다.

자세한 내용은 IAM 사용 설명서의 the section called “JSON에서 관리형 정책 생성” 및 JSON 정책 개요를 참조하세요. https://docs.aws.amazon.com/IAM/latest/UserGuide/access_policies.html#access_policies-json

`ClientConfig` 클래스

Amazon.Runtime.ClientConfig 클래스는 AmazonS3Config와 같은 서비스 클라이언트 구성 클래스의 기본 클래스입니다. 이 클래스에서 다음 프로그래밍 요소가 제거되었습니다.

- `DetermineServiceURL` 및 `DetermineDnsSuffix` 메서드가 제거되었습니다. 대신 서비스 클라이언트의 `DetermineServiceOperationEndpoint` 메서드를 사용합니다. 예: AmazonS3Client.DetermineServiceOperationEndpoint.
- `ReadEntireResponse` 속성이 제거되었습니다. 대신 다음 중 하나를 사용합니다.
  - AWSConfigs.LoggingConfig 클래스의 `LogResponses` 속성입니다.
  - AmazonS3Config와 같은 클라이언트 구성의 `LogResponse` 속성입니다.

`Amazon.Runtime` 네임스페이스

Amazon.Runtime 네임스페이스가 다음과 같이 업데이트되었습니다.

- 더 이상 사용되지 않는 `ECSTaskCredentials` 클래스가 네임스페이스에서 제거되었습니다. 대신 Amazon EKS Pod Identity를 지원하는 GenericContainerCredentials 공급자를 사용합니다.
- 더 이상 사용되지 않는 `StoredProfileAWSCredentials` 및 `StoredProfileCredentials` 클래스가 네임스페이스에서 제거되었습니다. 대신 Amazon.Runtime.CredentialManagement 네임스페이스의 NetSDKCredentialsFile 또는 SharedCredentialsFile 클래스를 사용합니다. CredentialManagement
- SSOAWSCredentials 클래스의 더 이상 사용되지 않는 `HasCachedAccessTokenAvailable` 메서드가 네임스페이스에서 제거되었습니다.
- 더 이상 사용되지 않는 `EnvironmentAWSCredentials` 클래스가 네임스페이스에서 제거되었습니다. 대신 AppConfigAWSCredentials 클래스를 사용합니다.
- 더 이상 사용되지 않는 `StoredProfileFederatedCredentials` 클래스가 네임스페이스에서 제거되었습니다. 대신 FederatedAWSCredentials 클래스를 사용합니다.
- 더 이상 사용되지 않는 클래스인 `EnvironmentVariableAWSEndpointDiscoveryEnabled`, `ProfileAWSEndpointDiscoveryEnabled`및가 네임스페이스에서 제거되었습니다`FallbackEndpointDiscoveryEnabledFactory`.
- 더 이상 사용되지 않는 `UseSigV4` 속성이 AmazonWebServiceRequest 클래스에서 제거되었습니다. 대신 `SignatureVersion` 속성을 사용합니다.
- `Amazon.Runtime.Internal.Util` 네임스페이스의 `ProfileIniFile` 클래스에는 라는 오버로드된 메서드가 있습니다`TryGetSection`. 에 대한 `out` 파라미터를 지원하지 않는 메서드의 버전이 클래스에서 제거`nestedProperties`되었습니다.

- `Amazon.Runtime.Internal.Auth` 네임스페이스에서 더 이상 사용되지 않는 `EventBridgeSigner` 클래스가 제거되었습니다.
- 더 이상 사용되지 않는 `Parameters` 사전 속성이 WebServiceRequestEventArgs 클래스에서 제거되었습니다. 대신 `ParameteCollection` 속성을 사용합니다.

BouncyCastle

BouncyCastle의 소스 복사본이 SDK의 V4에서 제거되었습니다.

`StoredProfileSAMLCredentials` 클래스

Amazon.SecurityToken.SAML 네임스페이스에서 더 이상 사용되지 않는 `StoredProfileSAMLCredentials` 클래스가 제거되었습니다. 대신 Amazon.Runtime 네임스페이스에서 FederatedAWSCredentials 클래스를 사용합니다.

`AWSSDKUtils` 클래스

AWSSDKUtils 클래스에서 `ResolveResourcePath`, `ProtectEncodedSlashUrlEncode`및 메서드가 제거되었습니다`ConvertToUnixEpochMilliSeconds`.

`ProfileManager` 클래스

더 이상 사용되지 않는 `ProfileManager` 클래스가 Amazon.Util 네임스페이스에서 제거되었습니다. 대신 Amazon.Runtime.CredentialManagement 네임스페이스의 NetSDKCredentialsFile 또는 SharedCredentialsFile 클래스를 사용합니다. CredentialManagement

`AWSConfigs` 클래스

AWSConfigs 클래스에서 더 이상 사용되지 않는 속성인 `Logging`, `ResponseLogging`및가 제거되었습니다`LogMetrics`. 대신 `LoggingConfig` 속성을 사용합니다.

`ConditionFactory` 클래스

ConditionFactory 클래스에서 다음 서명이 있는 메서드가 제거되었습니다`NewCondition(ConditionFactory.DateComparisonType, DateTime)`. 대신 NewConditionUtc 메서드를 사용합니다.

Amazon CloudFront 유틸리티

더 이상 사용되지 않는 `Amazon.CloudFront.Util` 네임스페이스와 `AmazonCloudFrontUtil` 클래스가 제거되었습니다.

AWS IoT

ListPrincipalThingsResponse 클래스에서는 페이지 매김을 위해 NextToken 재정의를 위한 레거시 사용자 지정이 제거되었습니다.

AWS Lambda

이름이 혼란스러웠기 때문에 AmazonLambdaClient 클래스의 다음 Invoke... 메서드가 제거되었습니다.

- 서명이 인 V3 메서드가 제거되었습니다InvokeAsyncResponse InvokeAsync(InvokeAsyncRequest). SDK의 V3에서 동기식 메서드입니다. 대신 InvokeResponse Invoke(InvokeRequest) (동기 처리의 경우) 또는 Task InvokeAsync(InvokeRequest, CancellationToken) (비동기 처리의 경우)를 사용합니다.
- 서명이 인 V3 메서드가 제거되었습니다Task InvokeAsyncAsync(InvokeAsyncRequest, CancellationToken). SDK의 V3에서 비동기식 메서드입니다. 대신 Task InvokeAsync(InvokeRequest, CancellationToken)을 사용하세요.

Amazon SageMaker 런타임

PayloadPart 클래스에 대한 더 이상 사용되지 않는 생성자가 제거되었습니다.

.NET Standard 1.3에서 마이그레이션

2019년 6월 27일에 Microsoft는 .NET Core 1.0 및 .NET Core 1.1 버전에 대한 지원을 종료했습니다. 이 발표 AWS 에 따라는 2020년 12월 31일에 .NET Standard 1.3 AWS SDK for .NET 에 대한 지원을 종료했습니다.

AWS 는 2020년 10월 1일까지 AWS SDK for .NET 대상 .NET Standard 1.3에 대한 서비스 업데이트 및 보안 수정 사항을 계속 제공했습니다. 이 날짜 이후에는 .NET Standard 1.3 대상이 유지 관리 모드로 전환되어 새 업데이트가 릴리스되지 않았으며 중요한 버그 수정 및 보안 패치만 AWS 적용되었습니다.

2020년 12월 31일에의 .NET Standard 1.3 지원이 종료 AWS SDK for .NET 되었습니다. 이 날짜 이후에는 버그 수정 또는 보안 패치가 적용되지 않았습니다. 해당 대상으로 빌드된 아티팩트는 NuGet에서 다운로드할 수 있습니다.

알아야 할 내용

- .NET Framework를 사용하는 애플리케이션을 실행하는 경우 영향을 받지 않습니다.

- .NET Core 2.0 이상을 사용하는 애플리케이션을 실행하는 경우 영향을 받지 않습니다.

- .NET Core 1.0 또는 .NET Core 1.1을 사용하는 애플리케이션을 실행하는 경우 Microsoft 마이그레이션 지침에 따라 애플리케이션을 최신 버전의 .NET Core로 마이그레이션하십시오. 최소한 .NET Core 3.1을 권장합니다.

- 현재 업그레이드할 수 없는 비즈니스 크리티컬 애플리케이션을 실행하는 경우 현재 버전의 AWS SDK for .NET을 계속 사용할 수 있습니다.

질문이나 우려 사항이 있는 경우 AWS Support에 문의하십시오.

이 AWS 제품 또는 서비스에 대한 보안

Amazon Web Services(AWS)에서 가장 우선순위가 높은 것이 클라우드 보안입니다. AWS 고객으로서 여러분은 가장 높은 보안 요구 사항을 충족하기 위해 설계된 데이터 센터 및 네트워크 아키텍처의 혜택을 받게 됩니다. 보안은 AWS 와 사용자 간의 공동 책임입니다. 공동 책임 모델은 이 사항을 클라우드 내 보안 및 클라우드의 보안으로 설명합니다.

클라우드 보안 - AWS 는 클라우드에서 제공되는 모든 서비스를 실행하는 인프라를 보호하고 안전하게 사용할 수 있는 서비스를 AWS 제공할 책임이 있습니다. 당사의 보안 책임은에서 최우선 순위이며 AWS, 타사 감사자는 AWS 규정 준수 프로그램의 일환으로 보안의 효과를 정기적으로 테스트하고 검증합니다.

클라우드의 보안 - 사용자의 책임은 사용 중인 AWS 서비스와 데이터의 민감도, 조직의 요구 사항, 관련 법률 및 규정을 비롯한 기타 요인에 따라 결정됩니다.

이 AWS 제품 또는 서비스는 지원하는 특정 Amazon Web Services(AWS) 서비스를 통해 공동 책임 모델을 따릅니다. AWS 서비스 보안 정보는 AWS 서비스 보안 설명서 페이지 및 규정 AWSAWS 준수 프로그램의 규정 준수 작업 범위에 속하는 서비스를 참조하세요.

