Entwicklerhandbuch

AWS SDK für Go v2

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

AWS SDK für Go v2: Entwicklerhandbuch

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

Die Handelsmarken und Handelsaufmachung von Amazon dürfen nicht in einer Weise in Verbindung mit nicht von Amazon stammenden Produkten oder Services verwendet werden, durch die Kunden irregeführt werden könnten oder Amazon in schlechtem Licht dargestellt oder diskreditiert werden könnte. Alle anderen Handelsmarken, die nicht Eigentum von Amazon sind, gehören den jeweiligen Besitzern, die möglicherweise zu Amazon gehören oder nicht, mit Amazon verbunden sind oder von Amazon gesponsert werden.

Table of Contents

Was ist die AWS SDK für Go v2? ... 1
Wartung und Support für SDK-Hauptversionen ... 1
Erste Schritte ... 2
Holen Sie sich ein Amazon-Konto ... 2
Installiere die AWS SDK für Go Version 2 ... 2
Holen Sie sich Ihre AWS Zugangsschlüssel ... 3
Um Ihre Zugangsschlüssel-ID und Ihren geheimen Zugriffsschlüssel zu erhalten. ... 3
Verwandte Themen ... 4
Rufen Sie einen Vorgang auf ... 4
Das SDKs konfigurieren ... 6
AWS Gemeinsam genutzte Konfigurationsdateien werden geladen ... 6
Angabe der AWS Region ... 7
Region mit Umgebungsvariable konfigurieren ... 7
Region programmgesteuert angeben ... 7
Festlegen von Anmeldeinformationen ... 7
IAM-Rollen für Aufgaben ... 9
IAM-Rollen für Amazon-Instances EC2 ... 9
Gemeinsame Anmeldeinformationen und Konfiguration ... 9
Umgebungsvariablen ... 13
Geben Sie Anmeldeinformationen programmgesteuert an ... 13
Authentifizierung ... 16
Definitionen ... 16
Arbeitsablauf zur Auflösung des Authentifizierungsschemas ... 19
Wird nativ unterstützt `AuthScheme` ... 20
Client-Endpunkte ... 23
Anpassung ... 23
V2: + `EndpointResolverV2BaseEndpoint` ... 24
V1: `EndpointResolver` ... 28
Migration ... 31
HTTP-Client ... 35
Überschreiben beim Laden der Konfiguration ... 35
Zeitüberschreitung ... 35
Dialer ... 36
Transport ... 37

Protokollierung ... 40
Logger ... 40
ClientLogMode ... 41
Wiederholversuche und Zeitüberschreitungen ... 41
Standardtrockner ... 41
NopRetryer ... 42
Verhalten anpassen ... 42
Timeouts ... 46
Verwenden der SDK ... 48
Einen Service-Client erstellen ... 48
NewFromConfig ... 48
Neu ... 49
Service Operations aufrufen ... 51
Übergabe von Parametern an einen Dienstvorgang ... 51
Client-Optionen für Operation Call überschreiben ... 53
Umgang mit Betriebsantworten ... 53
Gleichzeitige Verwendung von Service-Clients ... 55
Verwendung von Operation Paginators ... 57
Verwenden von Waiter-Objekten ... 59
Die Kellner-Konfiguration wird außer Kraft gesetzt ... 60
Die erweiterte Kellnerkonfiguration hat Vorrang ... 62
Behandlung von Fehlern im SDK ... 63
Fehler protokollieren ... 64
Fehler beim Service Client ... 64
Anforderungskennungen werden abgerufen ... 66
AWS Dienste nutzen ... 68
Amazon S3 S3-Prüfsummen ... 68
Hochladen eines Objekts ... 69
Herunterladen eines Objekts ... 72
SDK-Dienstprogramme ... 74
Amazon RDS-Dienstprogramme ... 74
IAM-Authentifizierung ... 74
Amazon CloudFront Utilities ... 75
CloudFront Amazon-URL-Signaturer ... 75
Metadatenservice für EC2 Amazon-Instances ... 76
Amazon S3 S3-Dienstprogramme ... 77