주제

- 이 AWS 제품 또는 서비스의 데이터 보호
- 자격 증명 및 액세스 관리
- 이 AWS 제품 또는 서비스에 대한 규정 준수 검증
- 이 AWS 제품 또는 서비스에 대한 복원력
- 이 AWS 제품 또는 서비스에 대한 인프라 보안
- 에서 최소 TLS 버전 적용 AWS SDK for .NET
- S3 암호화 클라이언트 마이그레이션(V1에서 V2로)
- S3 암호화 클라이언트 마이그레이션(V2에서 V4로)

이 AWS 제품 또는 서비스의 데이터 보호

AWS 공동 책임 모델이 AWS 제품 또는 서비스의 데이터 보호에 적용됩니다. 이 모델에 설명된 대로 AWS 는 모든를 실행하는 글로벌 인프라를 보호할 책임이 있습니다 AWS 클라우드. 사용자는 이 인프라에 호스팅되는 콘텐츠에 대한 통제 권한을 유지할 책임이 있습니다. 사용하는 AWS 서비스 의 보안 구성과 관리 태스크에 대한 책임도 사용자에게 있습니다. 데이터 프라이버시에 관한 자세한 내용은 데

이터 프라이버시 FAQ를 참조하세요. 유럽의 데이터 보호에 대한 자세한 내용은 AWS 보안 블로그의 AWS 공동 책임 모델 및 GDPR 블로그 게시물을 참조하세요.

데이터 보호를 위해 자격 증명을 보호하고 AWS 계정 AWS IAM Identity Center 또는 AWS Identity and Access Management (IAM)를 사용하여 개별 사용자를 설정하는 것이 좋습니다. 이렇게 하면 개별 사용자에게 자신의 직무를 충실히 이행하는 데 필요한 권한만 부여됩니다. 또한 다음과 같은 방법으로 데이터를 보호하는 것이 좋습니다.

- 각 계정에 다중 인증(MFA)을 사용합니다.
- SSL/TLS를 사용하여 AWS 리소스와 통신합니다. TLS 1.2는 필수이며 TLS 1.3을 권장합니다.
- 를 사용하여 API 및 사용자 활동 로깅을 설정합니다 AWS CloudTrail. CloudTrail 추적을 사용하여 AWS 활동을 캡처하는 방법에 대한 자세한 내용은 AWS CloudTrail 사용 설명서의 CloudTrail 추적 작업을 참조하세요.
- 내부의 모든 기본 보안 제어와 함께 AWS 암호화 솔루션을 사용합니다 AWS 서비스.
- Amazon S3에 저장된 민감한 데이터를 검색하고 보호하는 데 도움이 되는 Amazon Macie와 같은 고급 관리형 보안 서비스를 사용합니다.
- 명령줄 인터페이스 또는 API를 AWS 통해에 액세스할 때 FIPS 140-3 검증 암호화 모듈이 필요한 경우 FIPS 엔드포인트를 사용합니다. 사용 가능한 FIPS 엔드포인트에 대한 자세한 내용은 연방 정보 처리 표준(FIPS) 140-3을 참조하세요.

고객의 이메일 주소와 같은 기밀 정보나 중요한 정보는 태그나 이름 필드와 같은 자유 형식 텍스트 필드에 입력하지 않는 것이 좋습니다. 여기에는이 AWS 제품 또는 서비스 또는 기타 AWS 서비스 에서 콘솔 AWS CLI, API 또는 AWS SDKs를 사용하여 작업하는 경우가 포함됩니다. 이름에 사용되는 태그 또는 자유 형식 텍스트 필드에 입력하는 모든 데이터는 청구 또는 진단 로그에 사용될 수 있습니다. 외부 서버에 URL을 제공할 때 해당 서버에 대한 요청을 검증하기 위해 보안 인증 정보를 URL에 포함시켜서는 안 됩니다.

자격 증명 및 액세스 관리

AWS Identity and Access Management (IAM)는 관리자가 AWS 리소스에 대한 액세스를 안전하게 제어하는 데 도움이 AWS 서비스 되는 입니다. IAM 관리자는 누가 AWS 리소스를 사용할 수 있는 인증(로그인) 및 권한(권한 있음)을 받을 수 있는지 제어합니다. IAM은 추가 비용 없이 사용할 수 AWS 서비스 있는 입니다.

주제

- 대상

- ID를 통한 인증
- 정책을 사용하여 액세스 관리
- 에서 IAM AWS 서비스 을 사용하는 방법
- AWS 자격 증명 및 액세스 문제 해결

대상

AWS Identity and Access Management (IAM)를 사용하는 방법은에서 수행하는 작업에 따라 다릅니다 AWS.

서비스 사용자 - AWS 서비스 를 사용하여 작업을 수행하는 경우 필요한 자격 증명과 권한을 관리자가 제공합니다. 더 많은 AWS 기능을 사용하여 작업을 수행하게 되면 추가 권한이 필요할 수 있습니다. 액세스 권한 관리 방법을 이해하면 관리자에게 올바른 권한을 요청하는 데 도움이 됩니다. 에서 기능에 액세스할 수 없는 경우 AWS 자격 증명 및 액세스 문제 해결 또는 사용 중인의 사용 설명서를 AWS참조 AWS 서비스 하세요.

서비스 관리자 - 회사에서 AWS 리소스를 책임지고 있는 경우에 대한 전체 액세스 권한을 가지고 있을 것입니다 AWS. 서비스 관리자는 서비스 사용자가 액세스해야 하는 AWS 기능과 리소스를 결정합니다. 그런 다음 IAM 관리자에게 요청을 제출하여 서비스 사용자의 권한을 변경해야 합니다. 이 페이지의 정보를 검토하여 IAM의 기본 개념을 이해하세요. 회사가 IAM을 사용하는 방법에 대해 자세히 알아보려면 사용 중인의 AWS 서비스 사용 설명서를 AWS참조하세요.

IAM 관리자 - IAM 관리자라면 AWS에 대한 액세스 권한 관리 정책 작성 방법을 자세히 알고 싶을 것입니다. IAM에서 사용할 수 있는 자격 AWS 증명 기반 정책 예제를 보려면 사용 중인의 사용 설명서를 참조 AWS 서비스 하세요.

ID를 통한 인증

인증은 자격 증명 자격 증명을 AWS 사용하여에 로그인하는 방법입니다. AWS 계정 루트 사용자, IAM 사용자 또는 IAM 역할을 수임하여 인증되어야 합니다.

AWS IAM Identity Center (IAM Identity Center), Single Sign-On 인증 또는 Google/Facebook 자격 증명과 같은 자격 증명 소스의 자격 증명을 사용하여 페더레이션 자격 증명으로 로그인할 수 있습니다. 로그인하는 방법에 대한 자세한 내용은 AWS 로그인 사용 설명서의 AWS 계정에 로그인하는 방법 섹션을 참조하세요.

프로그래밍 방식 액세스를 위해는 요청에 암호화 방식으로 서명할 수 있는 SDK 및 CLI를 AWS 제공합니다. 자세한 내용은 IAM 사용 설명서의 API 요청용AWS Signature Version 4 섹션을 참조하세요.

AWS 계정 루트 사용자

를 생성할 때 모든 AWS 서비스 및 리소스에 대한 완전한 액세스 권한이 있는 AWS 계정 theroot 사용자라는 하나의 로그인 자격 증명으로 AWS 계정시작합니다. 일상적인 태스크에 루트 사용자를 사용하지 않을 것을 강력히 권장합니다. 루트 사용자 자격 증명이 필요한 작업은 IAM 사용 설명서의 루트 사용자 자격 증명이 필요한 작업 섹션을 참조하세요.

페더레이션 ID

가장 좋은 방법은 인간 사용자에게 자격 증명 공급자와의 페더레이션을 사용하여 임시 자격 증명을 AWS 서비스 사용하여에 액세스하도록 요구하는 것입니다.

페더레이션 자격 증명은 엔터프라이즈 디렉터리, 웹 자격 증명 공급자 또는 자격 증명 소스의 자격 증명을 AWS 서비스 사용하여 Directory Service 에 액세스하는 사용자입니다. 페더레이션 ID는 임시 자격 증명을 제공하는 역할을 수임합니다.

중앙 집중식 액세스 관리를 위해 AWS IAM Identity Center를 추천합니다. 자세한 정보는 AWS IAM Identity Center 사용 설명서의 What is IAM Identity Center?를 참조하세요.

IAM 사용자 및 그룹

IAM 사용자는 단일 개인 또는 애플리케이션에 대한 특정 권한을 가진 ID입니다. 장기 자격 증명이 있는 IAM 사용자 대신 임시 자격 증명을 사용하는 것이 좋습니다. 자세한 내용은 IAM 사용 설명서의 자격 증명 공급자와의 페더레이션을 사용하여 임시 자격 증명을 AWS 사용하여에 액세스하도록 인간 사용자에게 요구하기를 참조하세요.

IAM 그룹은 IAM 사용자 모음을 지정하고 대규모 사용자 집합에 대한 관리 권한을 더 쉽게 만듭니다. 자세한 내용은 IAM 사용 설명서의 IAM 사용자 사용 사례 섹션을 참조하세요.

IAM 역할

IAM 역할은 임시 자격 증명을 제공하는 특정 권한이 있는 자격 증명입니다. 사용자에서 IAM 역할(콘솔)로 전환하거나 또는 API 작업을 호출하여 역할을 수임할 수 있습니다. AWS CLI AWS 자세한 내용은 IAM 사용 설명서의 역할 수임 방법을 참조하세요.

IAM 역할은 페더레이션 사용자 액세스, 임시 IAM 사용자 권한, 교차 계정 액세스, 교차 서비스 액세스 및 Amazon EC2에서 실행되는 애플리케이션에 유용합니다. 자세한 내용은 IAM 사용 설명서의 교차 계정 리소스 액세스를 참조하세요.

정책을 사용하여 액세스 관리

정책을 AWS 생성하고 자격 증명 또는 리소스에 연결하여 AWS 에서 액세스를 제어합니다. 정책은 자격 증명 또는 리소스와 연결될 때 권한을 정의합니다.는 보안 주체가 요청할 때 이러한 정책을 AWS 평가합니다. 대부분의 정책은에 JSON 문서 AWS 로 저장됩니다. JSON 정책 문서에 대한 자세한 내용은 IAM 사용 설명서의 JSON 정책 개요 섹션을 참조하세요.

정책을 사용하여 관리자는 어떤 보안 주체가 어떤 리소스에 대해 어떤 조건에서 작업을 수행할 수 있는지 정의하여 누가 무엇을 액세스할 수 있는지 지정합니다.

기본적으로 사용자 및 역할에는 어떠한 권한도 없습니다. IAM 관리자는 IAM 정책을 생성하고 사용자가 수임할 수 있는 역할에 추가합니다. IAM 정책은 작업을 수행하기 위해 사용하는 방법과 관계없이 작업에 대한 권한을 정의합니다.

ID 기반 정책

ID 기반 정책은 ID(사용자, 사용자 그룹 또는 역할)에 연결하는 JSON 권한 정책 문서입니다. 이러한 정책은 자격 증명이 수행할 수 있는 작업, 대상 리소스 및 이에 관한 조건을 제어합니다. ID 기반 정책을 생성하는 방법을 알아보려면 IAM 사용 설명서에서 고객 관리형 정책으로 사용자 지정 IAM 권한 정의를 참조하세요.

ID 기반 정책은 인라인 정책(단일 ID에 직접 포함) 또는 관리형 정책(여러 ID에 연결된 독립 실행형 정책)일 수 있습니다. 관리형 정책 또는 인라인 정책을 선택하는 방법을 알아보려면 IAM 사용 설명서의 관리형 정책 및 인라인 정책 중에서 선택 섹션을 참조하세요.

리소스 기반 정책

리소스 기반 정책은 리소스에 연결하는 JSON 정책 설명서입니다. 예를 들어 IAM 역할 신뢰 정책 및 Amazon S3 버킷 정책이 있습니다. 리소스 기반 정책을 지원하는 서비스에서 서비스 관리자는 이러한 정책을 사용하여 특정 리소스에 대한 액세스를 통제할 수 있습니다. 리소스 기반 정책에서 보안 주체를 지정해야 합니다.

리소스 기반 정책은 해당 서비스에 있는 인라인 정책입니다. 리소스 기반 정책에서는 IAM의 AWS 관리형 정책을 사용할 수 없습니다.

액세스 제어 목록(ACL)

액세스 제어 목록(ACL)은 어떤 위탁자(계정 멤버, 사용자 또는 역할)가 리소스에 액세스할 수 있는 권한을 가지고 있는지를 제어합니다. ACL은 JSON 정책 문서 형식을 사용하지 않지만 리소스 기반 정책과 유사합니다.

Amazon S3 AWS WAF및 Amazon VPC는 ACLs. ACL에 관한 자세한 내용은 Amazon Simple Storage Service 개발자 가이드의 액세스 제어 목록(ACL) 개요를 참조하세요.

기타 정책 타입

AWS 는 보다 일반적인 정책 유형에서 부여한 최대 권한을 설정할 수 있는 추가 정책 유형을 지원합니다.

- 권한 경계 - ID 기반 정책에서 IAM 엔터티에 부여할 수 있는 최대 권한을 설정합니다. 자세한 정보는 IAM 사용 설명서의 IAM 엔터티의 권한 범위를 참조하세요.
- 서비스 제어 정책(SCP) - AWS Organizations내 조직 또는 조직 단위에 대한 최대 권한을 지정합니다. 자세한 내용은AWS Organizations 사용 설명서의 서비스 제어 정책을 참조하세요.
- 리소스 제어 정책(RCP) – 계정의 리소스에 사용할 수 있는 최대 권한을 설정합니다. 자세한 내용은 AWS Organizations 사용 설명서의 리소스 제어 정책(RCP)을 참조하세요.
- 세션 정책 – 역할 또는 페더레이션 사용자에 대해 임시 세션을 프로그래밍 방식으로 생성할 때 파라미터로 전달하는 고급 정책입니다. 자세한 내용은 IAM 사용 설명서의 세션 정책을 참조하세요.

여러 정책 유형

여러 정책 유형이 요청에 적용되는 경우, 결과 권한은 이해하기가 더 복잡합니다. 에서 여러 정책 유형이 관련될 때 요청을 허용할지 여부를 AWS 결정하는 방법을 알아보려면 IAM 사용 설명서의 정책 평가 로직을 참조하세요.

에서 IAM AWS 서비스 을 사용하는 방법

가 대부분의 IAM 기능을 AWS 서비스 사용하는 방법을 개괄적으로 알아보려면 IAM 사용 설명서의 AWS IAM으로 작업하는 서비스를 참조하세요.

IAM AWS 서비스 에서 특정를 사용하는 방법을 알아보려면 관련 서비스 사용 설명서의 보안 섹션을 참조하세요.

AWS 자격 증명 및 액세스 문제 해결

다음 정보를 사용하여 및 IAM으로 작업할 때 발생할 수 있는 일반적인 문제를 진단 AWS 하고 수정할 수 있습니다.

주제

- 에서 작업을 수행할 권한이 없음 AWS

- iam:PassRole을 수행하도록 인증되지 않음
- 내 외부의 사람이 내 AWS 리소스에 액세스 AWS 계정 하도록 허용하고 싶습니다.

에서 작업을 수행할 권한이 없음 AWS

작업을 수행할 권한이 없다는 오류가 표시되면 작업을 수행할 수 있도록 정책을 업데이트해야 합니다.