Amazon S3 S3-Übertragungsmanager 77
Streaming-Eingabe nicht auffindbar 87
Middleware 89
Eine benutzerdefinierte Middleware schreiben 90
Middleware an alle Clients anhängen 92
Middleware an einen bestimmten Vorgang anhängen 92
Metadaten im Stapel weiterleiten 93
Vom SDK bereitgestellte Metadaten 95
Metadaten im Stapel nach oben weiterleiten 95
Häufig gestellte Fragen 98
Wie konfiguriere ich den HTTP-Client meines SDK? Gibt es Richtlinien oder bewährte Verfahren? 98
Wie sollte ich Betriebs-Timeouts konfigurieren? 98
Anfragen des SDK haben eine Zeitüberschreitung oder dauern zu lange. Wie behebe ich das? 99
Wie behebe ich einen `read: connection reset` Fehler? 99
Warum erhalte ich die Fehlermeldung „Ungültige Signatur“, wenn ich einen HTTP-Proxy mit dem SDK verwende? 100
Timing von SDK-Vorgängen 100
Einheitentests 107
Verspottung von Client-Vorgängen 107
Spöttische Paginatoren 109
Codebeispiele 112
API Gateway 113
AWS Beiträge der Gemeinschaft 113
Aurora 114
Grundlagen 116
Aktionen 134
Amazon Bedrock 154
Aktionen 134
Amazon Bedrock Runtime 157
Szenarien 161
AI21 Labore Jurassic-2 165
Amazon Titan Image Generator 167
Amazon Titan Text 170
Anthropic Claude 172

AWS CloudFormation ... 179
Aktionen ... 134
CloudWatch Logs ... 181
Aktionen ... 134
Amazon Cognito Identity Provider ... 183
Aktionen ... 134
Szenarien ... 161
Amazon DocumentDB ... 265
Serverless-Beispiele ... 265
DynamoDB ... 267
Grundlagen ... 116
Aktionen ... 134
Szenarien ... 161
Serverless-Beispiele ... 265
AWS Beiträge der Gemeinschaft ... 113
IAM ... 340
Grundlagen ... 116
Aktionen ... 134
Kinesis ... 401
Serverless-Beispiele ... 265
Lambda ... 404
Grundlagen ... 116
Aktionen ... 134
Szenarien ... 161
Serverless-Beispiele ... 265
AWS Beiträge der Gemeinschaft ... 113
Amazon MSK ... 516
Serverless-Beispiele ... 265
Partner Central ... 517
Aktionen ... 134
Amazon RDS ... 520
Grundlagen ... 116
Aktionen ... 134
Serverless-Beispiele ... 265
Amazon Redshift ... 556
Grundlagen ... 116

Aktionen 134
Amazon S3 575
Grundlagen 116
Aktionen 134
Szenarien 161
Serverless-Beispiele 265
Amazon SNS 665
Aktionen 134
Szenarien 161
Serverless-Beispiele 265
Amazon SQS 693
Aktionen 134
Szenarien 161
Serverless-Beispiele 265
Migrieren Sie zu v2 726
Minimale Go-Version 726
Modularisierung 726
Konfiguration wird geladen 727
Beispiele 26
Spott und `*iface` 730
Anmeldeinformationen und Anbieter von Anmeldeinformationen 731
Statische Anmeldeinformationen 14
Anmeldeinformationen für Amazon EC2 IAM-Rollen 733
Anmeldeinformationen für Endgeräte 734
Anmeldeinformationen verarbeiten 734
AWS Security Token Service Erweitern Sie im angezeigten Detailbereich die Option 735
Kunden bedienen 738
Aufbau eines Kunden 738
Endpunkte 740
Authentifizierung 741
API-Operationen aufrufen 741
Service-Datentypen 742
Zeiger-Parameter 743
Arten von Fehlern 743
Paginatoren 745
Waiter 747

Vorsignierte Anfragen 747
Anpassung der Anfrage 749
Operation Eingabe/Ausgabe 749
HTTP-Anfrage/Antwort 754
Handler-Phasen 756
Features 758
Metadatenservice für EC2 Amazon-Instances 758
Amazon S3 Transfer-Manager 759
Amazon CloudFront Signing Utilities 759
Amazon S3 Encryption Client 759
Änderungen an den Serviceanpassungen 760
Amazon S3 760
Sicherheit 762
Datenschutz 762
Compliance-Validierung 764
Ausfallsicherheit 765
Dokumentverlauf 766
..... dcclxvii

Was ist die AWS SDK für Go v2?