다음의 예제 오류는 mateojackson IAM 사용자가 콘솔을 사용하여 가상 *my-example-widget* 리소스에 대한 세부 정보를 보려고 하지만 가상 awes:*GetWidget* 권한이 없을 때 발생합니다.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 awes:GetWidget on resource: my-example-widget
```

이 경우, awes:*GetWidget* 작업을 사용하여 *my-example-widget* 리소스에 액세스할 수 있도록 mateojackson 사용자 정책을 업데이트해야 합니다.

도움이 필요한 경우 AWS 관리자에게 문의하세요. 관리자는 로그인 자격 증명을 제공한 사람입니다.

iam:PassRole을 수행하도록 인증되지 않음

iam:PassRole 작업을 수행할 수 있는 권한이 없다는 오류가 수신되면 AWS에 역할을 전달할 수 있도록 정책을 업데이트해야 합니다.

일부 AWS 서비스 에서는 새 서비스 역할 또는 서비스 연결 역할을 생성하는 대신 기존 역할을 해당 서비스에 전달할 수 있습니다. 이렇게 하려면 역할을 서비스에 전달할 권한이 있어야 합니다.

다음 예 오류는 marymajor라는 IAM 사용자가 콘솔을 사용하여 AWS에서 작업을 수행하려고 하는 경우에 발생합니다. 하지만 작업을 수행하려면 서비스 역할이 부여한 권한이 서비스에 있어야 합니다. Mary는 서비스에 역할을 전달할 권한이 없습니다.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

이 경우, Mary가 iam:PassRole작업을 수행할 수 있도록 Mary의 정책을 업데이트해야 합니다.

도움이 필요한 경우 AWS 관리자에게 문의하세요. 관리자는 로그인 자격 증명을 제공한 사람입니다.

내 외부의 사람이 내 AWS 리소스에 액세스 AWS 계정 하도록 허용하고 싶습니다.

다른 계정의 사용자 또는 조직 외부의 사람이 리소스에 액세스할 때 사용할 수 있는 역할을 생성할 수 있습니다. 역할을 수임할 신뢰할 수 있는 사람을 지정할 수 있습니다. 리소스 기반 정책 또는 액세스 제

어 목록(ACL)을 지원하는 서비스의 경우, 이러한 정책을 사용하여 다른 사람에게 리소스에 대한 액세스 권한을 부여할 수 있습니다.

자세한 내용은 다음을 참조하세요.

- 에서 이러한 기능을 AWS 지원하는지 여부를 알아보려면 섹션을 참조하세요에서 IAM AWS 서비스을 사용하는 방법.
- 소유 AWS 계정 한의 리소스에 대한 액세스 권한을 제공하는 방법을 알아보려면 IAM 사용 설명서의 소유한 다른의 IAM 사용자에게 액세스 권한 제공을 참조 AWS 계정 하세요.
- 타사에 리소스에 대한 액세스 권한을 제공하는 방법을 알아보려면 IAM 사용 설명서의 타사가 AWS 계정 소유한에 대한 액세스 권한 제공을 AWS 계정참조하세요.
- ID 페더레이션을 통해 액세스 권한을 제공하는 방법을 알아보려면 IAM 사용 설명서의 외부에서 인증된 사용자에게 액세스 권한 제공(ID 페더레이션)을 참조하세요.
- 크로스 계정 액세스에 대한 역할과 리소스 기반 정책 사용의 차이점을 알아보려면 IAM 사용 설명서의 IAM의 크로스 계정 리소스 액세스를 참조하세요.

이 AWS 제품 또는 서비스에 대한 규정 준수 검증

AWS 서비스 가 특정 규정 준수 프로그램의 범위 내에 있는지 알아보려면 AWS 서비스 규정 준수 프로그램 범위 내를 참조하고 관심 있는 규정 준수 프로그램을 선택합니다. 일반 정보는 AWS 규정 준수 프로그램.

를 사용하여 타사 감사 보고서를 다운로드할 수 있습니다 AWS Artifact. 자세한 내용은 에서 보고서 다운로드 AWS Artifact.

사용 시 규정 준수 책임은 데이터의 민감도, 회사의 규정 준수 목표, 관련 법률 및 규정에 따라 AWS 서비스 결정됩니다. 사용 시 규정 준수 책임에 대한 자세한 내용은 AWS 보안 설명서를 AWS 서비스참조하세요.

이 AWS 제품 또는 서비스는 지원하는 특정 Amazon Web Services(AWS) 서비스를 통해 공동 책임 모델을 따릅니다. AWS 서비스 보안 정보는 AWS 서비스 보안 설명서 페이지 및 규정 AWSAWS 준수 프로그램의 규정 준수 작업 범위에 속하는 서비스를 참조하세요.

이 AWS 제품 또는 서비스에 대한 복원력

AWS 글로벌 인프라는 AWS 리전 및 가용 영역을 중심으로 구축됩니다.

AWS 리전 는 물리적으로 분리되고 격리된 여러 가용 영역을 제공하며,이 가용 영역은 지연 시간이 짧고 처리량이 높으며 중복성이 높은 네트워킹과 연결됩니다.

가용 영역을 사용하면 중단 없이 영역 간에 자동으로 장애 극복 조치가 이루어지는 애플리케이션 및 데이터베이스를 설계하고 운영할 수 있습니다. 가용 영역은 기존의 단일 또는 다중 데이터 센터 인프라보다 가용성, 내결함성, 확장성이 뛰어납니다.

AWS 리전 및 가용 영역에 대한 자세한 내용은 AWS 글로벌 인프라를 참조하세요.

이 AWS 제품 또는 서비스는 지원하는 특정 Amazon Web Services(AWS) 서비스를 통해 공동 책임 모델을 따릅니다. AWS 서비스 보안 정보는 AWS 서비스 보안 설명서 페이지 및 규정 AWSAWS 준수 프로그램의 규정 준수 작업 범위에 속하는 서비스를 참조하세요.

이 AWS 제품 또는 서비스에 대한 인프라 보안

이 AWS 제품 또는 서비스는 관리형 서비스를 사용하므로 글로벌 네트워크 보안으로 AWS 보호됩니다. AWS 보안 서비스 및가 인프라를 AWS 보호하는 방법에 대한 자세한 내용은 AWS 클라우드 보안을 참조하세요. 인프라 보안 모범 사례를 사용하여 AWS 환경을 설계하려면 보안 원칙 AWS Well-Architected Framework의 인프라 보호를 참조하세요.

AWS 에서 게시한 API 호출을 사용하여 네트워크를 통해이 AWS 제품 또는 서비스에 액세스합니다. 클라이언트는 다음을 지원해야 합니다.

- Transport Layer Security(TLS). TLS 1.2는 필수이며 TLS 1.3을 권장합니다.
- DHE(Ephemeral Diffie-Hellman) 또는 ECDHE(Elliptic Curve Ephemeral Diffie-Hellman)와 같은 완전 전송 보안(PFS)이 포함된 암호 제품군. Java 7 이상의 최신 시스템은 대부분 이러한 모드를 지원합니다.

또한 요청은 액세스 키 ID 및 IAM 위탁자와 관련된 시크릿 액세스 키를 사용하여 서명해야 합니다. 또는 AWS Security Token Service(AWS STS)를 사용하여 임시 보안 자격 증명을 생성하여 요청에 서명할 수 있습니다.

이 AWS 제품 또는 서비스는 지원하는 특정 Amazon Web Services(AWS) 서비스를 통해 공동 책임 모델을 따릅니다. AWS 서비스 보안 정보는 AWS 서비스 보안 설명서 페이지 및 규정 AWSAWS 준수 프로그램의 규정 준수 노력 범위에 속하는 서비스를 참조하세요.

에서 최소 TLS 버전 적용 AWS SDK for .NET

AWS 서비스와 통신할 때 보안을 강화하려면 TLS 1.2 이상을 사용하도록 AWS SDK for .NET 를 구성해야 합니다.

AWS SDK for .NET 는 기본 .NET 런타임을 사용하여 사용할 보안 프로토콜을 결정합니다. 기본적으로 현재 버전의 .NET은 운영 체제에서 지원하는 최신 구성 프로토콜을 사용합니다. 애플리케이션에서 이 SDK 동작을 재정의할 수 있지만 그렇게 하지 않는 것이 좋습니다.

.NET Core

기본적으로 .NET Core는 운영 체제에서 지원하는 최신 구성 프로토콜을 사용합니다. AWS SDK for .NET 에서는 이를 재정의하는 메커니즘을 제공하지 않습니다.

.NET Core 2.1 이전 버전을 사용하는 경우 .NET Core 버전을 업그레이드할 것을 적극 권장합니다.

각 운영 체제에 대한 자세한 내용은 다음을 참조하십시오.

Windows

최신 Windows 배포판에는 TLS 1.2 지원이 기본적으로 활성화되어 있습니다. Windows 7 SP1 또는 Windows Server 2008 R2 SP1에서 실행하는 경우 https://learn.microsoft.com/en-us/windows-server/security/tls/tls-registry-settings#tls-12에 설명된 대로 레지스트리에서 TLS 1.2 지원이 활성화되어 있는지 확인해야 합니다. 이전 배포를 실행 중인 경우 운영 체제를 업그레이드해야 합니다. Windows의 TLS 1.3 지원에 대한 자세한 내용은 최신 Microsoft 문서에서 필요한 최소 클라이언트 또는 서버 버전을 확인하세요.

macOS

.NET Core 2.1 이상을 실행하는 경우 TLS 1.2가 기본적으로 활성화되어 있습니다. TLS 1.2는 OS X Mavericks v10.9 이상에서 지원됩니다. .NET Core 버전 2.1 이상에는 설명된 대로 최신 버전의 macOS가 필요합니까? https://learn.microsoft.com/en-us/dotnet/core/install/windows?tabs=net80&pivots=os-macos.

.NET Core 1.0의 경우 .NET Core가 별도로 설치해야 하는 종속 프로그램인 OpenSSL을 macOS에서 사용합니다. OpenSSL은 버전 1.0.1에서 TLS 1.2에 대한 지원을 추가했으며 버전 1.1.1에서 TLS 1.3에 대한 지원을 추가했습니다.

Linux

Linux에서는 .NET Core에 OpenSSL이 필요합니다. OpenSSL은 많은 Linux 배포판에 번들로 함께 제공되는데 별도로 설치할 수도 있습니다. OpenSSL은 버전 1.0.1에서 TLS 1.2에 대한 지원을 추가했으며 버전 1.1.1에서 TLS 1.3에 대한 지원을 추가했습니다. 최신 버전의 .NET Core(2.1 이상)를 사용하며 패키지 관리자를 설치한 경우 최신 버전의 OpenSSL이 설치되었을 가능성이 큽니다.

터미널에서 **`openssl version`**을 실행하여 버전이 1.0.1 이상인지 확인할 수 있습니다.

.NET Framework

최신 버전의 .NET Framework (4.7 이상) 및 최신 버전의 Windows(클라이언트의 경우 Windows 8 이상, 서버의 경우 Windows Server 2012 이상)를 실행하는 경우 TLS 1.2가 기본적으로 활성화되어 있고 사용됩니다.

운영 체제 설정(.NET Framework 3.5~4.5.2)을 사용하지 않는 .NET Framework 런타임을 사용하는 경우 AWS SDK for .NET 는 지원되는 프로토콜에 TLS 1.1 및 TLS 1.2에 대한 지원을 추가하려고 시도합니다. .NET Framework 3.5를 사용하는 경우 다음과 같이 적절한 핫 패치가 설치된 경우에만 이 시도가 성공합니다.

- Windows 10 버전 1511 및 Windows Server 2016 – KB3156421
- Windows 8.1 및 Windows Server 2012 R2 – KB3154520
- Windows Server 2012 – KB3154519
- Windows 7 SP1 및 Server 2008 R2 SP1 – KB3154518

> **Warning**
>
> 2024년 8월 15일에는 .NET Framework 3.5에 대한 지원을 AWS SDK for .NET 종료하고 최소 .NET Framework 버전을 4.7.2로 변경했습니다. 자세한 내용은 블로그 게시물 AWS SDK for .NET의 .NET Framework 3.5 및 4.5 대상에 대한 중요 변경 사항을 참조하세요.

애플리케이션이 최신 버전의 .NET Framework를 Windows 7 SP1 또는 Windows Server 2008 R2 SP1에서 실행하는 경우 https://learn.microsoft.com/en-us/windows-server/security/tls/tls-registry-settings#tls-12에 설명된 대로 레지스트리에서 TLS 1.2 지원이 활성화되어 있는지 확인해야 합니다. 이보다 최신 버전의 Windows에서는 TLS 1.2가 기본적으로 활성화되어 있습니다.

.NET Framework에서 TLS를 사용하는 방법에 대한 자세한 모범 사례는 https://learn.microsoft.com/en-us/dotnet/framework/network-programming/tls의 Microsoft 문서를 참조하세요.

AWS Tools for PowerShell

AWS Tools for PowerShell는 AWS 서비스에 AWS SDK for .NET 대한 모든 호출에를 사용합니다. 다음과 같이 사용자 환경의 동작은 실행 중인 Windows PowerShell 버전에 따라 달라집니다.

Windows PowerShell 2.0 ~ 5.x

Windows PowerShell 2.0 ~ 5.x는 .NET Framework에서 실행됩니다. 다음 명령을 사용하여 PowerShell에서 사용되는 .NET 런타임(2.0 또는 4.0)을 확인할 수 있습니다.

```
$PSVersionTable.CLRVersion
```

- .NET 런타임 2.0을 사용하는 경우 AWS SDK for .NET 및 .NET Framework 3.5와 관련하여 앞서 제공한 지침을 따르십시오.

  > **Warning**
  >
  > 2024년 8월 15일에는 .NET Framework 3.5에 대한 지원을 AWS SDK for .NET 종료하고 최소 .NET Framework 버전을 4.7.2로 변경했습니다. 자세한 내용은 블로그 게시물 AWS SDK for .NET의 .NET Framework 3.5 및 4.5 대상에 대한 중요 변경 사항을 참조하세요.

- .NET 런타임 4.0을 사용하는 경우 AWS SDK for .NET 및 .NET Framework 4+와 관련하여 앞서 제공한 지침을 따르십시오.

Windows PowerShell 6.0

Windows PowerShell 6.0 이상은 .NET Core에서 실행됩니다. 다음 명령을 실행하여 사용되는 .NET Core 버전을 확인할 수 있습니다.

```
[System.Reflection.Assembly]::GetEntryAssembly().GetCustomAttributes([System.Runtime.Versioning
 $true).FrameworkName
```

AWS SDK for .NET 및 관련 .NET Core 버전과 관련하여 앞서 제공된 지침을 따릅니다.

Xamarin

Xamarin의 경우 https://learn.microsoft.com/en-us/xamarin/cross-platform/app-fundamentals/transport-layer-security의 지침을 참조하세요. 요약하면 다음과 같습니다.

Android

- Android 5.0 이상이 필요합니다.
- Project Properties(프로젝트 속성), Android Options(Android 옵션): HttpClient 구현을 Android로 설정하고 SSL/TLS 구현을 Native TLS 1.2+(네이티브 TLS 1.2 이상)로 설정해야 합니다.

iOS

- iOS 7 이상이 필요합니다.
- Project Properties(프로젝트 속성), iOS Build(iOS 빌드): HttpClient 구현을 NSUrlSession으로 설정해야 합니다.

macOS의 경우

- macOS 10.9 이상이 필요합니다.
- Project Options(프로젝트 옵션), Build(빌드), Mac Build(Mac 빌드): HttpClient 구현을 NSUrlSession으로 설정해야 합니다.

Unity

Unity 2018.2 이상을 사용하며 .NET 4.x와 동등한 스크립팅 런타임을 사용해야 합니다. https://docs.unity3d.com/2019.1/Documentation/Manual/ScriptingRuntimeUpgrade.html에 설명된 대로 프로젝트 설정, 구성, 플레이어에서 이를 설정할 수 있습니다. .NET 4.x와 동등한 스크립팅 런타임을 사용하면 Mono 또는 IL2CPP를 실행하는 모든 Unity 플랫폼에서 TLS 1.2를 지원할 수 있습니다.

브라우저(Blazor WebAssembly용)

WebAssembly는 서버가 아닌 브라우저에서 실행되며 브라우저를 사용하여 HTTP 트래픽을 처리합니다. 따라서 TLS 지원이 브라우저 지원에 의해 결정됩니다.

ASP.NET Core 3.1에 대한 미리 보기에서 Blazor WebAssembly는 https://learn.microsoft.com/en-us/aspnet/core/blazor/supported-platforms 설명된 대로 WebAssembly를 지원하는 브라우저에서만 지원됩니다. 모든 주요 브라우저는 WebAssembly를 지원하기 전에 TLS 1.2를 지원했습니다. 사용 중인 브라우저가 이에 해당하는 경우 앱이 실행되면 TLS 1.2를 통해 통신할 수 있습니다.

자세한 내용과 확인이 필요하면 해당 브라우저 설명서를 참조하십시오.

S3 암호화 클라이언트 마이그레이션(V1에서 V2로)

> **Note**
>
> V2를 사용 중이고 V4로 마이그레이션하려면 섹션을 참조하세요S3 암호화 클라이언트 마이그레이션(V2에서 V4로).

이 주제에서는 Amazon Simple Storage Service(S3) 암호화 클라이언트 버전 1(V1)에서 버전 2(V2)로 애플리케이션을 마이그레이션하고 마이그레이션 프로세스 전반에 걸쳐 애플리케이션 가용성을 보장하는 방법을 보여줍니다.

V2 클라이언트로 암호화된 객체는 V1 클라이언트로 해독할 수 없습니다. 모든 객체를 한 번에 다시 암호화할 필요 없이 새 클라이언트로 쉽게 마이그레이션할 수 있도록 "V1-transitional" 클라이언트가 제공되었습니다. 이 클라이언트는 V1과 V2로 암호화된 객체를 모두 해독할 수 있지만 V1 호환 형식으로만 객체를 암호화합니다. V2 클라이언트는 V1과 V2로 암호화된 객체를 모두 해독할 수 있지만(V1 객체에 대해 활성화된 경우) V2 호환 형식으로만 객체를 암호화합니다.

마이그레이션 개요

이 마이그레이션은 다음 세 단계로 진행됩니다. 이 단계는 여기에 소개되고 다음에 자세히 설명되어 있습니다. 다음 단계가 시작되기 전에 공유 객체를 사용하는 모든 클라이언트에 대해 각 단계를 완료해야 합니다.

1. 새 형식을 읽으려면 기존 클라이언트를 V1 전환 클라이언트로 업데이트하십시오. 먼저 V1 클라이언트 대신 V1 전환 클라이언트에 종속되도록 애플리케이션을 업데이트합니다. V1 전환 클라이언트를 사용하면 새 V2 클라이언트가 작성한 객체와 V1 호환 형식으로 작성된 객체를 기존 코드에서 해독할 수 있습니다.

   > **Note**
   >
   > V1 전환 클라이언트는 마이그레이션 목적으로만 제공됩니다. V1 전환 클라이언트로 이동한 후 V2 클라이언트로 업그레이드를 진행합니다.

2. V1 전환 클라이언트를 V2 클라이언트로 마이그레이션하여 새 형식을 작성하십시오. 다음으로, 애플리케이션의 모든 V1 전환 클라이언트를 V2 클라이언트로 교체하고 보안 프로필을 `V2AndLegacy`로 설정합니다. V2 클라이언트에서 이 보안 프로필을 설정하면 해당 클라이언트가 V1 호환 형식으로 암호화된 객체를 해독할 수 있습니다.

3. V1 형식을 더 이상 읽지 않도록 V2 클라이언트를 업데이트하십시오. 마지막으로 모든 클라이언트를 V2로 마이그레이션하고 모든 객체를 V2 호환 형식으로 암호화하거나 다시 암호화한 후에는 V2 보안 프로필을 `V2AndLegacy` 대신 `V2`로 설정합니다. 이렇게 하면 V1 호환 형식의 객체를 해독할 수 없습니다.

새 형식을 읽으려면 기존 클라이언트를 V1 전환 클라이언트로 업데이트

V2 암호화 클라이언트는 이전 버전의 클라이언트에서 지원하지 않는 암호화 알고리즘을 사용합니다. 마이그레이션의 첫 번째 단계는 새로운 형식을 읽을 수 있도록 V1 암호 해독 클라이언트를 업데이트하는 것입니다.

V1 전환 클라이언트를 사용하면 애플리케이션이 V1 및 V2로 암호화된 객체를 모두 해독할 수 있습니다. 이 클라이언트는 Amazon.Extensions.S3.Encryption NuGet 패키지의 일부입니다. V1 전환 클라이언트를 사용하려면 각 애플리케이션에서 다음 단계를 수행하세요.

1. Amazon.Extensions.S3.Encryption 패키지에 대한 새로운 종속성을 구축합니다. 프로젝트가 AWSSDK.S3 또는 AWSSDK.KeyManagementService 패키지에 직접 의존하는 경우 이러한 종속성을 업데이트하거나 제거하여 업데이트된 버전을 이 새 패키지와 함께 가져오도록 해야 합니다.
2. 다음과 같이 적절한 `using` 문을 `Amazon.S3.Encryption`에서 `Amazon.Extensions.S3.Encryption`으로 변경합니다.

```
// using Amazon.S3.Encryption;
  using Amazon.Extensions.S3.Encryption;
```

3. 애플리케이션을 다시 빌드하고 다시 배포합니다.

V1 전환 클라이언트는 V1 클라이언트와 완전히 API 호환되므로 다른 코드 변경이 필요하지 않습니다.

V1 전환 클라이언트를 V2 클라이언트로 마이그레이션하여 새 형식 작성

V2 클라이언트는 Amazon.Extensions.S3.Encryption NuGet 패키지의 일부입니다. 이를 통해 애플리케이션은 V1 및 V2로 암호화된 객체(그렇게 구성된 경우)를 모두 해독할 수 있지만 객체는 V2 호환 형식으로만 암호화합니다.

새 암호화 형식을 읽도록 기존 클라이언트를 업데이트한 후 애플리케이션을 V2 암호화 및 복호화 클라이언트로 안전하게 업데이트를 진행할 수 있습니다. V2 클라이언트를 사용하려면 각 애플리케이션에서 다음 단계를 수행하세요.

1. `EncryptionMaterials`를 `EncryptionMaterialsV2`로 변경합니다.

a. KMS를 사용하는 경우:

   i. KMS 키 ID를 제공합니다.

   ii. 사용 중인 암호화 방법, 즉, `KmsType.KmsContext`를 선언합니다.

   iii. 이 데이터 키와 연결할 암호화 컨텍스트를 KMS에 제공합니다. 빈 딕셔너리를 보낼 수 있지만(Amazon 암호화 컨텍스트는 계속 병합됨) 추가 컨텍스트를 제공하는 것이 좋습니다.

b. 사용자 제공 키 랩 방법(대칭 또는 비대칭 암호화)을 사용하는 경우:

   i. 암호화 자료가 포함된 AES 또는 RSA 인스턴스를 제공합니다.

   ii. 사용할 암호화 알고리즘, 즉, `SymmetricAlgorithmType.AesGcm` 또는 `AsymmetricAlgorithmType.RsaOaepSha1`를 선언합니다.

2. `SecurityProfile` 속성을 `SecurityProfile.V2AndLegacy`로 설정하여 `AmazonS3CryptoConfiguration`을 `AmazonS3CryptoConfigurationV2`로 변경합니다.

3. `AmazonS3EncryptionClient`를 `AmazonS3EncryptionClientV2`로 변경합니다. 이 클라이언트는 이전 단계에서 새로 변환된 `AmazonS3CryptoConfigurationV2` 및 `EncryptionMaterialsV2` 객체를 가져옵니다.

예: KMS에서 KMS+컨텍스트로

사전 마이그레이션

```
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);
```

마이그레이션 후

```
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);
```

예: 대칭 알고리즘(AES-CBC에서 AES-GCM 키 랩으로)

`StorageMode`는 `ObjectMetadata` 또는 `InstructionFile`일 수 있습니다.

사전 마이그레이션

```
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);
```

마이그레이션 후

```
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**
>
> AES-GCM으로 해독할 때는 해독된 데이터를 사용하기 전에 전체 객체를 끝까지 읽습니다. 이는 암호화되었던 객체이므로 객체가 수정되지 않았는지 확인하기 위함입니다.

예: 비대칭 알고리즘(RSA에서 RSA-OAEP-SHA1 키 랩으로)

`StorageMode`는 `ObjectMetadata` 또는 `InstructionFile`일 수 있습니다.

사전 마이그레이션

```
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);
```

마이그레이션 후

```
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);
```

V1 형식을 더 이상 읽지 않도록 V2 클라이언트 업데이트

결국에는 모든 객체가 V2 클라이언트를 사용하여 암호화되거나 다시 암호화됩니다. 변환이 완료되면 다음 코드 조각에 표시된 것처럼 `SecurityProfile` 속성을 `SecurityProfile.V2`로 설정하여 V2 클라이언트에서 V1 호환성을 비활성화할 수 있습니다.

```
//var configuration = new AmazonS3CryptoConfigurationV2(SecurityProfile.V2AndLegacy);
var configuration = new AmazonS3CryptoConfigurationV2(SecurityProfile.V2);
```

S3 암호화 클라이언트 마이그레이션(V2에서 V4로)

> **Note**
>
> V1을 사용하고 V4로 마이그레이션하려면 먼저 V2로 마이그레이션해야 합니다. S3 암호화 클라이언트 마이그레이션(V1에서 V2로)을(를) 참조하세요.

이 주제에서는 Amazon Simple Storage Service(S3) 암호화 클라이언트 버전 2(V2)에서 버전 4(V4)로 애플리케이션을 마이그레이션하고 마이그레이션 프로세스 전반에 걸쳐 애플리케이션 가용성을 보장하는 방법을 보여줍니다. V4는 콘텐츠 암호화를 위한 키 커밋과 함께 AES-GCM을 사용하고 키 대체 공격에 대한 보안을 강화하기 위한 커밋 정책을 도입합니다.

V4 클라이언트는 Amazon.Extensions.S3.Encryption NuGet 패키지에서 사용할 수 있습니다.