Die AWS SDK für Go Version v2 bietet APIs Dienstprogramme, mit denen Entwickler Go-Anwendungen erstellen können, die AWS Dienste wie Amazon Elastic Compute Cloud (Amazon EC2) und Amazon Simple Storage Service (Amazon S3) nutzen.

Das SDK beseitigt die Komplexität der Codierung direkt an einer Webservice-Schnittstelle. Es verbirgt viele der untergeordneten Funktionen, wie Authentifizierung, erneute Versuche von Anfragen und Fehlerbehandlung.

Das SDK enthält auch hilfreiche Hilfsprogramme. Beispielsweise kann der Amazon S3 S3-Download- und Upload-Manager große Objekte automatisch in mehrere Teile zerlegen und parallel übertragen.

Verwenden Sie das AWS SDK für Go Entwicklerhandbuch, um Ihnen bei der Installation, Konfiguration und Verwendung des SDK zu helfen. Dieses Handbuch enthält Konfigurationsinformationen, Beispielcode und eine Einführung in die SDK-Dienstprogramme.

Wartung und Support für SDK-Hauptversionen

Informationen zur Wartung und zum Support für SDK-Hauptversionen und die ihnen zugrunde liegenden Abhängigkeiten finden Sie im Referenzhandbuch AWS SDKs und im Tools-Referenzhandbuch:

- AWS SDKs Richtlinien zur Wartung von Tools
- AWS SDKs und Matrix zur Unterstützung der Tools-Versionen

Fangen Sie an mit AWS SDK für Go

Das AWS SDK für Go erfordert Go 1.20 oder höher. Sie können Ihre aktuelle Version von Go anzeigen, indem Sie den folgenden Befehl ausführen:

```
go version
```

Informationen zur Installation oder Aktualisierung Ihrer Version von Go finden Sie unter https://golang.org/doc/install.

Holen Sie sich ein Amazon-Konto

Bevor Sie die AWS SDK für Go v2 verwenden können, müssen Sie über ein Amazon-Konto verfügen. Weitere Informationen findest du unter Wie erstelle und aktiviere ich ein neues AWS Konto? für Einzelheiten.

Installiere die AWS SDK für Go Version 2

Die AWS SDK für Go Version 2 verwendet Go-Module, eine Funktion, die in Go 1.11 eingeführt wurde. Initialisieren Sie Ihr lokales Projekt, indem Sie den folgenden Go-Befehl ausführen.

```
go mod init example
```

Nachdem Sie Ihr Go-Modulprojekt initialisiert haben, können Sie das SDK und die erforderlichen Abhängigkeiten mithilfe des `go get` Befehls abrufen. Diese Abhängigkeiten werden in der `go.mod` Datei aufgezeichnet, die mit dem vorherigen Befehl erstellt wurde.

Die folgenden Befehle zeigen, wie Sie den Standardsatz von SDK-Modulen abrufen, die in Ihrer Anwendung verwendet werden sollen.

```
go get github.com/aws/aws-sdk-go-v2
go get github.com/aws/aws-sdk-go-v2/config
```

Dadurch werden das Kern-SDK-Modul und das Konfigurationsmodul abgerufen, das zum Laden der AWS gemeinsam genutzten Konfiguration verwendet wird.