> **Important**
>
> 주요 변경 사항: `RequireEncryptRequireDecrypt` 정책으로 구성된 V4 클라이언트는 V1 또는 V2 클라이언트로 암호화된 객체를 해독할 수 없습니다. 최신 V2 클라이언트만 키 커밋으로 V4-encrypted 복호화할 수 있습니다. `RequireEncryptRequireDecrypt` 정책으로 이동하기 전에 키 커밋이 활성화된 V4 클라이언트를 사용하여 기존 데이터를 모두 다시 암호화해야 합니다.

V4 개념 이해

V4는 콘텐츠 암호화를 위한 키 커밋과 함께 AES-GCM을 사용하며 암호화된 데이터의 보호를 강화하는 주요 보안 개념을 도입합니다.

약정 정책

커밋 정책은 암호화 및 복호화 작업 중에 암호화 클라이언트가 키 커밋을 처리하는 방법을 제어합니다. V4는 세 가지 커밋 정책을 지원합니다.

`ForbidEncryptAllowDecrypt`

암호화: 커밋 없음

복호화: 커밋되지 않는 객체 허용

보안: 약정을 적용하지 않으며 변조를 허용할 수 있음

호환성: 모든 V2 및 V4 구현은이 정책으로 암호화된 객체를 읽을 수 있습니다.

`RequireEncryptAllowDecrypt`

암호화: 키 커밋 사용

복호화: 커밋하는 객체와 커밋하지 않는 객체 모두 허용

보안: 새 객체가 키 대체 공격으로부터 보호되고 이전 객체는 여전히 읽을 수 있음

호환성: V4만이 정책을 지원합니다.

`RequireEncryptRequireDecrypt`(V4의 기본값)

암호화: 키 커밋 사용

복호화: 객체만 커밋

보안: 최대 보안을 위한 전체 약정 적용

호환성: V4만이 정책을 지원합니다.

키 커밋이 있는 AES GCM

V4는 향상된 보안을 제공하는 콘텐츠 암호화에 대한 키 커밋과 함께 AES-GCM을 사용합니다.

- 변조 방지: 암호화된 데이터에 키를 암호화 방식으로 바인딩하여 키 대체 공격으로부터 보호합니다.
- 버전 호환성: 키 커밋으로 암호화된 객체는 V4 클라이언트 및 최신 버전에서만 해독할 수 있습니다.

> **Warning**
>
> 프로덕션 환경에서 키 커밋 암호화를 활성화하기 전에 V2 클라이언트가 더 이상 사용되지 않으므로 객체를 복호화해야 하는 모든 애플리케이션이 V4 이상으로 업그레이드되었는지 확인합니다.

V4 형식을 읽도록 기존 클라이언트 업데이트

V4 암호화 클라이언트는 이전 버전의 클라이언트가 지원하지 않는 암호화 알고리즘을 사용합니다. 마이그레이션의 첫 번째 단계는 새 V4 형식을 읽을 수 있도록 V2 클라이언트를 업데이트하는 것입니다. V4

NuGet 패키지 종속성 업데이트

V4 지원이 포함된 최신 버전의 Amazon.Extensions.S3.Encryption NuGet 패키지를 사용하도록 애플리케이션을 업데이트합니다. 각 애플리케이션에서 다음 단계를 수행합니다.

1. 최신 Amazon.Extensions.S3.Encryption 패키지로 업데이트합니다. 프로젝트가 AWSSDK.S3 또는 AWSSDK.KeyManagementService 패키지에 직접 의존하는 경우 이러한 종속성을 업데이트하거나 제거하여 업데이트된 버전을 이 새 패키지와 함께 가져오도록 해야 합니다.
2. `using` 문이 올바른 네임스페이스를 참조하는지 확인합니다.

   ```
   using Amazon.Extensions.S3.Encryption;
   using Amazon.Extensions.S3.Encryption.Primitives;
   ```

3. 애플리케이션을 다시 빌드하고 다시 배포합니다.

기존 V2 클라이언트는 업데이트된 패키지로 계속 작동하며 V4 클라이언트로 암호화된 객체를 복호화할 수 있습니다(사용된 커밋 정책에 따라 다름).

애플리케이션 빌드 및 배포

NuGet 패키지 종속성을 업데이트한 후:

1. 애플리케이션을 빌드하여 모든 종속성이 올바르게 해결되도록 합니다.
2. 개발 환경에서 애플리케이션을 테스트하여 기존 기능이 계속 작동하는지 확인합니다.
3. 업데이트된 애플리케이션을 프로덕션 환경에 배포합니다.

이 업데이트를 통해 기존 V2 클라이언트는 V4 클라이언트가 암호화할 객체를 복호화하여 마이그레이션 프로세스 중에 호환성을 보장할 수 있습니다.

V4 클라이언트로 마이그레이션

기존 클라이언트를 업데이트하여 새 암호화 형식을 읽은 후 V4 암호화 및 복호화 클라이언트를 사용하도록 애플리케이션을 안전하게 업데이트할 수 있습니다. V4 클라이언트는 기존의 암호화된 객체와의 호환성을 유지하면서 키 커밋을 통해 향상된 보안을 제공합니다.

4-Step 마이그레이션 프로세스

V2에서 V4로의 마이그레이션은 호환성과 보안을 보장하기 위해 구조화된 4단계 프로세스를 따릅니다. 각 단계는 다음 단계로 진행하기 전에 모든 애플리케이션에 배포해야 하는 특정 구성을 나타냅니다.

1. 0단계: V2 클라이언트(시작점) - 기존 V2 구현
2. 1단계: V4 있는 V4 V2 V2-compatible 암호화 동작을 유지하면서 V4 클라이언트로 마이그레이션
3. 2단계: 키 커밋 쓰기가 포함된 V4 - 레거시 객체의 복호화를 허용하면서 키 커밋으로 암호화 시작
4. 3단계: 전체 적용이 포함된 V4 - 암호화 및 복호화 모두에 키 커밋 필요

0단계: V2 클라이언트(시작점)

이는 기존 V2 클라이언트 구성을 나타냅니다. 이 단계에서는 마이그레이션 전 시작 상태를 보여줍니다.

```
using Amazon.Extensions.S3.Encryption;
using Amazon.Extensions.S3.Encryption.Primitives;
using Amazon.S3.Model;

// Step 0: V2 Client - Starting configuration
var encryptionContext = new Dictionary<string, string>();
var encryptionMaterial = new EncryptionMaterialsV2(kmsKeyId, KmsType.KmsContext,
 encryptionContext);

#pragma warning disable 0618
var configuration = new AmazonS3CryptoConfigurationV2(SecurityProfile.V2);
#pragma warning enable 0618

var encryptionClient = new AmazonS3EncryptionClientV2(configuration,
 encryptionMaterial);
```

```
// Use the client for PutObject and GetObject operations
await encryptionClient.PutObjectAsync(new PutObjectRequest
{
    BucketName = bucket,
    Key = objectKey,
    ContentBody = content
});
```

1단계: V2 호환성이 있는 V4 V2

V2와 동일한 동작을 유지하면서 V4 클라이언트로 마이그레이션합니다. V2 이 단계에서는 `ForbidEncryptAllowDecrypt` 정책을 사용하여 커밋 없이 암호화하고 모든 객체의 복호화를 허용합니다.

```
using Amazon.Extensions.S3.Encryption;
using Amazon.Extensions.S3.Encryption.Primitives;
using Amazon.S3.Model;

// Step 1: V4 Client with V2 compatibility
var encryptionContext = new Dictionary<string, string>();
var encryptionMaterial = new EncryptionMaterialsV4(kmsKeyId, KmsType.KmsContext,
 encryptionContext);

var configuration = new AmazonS3CryptoConfigurationV4(
    SecurityProfile.V4,
    CommitmentPolicy.ForbidEncryptAllowDecrypt,
    ContentEncryptionAlgorithm.AesGcm);

var encryptionClient = new AmazonS3EncryptionClientV4(configuration,
 encryptionMaterial);

// Use the client for PutObject and GetObject operations
await encryptionClient.PutObjectAsync(new PutObjectRequest
{
    BucketName = bucket,
    Key = objectKey,
    ContentBody = content
});
```

동작: 커밋 없는 암호화는 커밋 객체와 커밋되지 않은 객체를 모두 복호화할 수 있습니다. V2 동작과 동일합니다.

2단계: 키 커밋 쓰기가 포함된 V4

복호화를 위한 이전 버전과의 호환성을 유지하면서 키 커밋으로 암호화를 시작합니다. 이 단계에서는 `RequireEncryptAllowDecrypt` 정책을 사용합니다.

> **Warning**
>
> 2단계를 배포하기 전에 키 커밋 암호화를 처리하기 위해 모든 리더가 1단계 이상으로 업데이트되었는지 확인합니다.

```
using Amazon.Extensions.S3.Encryption;
using Amazon.Extensions.S3.Encryption.Primitives;
using Amazon.S3.Model;

// Step 2: V4 Client with key commitment writes
var encryptionContext = new Dictionary<string, string>();
var encryptionMaterial = new EncryptionMaterialsV4(kmsKeyId, KmsType.KmsContext,
 encryptionContext);

var configuration = new AmazonS3CryptoConfigurationV4(
    SecurityProfile.V4,
    CommitmentPolicy.RequireEncryptAllowDecrypt,
    ContentEncryptionAlgorithm.AesGcmWithCommitment);

var encryptionClient = new AmazonS3EncryptionClientV4(configuration,
 encryptionMaterial);

// Use the client for PutObject and GetObject operations
await encryptionClient.PutObjectAsync(new PutObjectRequest
{
    BucketName = bucket,
    Key = objectKey,
    ContentBody = content
});
```

동작: 커밋으로 암호화하면 커밋 객체와 커밋되지 않은 객체를 모두 복호화할 수 있습니다. 새 객체는 키 대체 공격으로부터 보호됩니다.

3단계: 전체 적용이 적용된 V4

암호화와 복호화 모두에 키 커밋이 필요합니다. 이 단계에서는 보안을 극대화하기 위해 `RequireEncryptRequireDecrypt` 정책을 사용합니다.

> **Warning**
>
> 3단계를 배포하기 전에 시스템의 모든 객체가 키 커밋으로 다시 암호화되었는지 확인합니다(2단계). 이 단계는 커밋 없이 암호화된 객체를 복호화하지 못합니다.

```
using Amazon.Extensions.S3.Encryption;
using Amazon.Extensions.S3.Encryption.Primitives;
using Amazon.S3.Model;

// Step 3: V4 Client with full key commitment enforcement
var encryptionContext = new Dictionary<string, string>();
var encryptionMaterial = new EncryptionMaterialsV4(kmsKeyId, KmsType.KmsContext,
 encryptionContext);

var configuration = new AmazonS3CryptoConfigurationV4(
    SecurityProfile.V4,
    CommitmentPolicy.RequireEncryptRequireDecrypt,
    ContentEncryptionAlgorithm.AesGcmWithCommitment);

var encryptionClient = new AmazonS3EncryptionClientV4(configuration,
 encryptionMaterial);

// Use the client for PutObject and GetObject operations
await encryptionClient.PutObjectAsync(new PutObjectRequest
{
    BucketName = bucket,
    Key = objectKey,
    ContentBody = content
});
```

동작: 커밋으로 암호화하고 커밋으로 암호화된 객체만 복호화합니다. 키 대체 공격에 대한 최대 보안.

추가 구성 예제

이 섹션에서는 마이그레이션 중에 다양한 옵션으로 V4 클라이언트를 구성하는 추가 예제를 제공합니다.

레거시 지원 활성화

V4 클라이언트가 V1 및 V2 클라이언트로 암호화된 객체를 읽을 수 있도록 하려면 레거시 복호화를 허용하는 커밋 정책으로 클라이언트를 구성합니다.

```
using Amazon.Extensions.S3.Encryption;
using Amazon.Extensions.S3.Encryption.Primitives;

// Configure V4 client to read V1/V2 objects
var configuration = new AmazonS3CryptoConfigurationV4(SecurityProfile.V4AndLegacy,
 CommitmentPolicy.RequireEncryptAllowDecrypt)
{
    StorageMode = CryptoStorageMode.ObjectMetadata
};

// This configuration allows:
// - Encryption: With commitment (secure)
// - Decryption: Both V2 (non-committing) and V4 (committing) objects
```

이전 클라이언트가 암호화한 객체를 복호화해야 하는 경우 마이그레이션 중에이 구성을 사용하고 새 객체를 향상된 보안으로 암호화합니다.

스토리지 방법 구성

V4는 암호화 메타데이터에 대해 두 가지 스토리지 방법을 지원합니다. 사용 사례에 가장 적합한 방법을 선택합니다.

객체 메타데이터(기본값)

```
var configuration = new AmazonS3CryptoConfigurationV4(SecurityProfile.V4,
 CommitmentPolicy.RequireEncryptAllowDecrypt)
{
    StorageMode = CryptoStorageMode.ObjectMetadata
};
// Encryption metadata is stored in S3 object metadata
```

지침 파일

```
var configuration = new AmazonS3CryptoConfigurationV4(SecurityProfile.V4,
 CommitmentPolicy.RequireEncryptAllowDecrypt)
{
```

```
    StorageMode = CryptoStorageMode.InstructionFile
};
// Encryption metadata is stored in a separate S3 object (instruction file)
```

다른 목적으로 객체 메타데이터를 보존InstructionFile해야 하거나 메타데이터 크기 제한이 있는 객체로 작업할 때 사용합니다.

커밋 정책 구성

보안 요구 사항 및 마이그레이션 단계에 따라 적절한 커밋 정책을 선택합니다.

마이그레이션 단계(V2 호환성)

```
// For migration: encrypt without commitment, allow all decryption
var migrationConfig = new AmazonS3CryptoConfigurationV4(SecurityProfile.V4,
 CommitmentPolicy.ForbidEncryptAllowDecrypt);
```

전환 단계(권장)

```
// For transition: encrypt with commitment, allow legacy decryption
var transitionConfig = new AmazonS3CryptoConfigurationV4(SecurityProfile.V4,
 CommitmentPolicy.RequireEncryptAllowDecrypt);
```

전체 보안 단계

```
// For maximum security: require commitment for both encryption and decryption
var secureConfig = new AmazonS3CryptoConfigurationV4(SecurityProfile.V4,
 CommitmentPolicy.RequireEncryptRequireDecrypt);
```

초기 마이그레이션 ForbidEncryptAllowDecrypt 중에 로 시작하고 전환 단계를 RequireEncryptAllowDecrypt 위해 로 이동한 다음, 마지막으로 모든 클라이언트가 업그레이드되고 모든 객체가 커밋으로 다시 암호화되었을 RequireEncryptRequireDecrypt 때를 사용합니다.

에 대한 특별 고려 사항 AWS SDK for .NET

이 섹션에는 일반적인 구성 또는 절차가 적절하지 않거나 충분하지 않은 특수한 경우에 대한 고려 사항이 포함되어 있습니다.

주제

- 용 어셈블리 가져오기 AWS SDK for .NET
- 애플리케이션에서 자격 증명 및 프로필에 액세스
- Unity 지원에 대한 특별 고려 사항
- Xamarin 지원에 대한 특별 고려 사항

용 어셈블리 가져오기 AWS SDK for .NET

이 주제에서는 프로젝트에서 사용하기 위해 AWSSDK 어셈블리를 구하여 로컬(또는 온프레미스)에 저장하는 방법을 설명합니다. 이는 SDK 참조를 처리하는 데 권장되는 방법은 아니지만 일부 환경에서는 필요합니다.

> **Note**
>
> SDK 참조를 처리하는 권장되는 방법은 각 프로젝트에 필요한 NuGet 패키지만 다운로드하여 설치하는 것입니다. 이 방법은 NuGet을 사용하여 AWSSDK 패키지를 설치에 설명되어 있습니다.

프로젝트별로 NuGet 패키지를 다운로드 및 설치할 수 없거나 설치할 수 없는 경우 다음 옵션을 사용할 수 있습니다.

ZIP 파일 다운로드 및 압축

(이 방법은에 대한 참조를 처리하는 데 권장되는 방법이 아닙니다 AWS SDK for .NET.)

1. 다음 ZIP 파일 중 하나를 다운로드합니다.

   - aws-sdk-net8.0.zip - .NET 8 이상을 지원하는 어셈블리.
   - aws-sdk-netcoreapp3.1.zip - .NET Core 3.1 이상을 지원하는 어셈블리.
   - aws-sdk-netstandard2.0.zip - .NET Standard 2.0 및 2.1을 지원하는 어셈블리.

- aws-sdk-net472.zip - .NET Framework 4.5 이상을 지원하는 어셈블리입니다.

2. 어셈블리를 파일 시스템의 “다운로드” 폴더에 추출하세요. 어디에 있든 상관 없습니다. 이 폴더를 기록해 둡니다.
3. 프로젝트를 설정하면 NuGet 없이 AWSSDK 어셈블리 설치에 설명된 대로 이 폴더에서 필요한 어셈블리를 가져올 수 있습니다.

애플리케이션에서 자격 증명 및 프로필에 액세스

자격 증명을 사용하는 데 선호되는 방법은에서에 설명된 대로 자동으로 자격 증명을 AWS SDK for .NET 찾고 검색할 수 있도록 허용하는 것입니다보안 인증 정보 및 프로파일 확인.

그러나 프로필과 자격 증명을 적극적으로 검색한 다음 AWS 서비스 클라이언트를 생성할 때 해당 자격 증명을 명시적으로 사용하도록 애플리케이션을 구성할 수도 있습니다.

프로필과 자격 증명을 미리 검색하려면 Amazon.Runtime.CredentialManagement 네임스페이스의 클래스를 사용하세요.

- 자격 AWS 증명 파일 형식(기본 위치의 공유 AWS 자격 증명 파일 또는 사용자 지정 자격 증명 파일)을 사용하는 파일에서 프로파일을 찾으려면 SharedCredentialsFile 클래스를 사용합니다. 이 텍스트에서는 간결하게 설명하기 위해 이 형식의 파일을 간단히 보안 인증 파일이라고 부르기도 합니다.
- SDK 스토어에서 프로필을 찾으려면 NetSDKCredentialsFile 클래스를 사용하세요.
- 클래스 속성의 구성에 따라 보안 인증 파일과 SDK 스토어에서 모두 검색하려면 CredentialProfileStoreChain 클래스를 사용하세요.

  이 클래스를 사용하여 프로필을 찾을 수 있습니다. 클래스를 사용하는 대신이 `AWSCredentialsFactory` 클래스를 사용하여 자격 AWS 증명을 직접 요청할 수도 있습니다(다음 설명 참조).
- 프로필에서 다양한 유형의 보안 인증을 검색하거나 생성하려면 AWSCredentialsFactory 클래스를 사용하세요.

다음 섹션에는 이러한 클래스의 예가 나와 있습니다.

클래스 CredentialProfileStoreChain의 예

TryGetAWSCredentials 또는 TryGetProfile 메서드를 사용하여 CredentialProfileStoreChain 클래스에서 보안 인증 및 프로필을 가져올 수 있습니다. 다음과 같이 클래스의 `ProfilesLocation` 속성은 메서드의 동작을 결정합니다.

- `ProfilesLocation`가 null이거나 비어 있는 경우 플랫폼이 지원하는 경우 SDK 스토어를 검색한 다음 기본 위치에서 공유 AWS 자격 증명 파일을 검색합니다.
- `ProfilesLocation` 속성에 값이 포함된 경우 속성에 지정된 보안 인증 파일을 검색하세요.

SDK 스토어 또는 공유 자격 증명 파일에서 AWS 자격 증명 가져오기

이 예제는 `CredentialProfileStoreChain` 클래스를 사용하여 보안 인증을 가져온 다음 보안 인증을 사용하여 AmazonS3Client 객체를 생성하는 방법을 보여줍니다. 보안 인증 정보는 SDK 스토어에서 가져오거나 기본 위치의 공유 AWS 보안 인증 파일에서 가져올 수 있습니다.

이 예제에서는 Amazon.Runtime.AWSCredentials 클래스도 사용합니다.

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

SDK 스토어 또는 공유 AWS 자격 증명 파일에서 프로필 가져오기

이 예제에서는 CredentialProfileStoreChain 클래스를 사용하여 프로필을 가져오는 방법을 보여 줍니다. 자격 증명은 SDK 스토어 또는 기본 위치의 공유 AWS 자격 증명 파일에서 가져올 수 있습니다.