Als Nächstes können Sie einen oder mehrere AWS Service-API-Clients installieren, die für Ihre Anwendung erforderlich sind. Alle API-Clients befinden sich in der `github.com/aws/aws-sdk-go-`

`v2/service` Importhierarchie. Eine vollständige Liste der derzeit unterstützten API-Clients finden Sie hier. Um einen Service-Client zu installieren, führen Sie den folgenden Befehl aus, um das Modul abzurufen und die Abhängigkeit in Ihrer `go.mod` Datei aufzuzeichnen. In diesem Beispiel rufen wir den Amazon S3 S3-API-Client ab.

```
go get github.com/aws/aws-sdk-go-v2/service/s3
```

Holen Sie sich Ihre AWS Zugangsschlüssel

Zugriffsschlüssel bestehen aus einer Zugriffsschlüssel-ID und einem geheimen Zugriffsschlüssel. Diese werden zum Signieren der von Ihnen ausgeführten programmgesteuerten Anforderungen an AWS verwendet. Wenn Sie keine Zugriffsschlüssel haben, können Sie sie mithilfe der AWS Management Console erstellen. Wir empfehlen, IAM-Zugriffsschlüssel anstelle von AWS Root-Kontozugriffsschlüsseln zu verwenden. Mit IAM können Sie den Zugriff auf AWS Dienste und Ressourcen in Ihrem AWS Konto sicher kontrollieren.

> **Note**
>
> Um Zugriffsschlüssel zu erstellen, müssen Sie Berechtigungen zum Ausführen der erforderlichen IAM-Aktionen besitzen. Weitere Informationen finden Sie unter Erteilen von IAM-Benutzerberechtigungen zur Verwaltung von Kennwortrichtlinien und Anmeldeinformationen im IAM-Benutzerhandbuch.

Um Ihre Zugangsschlüssel-ID und Ihren geheimen Zugriffsschlüssel zu erhalten.

1. Öffnen Sie die IAM-Konsole
2. Wählen Sie im Navigationsmenü Users (Benutzer).
3. Wählen Sie Ihren IAM-Benutzernamen (nicht das Kontrollkästchen).
4. Öffnen Sie die Registerkarte Security credentials (Sicherheitsanmeldeinformationen) und wählen Sie Create access key (Zugriffsschlüssel erstellen) aus.
5. Wählen Sie zum Anzeigen des neuen Zugriffsschlüssels Show aus. Ihre Anmeldeinformationen sehen in etwa wie folgt aus:
   - Zugriffsschlüssel-ID: `AKIAIOSFODNN7EXAMPLE`
   - Geheimer Zugriffsschlüssel: `wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY`

6. Wählen Sie zum Herunterladen des Schlüsselpaares Download .csv file aus. Speichern Sie die Schlüssel an einem sicheren Ort.

> **Warning**
>
> Behandeln Sie die Schlüssel vertraulich, um Ihr AWS Konto zu schützen, und geben Sie sie niemals an Dritte außerhalb Ihrer Organisation weiter.

Verwandte Themen

- Was ist IAM? im IAM-Benutzerhandbuch.
- AWS Allgemeine Referenz zu Sicherheitsanmeldedaten in Amazon Web Services.

Rufen Sie einen Vorgang auf

Nachdem Sie das SDK installiert haben, importieren Sie AWS Pakete in Ihre Go-Anwendungen, um das SDK zu verwenden, wie im folgenden Beispiel gezeigt, das die Bibliotheken AWS, Config und Amazon S3 importiert. Nach dem Import der SDK-Pakete wird die gemeinsame AWS SDK-Konfiguration geladen, ein Client erstellt und eine API-Operation aufgerufen.

```
package main

import (
    "context"
    "log"
    "github.com/aws/aws-sdk-go-v2/aws"
    "github.com/aws/aws-sdk-go-v2/config"
    "github.com/aws/aws-sdk-go-v2/service/s3"
)

func main() {
    // Load the Shared AWS Configuration (~/.aws/config)
    cfg, err := config.LoadDefaultConfig(context.TODO())
    if err != nil {
        log.Fatal(err)
    }

    // Create an Amazon S3 service client
```

```
    client := s3.NewFromConfig(cfg)

    // Get the first page of results for ListObjectsV2 for a bucket
    output, err := client.ListObjectsV2(context.TODO(), &s3.ListObjectsV2Input{
        Bucket: aws.String("amzn-s3-demo-bucket"),
    })
    if err != nil {
        log.Fatal(err)
    }

    log.Println("first page results")
    for _, object := range output.Contents {
        log.Printf("key=%s size=%d", aws.ToString(object.Key), *object.Size)
    }
}
```

Das SDKs konfigurieren

In der AWS SDK für Go V2 können Sie allgemeine Einstellungen für Service-Clients konfigurieren, z. B. die Logger-, Log Level- und Wiederholungskonfiguration. Die meisten Einstellungen sind optional. Für jeden Service-Client müssen Sie jedoch eine AWS Region und Ihre Anmeldeinformationen angeben. Das SDK verwendet diese Werte, um Anfragen an die richtige Region zu senden und Anfragen mit den richtigen Anmeldeinformationen zu signieren. Sie können diese Werte programmgesteuert im Code oder über die Ausführungsumgebung angeben.