이 예제에서는 CredentialProfile 클래스도 사용합니다.

```
var chain = new CredentialProfileStoreChain();
CredentialProfile basicProfile;
if (chain.TryGetProfile("basic_profile", out basicProfile))
{
    // Use basicProfile
}
```

사용자 지정 보안 인증 파일에서 보안 인증 정보 가져오기

이 예제에서는 CredentialProfileStoreChain 클래스를 사용하여 보안 인증을 가져오는 방법을 보여 줍니다. 자격 증명은 AWS 자격 증명 파일 형식을 사용하지만 대체 위치에 있는 파일에서 가져옵니다.

이 예제에서는 Amazon.Runtime.AWSCredentials 클래스도 사용합니다.

```
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
}
```

클래스 SharedCredentialsFile 및 AWSCredentialsFactory의 예

SharedCredentialsFile 클래스를 사용하여 AmazonS3Client 생성

이 예제에서는 공유 AWS 자격 증명 파일에서 프로필을 찾고, 프로필에서 자격 증명을 생성한 AWS 다음, 자격 증명을 사용하여 AmazonS3Client 객체를 생성하는 방법을 보여줍니다. 이 예제에서는 SharedCredentialsFile 클래스를 사용합니다.

이 예제에서는 CredentialProfile 클래스 및 Amazon.Runtime.AWSCredentials 클래스도 사용합니다.

```
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 클래스를 동일한 방식으로 사용할 수 있습니다. 단, SharedCredentialsFile 객체 대신 새 NetSDKCredentialsFile 객체를 인스턴스화한다는 점이 다릅니다.

Unity 지원에 대한 특별 고려 사항

Unity 애플리케이션에 AWS SDK for .NET 및 .NET Standard 2.0을 사용하는 경우 애플리케이션은 NuGet을 사용하는 대신 어셈블리(DLL 파일)를 AWS SDK for .NET 직접 참조해야 합니다. 이 요구 사항을 고려할 때 수행해야 하는 중요한 작업은 다음과 같습니다.

- AWS SDK for .NET 어셈블리를 가져와 프로젝트에 적용해야 합니다. 이를 위한 자세한 방법은 AWSSDK 어셈블리 가져오기 주제의 ZIP 파일 다운로드 및 압축을 참조하세요.

- AWSSDK.Core용 DLLs 및 사용 중인 기타 AWS 서비스와 함께 Unity 프로젝트에 다음 DLLs을 포함해야 합니다. 의 버전 3.5.109부터 AWS SDK for .NET.NET Standard ZIP 파일에는 이러한 추가 DLLs.
  - Microsoft.Bcl.AsyncInterfaces.dll
  - System.Runtime.CompilerServices.Unsafe.dll
  - System.Threading.Tasks.Extensions.dll

- IL2CPP를 사용하여 Unity 프로젝트를 빌드하는 경우 코드 스트리핑을 방지하기 위해 자산 폴더에 `link.xml` 파일을 추가해야 합니다. `link.xml` 파일에는 사용 중인 모든 AWSSDK 어셈블리가 나열되어야 하며 각 어셈블리에는 `preserve="all"` 속성이 포함되어야 합니다. 이 파일에 대한 예는 다음 코드 조각에 표시됩니다.

```
<linker>
    <assembly fullname="AWSSDK.Core" preserve="all"/>
    <assembly fullname="AWSSDK.DynamoDBv2" preserve="all"/>
    <assembly fullname="AWSSDK.Lambda" preserve="all"/>
</linker>
```

> **Note**
>
> 이 요구 사항과 관련된 흥미로운 배경 정보를 읽으려면 https://aws.amazon.com/blogs/developer/referencing-the-aws-sdk-for-net-standard-2-0-from-unity-xamarin-or-uwp/ 문서를 참조하세요.

Xamarin 지원에 대한 특별 고려 사항

Xamarin 프로젝트(신규 및 기존)는 .NET Standard 2.0을 대상으로 해야 합니다. Xamarin.Forms의 .NET Standard 2.0 지원 및 .NET 구현 지원을 참조하세요.

또한 Portable Class Library 및 Xamarin에 대한 정보도 참조하세요.

에 대한 API 참조 AWS SDK for .NET

는 AWS 서비스에 액세스하는 데 사용할 수 있는 API를 AWS SDK for .NET 제공합니다. API에서 사용할 수 있는 클래스와 메서드를 확인하려면 AWS SDK for .NET API 참조를 참조하세요.

위에 제공된 일반 참조 외에도 가이드 코드 예제 섹션 아래의 각 예제에는 해당 예제에서 사용되는 특정 메서드 및 클래스에 대한 참조가 포함되어 있습니다.

API 참조 버전 정보

앞서 설명한 API 참조는의 버전 3.0 이상에 대한 것입니다 AWS SDK for .NET.

SDK의 이전 버전에서 마이그레이션하는 방법에 대한 자세한 내용은 섹션을 참조하세요. 프로젝트 마이그레이션

SDK API 참조의 이전 버전에서 더 이상 사용되지 않는 콘텐츠를 찾으려면 다음 항목(들)을 참조하세요.

- AWS SDK for .NET API 참조 V1(사용되지 않음)
- AWS SDK for .NET API 참조 V2(사용되지 않음)

더 이상 사용되지 않는 AWS SDK for .NET API 참조를 보려면 API 참조를 추출하고 웹 브라우저를 구성해야 합니다. 다음에 표시된 지침은 이를 수행하는 방법의 예입니다. Windows 시스템에서 Google Chrome 웹 브라우저를 사용하는 것을 기반으로 합니다. 특정 웹 브라우저 및 운영 체제에 맞게 조정합니다.

더 이상 사용되지 않는 API 참조 V1 보기

1. 더 이상 사용되지 않는 AWS SDK for .NET API 참조 V1에 대한 ZIP 파일을 다운로드합니다. 기본적으로 ZIP 파일의 이름은 입니다`sdkfornet-api-ref_v1_deprecated.zip`. 이 이름은이 지침 전체에서 사용됩니다.
2. 선택한 폴더에 ZIP 파일을 배치합니다. 이러한 지침에서 폴더 이름은 로 간주됩니다`C:\work\temp\api-refs\V1`.
3. ZIP 파일을 마우스 오른쪽 버튼으로 클릭하고 모두 추출을 선택합니다. 이 지침을 `C:\work\temp\api-refs\V1\sdkfornet-api-ref_v1` 위한 기본 위치를 수락합니다.

4. C:\work\temp\api-refs\V1 폴더에서 Google Chrome에 대한 바로 가기를 생성합니다. 원래 애플리케이션을 이동하지 않도록 주의하십시오.
5. 새 바로 가기의 속성에서 다음 필드를 설정합니다.

   - 대상: "*<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"

     --user-data-dir 인수의 경우 환경에 적합한 폴더 이름을 사용합니다. 폴더가 반드시 존재할 필요는 없습니다.

   - 시작 위치: C:\work\temp\api-refs\V1
6. 바로 가기에 적절한 이름을 지정합니다.
7. 바로 가기를 열어 이전 API 참조를 확인합니다.

더 이상 사용되지 않는 API 참조 V2 보기

1. 더 이상 사용되지 않는 AWS SDK for .NET API 참조 V2용 ZIP 파일을 다운로드합니다. 기본적으로 ZIP 파일의 이름은 입니다sdkfornet-api-ref_v2_deprecated.zip. 이 이름은이 지침 전체에서 사용됩니다.
2. 선택한 폴더에 ZIP 파일을 배치합니다. 이 지침에서 폴더 이름은 로 간주됩니다C:\work\temp\api-refs\V2.
3. ZIP 파일을 마우스 오른쪽 버튼으로 클릭하고 모두 추출을 선택합니다. 이 지침을 C:\work\temp\api-refs\V2\sdkfornet-api-ref_v2 위한 기본 위치를 수락합니다.
4. C:\work\temp\api-refs\V2 폴더에서 Google Chrome에 대한 바로 가기를 생성합니다. 원래 애플리케이션을 이동하지 않도록 주의하십시오.
5. 새 바로 가기의 속성에서 다음 필드를 설정합니다.

   - 대상: "*<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"

     --user-data-dir 인수의 경우 환경에 적합한 폴더 이름을 사용합니다. 폴더가 반드시 존재할 필요는 없습니다.

   - 시작 위치: C:\work\temp\api-refs\V2
6. 바로 가기에 적절한 이름을 지정합니다.

7. 바로 가기를 열어 이전 API 참조를 확인합니다.

문서 기록

다음 표에서는 AWS SDK for .NET 개발자 안내서의 최신 릴리스 이후 변경된 중요 사항에 대해 설명합니다. 이 설명서에 대한 업데이트 알림을 받으려면 RSS feed를 구독하세요.

변경 사항	설명	날짜
새로운 기능	.NET CLI용 AWS 배포 도구 버전 2.0에 대한 참고 사항이 추가되었습니다.	2025년 10월 17일
AWS OpsWorks	AWS OpsWorks 가 수명 종료에 도달했습니다. 설명서에서 제거되었습니다.	2025년 9월 16일
새로운 기능	의 V3에 대한 End-of-support 가 발표되었습니다 AWS SDK for .NET.	2025년 8월 21일
새로운 기능	분산 캐시 공급자의 일반 가용성	2025년 7월 2일
메시지 처리 프레임워크	.NET용 AWS 메시지 처리 프레임워크가 정식 출시되었습니다!	2025년 5월 5일
새로운 기능	버전 4 AWS SDK for .NET 를 정식 버전으로 사용할 수 있습니다. 애플리케이션을 V4로 마이그레이션하는 방법에 대한 자세한 내용은 버전 4로 마이그레이션을 참조하세요.	2025년 4월 28일
의 버전 4로 마이그레이션 AWS SDK for .NET	로 빌드된 애플리케이션을 버전 4로 마이그레이션 AWS SDK for .NET 하는 방법에 대한 미리 보기 정보가 포함되어 있습니다.	2025년 3월 3일

에서 .NET Aspire AWS 와 통합 AWS SDK for .NET	에서 AWS 가 .NET Aspire와 통합되는 방법에 대한 정보가 포함되어 있습니다 AWS SDK for .NET.	2025년 2월 15일
Observability	관찰성을 위한 GA 릴리스 발표	2025년 2월 10일
새로운 기능	무결성 보호를 위한 새로운 기본 동작에 대한 정보가 추가되었습니다.	2025년 1월 15일
S3 암호화 클라이언트 V4 마이그레이션	V2에서 V4로 마이그레이션하는 방법에 대한 정보가 추가되었습니다.	2024년 12월 9일
새로운 기능	AWS SDK for .NET 버전 4의 네 번째 미리 보기 릴리스에 대한 정보가 추가되었습니다.	2024년 11월 15일