AWS Gemeinsam genutzte Konfigurationsdateien werden geladen

Es gibt eine Reihe von Möglichkeiten, einen Service-API-Client zu initialisieren, aber das folgende Muster wird Benutzern am häufigsten empfohlen.

Verwenden Sie den folgenden Code, um das SDK für die Verwendung der AWS gemeinsam genutzten Konfigurationsdateien zu konfigurieren:

```
import (
  "context"
  "log"
  "github.com/aws/aws-sdk-go-v2/config"
)

// ...

cfg, err := config.LoadDefaultConfig(context.TODO())
if err != nil {
  log.Fatalf("failed to load configuration, %v", err)
}
```

`config.LoadDefaultConfig(context.TODO())`erstellt mithilfe der AWS gemeinsam genutzten Konfigurationsquellen eine AWS.Config. Dazu gehören die Konfiguration eines Anmeldeinformationsanbieters, die Konfiguration der AWS Region und das Laden der dienstspezifischen Konfiguration. Dienstclients können mithilfe der geladenen Datei erstellt werden`aws.Config`, wodurch ein konsistentes Muster für die Erstellung von Clients bereitgestellt wird.

Weitere Informationen zu AWS gemeinsam genutzten Konfigurationsdateien finden Sie unter Konfiguration im Referenzhandbuch AWS SDKs und im Tools-Referenzhandbuch.

Angabe der AWS Region

Wenn Sie die Region angeben, geben Sie an, wohin Anfragen gesendet werden sollen, z. B. `us-west-2` oder`us-east-2`. Eine Liste der Regionen für jeden Dienst finden Sie unter Dienstendpunkte und Kontingente in der Allgemeine Amazon Web Services-Referenz.

Das SDK hat keine Standardregion. Um eine Region anzugeben:

- Setzen Sie die `AWS_REGION` Umgebungsvariable auf die Standardregion.
- Stellen Sie die Region explizit mit config ein. WithRegionals Argument für `config.LoadDefaultConfig` beim Laden der Konfiguration.

RÜCKBLICK: Wenn Sie eine Region mit all diesen Techniken festlegen, verwendet das SDK die Region, die Sie explizit angegeben haben.

Region mit Umgebungsvariable konfigurieren

Linux, macOS oder Unix

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

Windows

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

Region programmgesteuert angeben

```
cfg, err := config.LoadDefaultConfig(context.TODO(), config.WithRegion("us-west-2"))
```

Festlegen von Anmeldeinformationen

AWS SDK für Go Für das Signieren von Anfragen sind Anmeldeinformationen (ein Zugriffsschlüssel und ein geheimer Zugriffsschlüssel) erforderlich. AWS Sie können Ihre Anmeldeinformationen je nach Ihrem speziellen Anwendungsfall an verschiedenen Stellen angeben. Hinweise zum Abrufen von Anmeldeinformationen finden Sie unterFangen Sie an mit AWS SDK für Go.

Wenn Sie eine `aws.Config` Instanz mithilfe initialisieren`config.LoadDefaultConfig`, verwendet das SDK seine standardmäßige Anmeldeinformationskette, um Anmeldeinformationen zu finden

AWS . Diese standardmäßige Anmeldeinformationskette sucht in der folgenden Reihenfolge nach Anmeldeinformationen:

1. Umgebungsvariablen.
   1. Statische Anmeldeinformationen (`AWS_ACCESS_KEY_ID`,`AWS_SECRET_ACCESS_KEY`,`AWS_SESSION_TOKEN`)
   2. Web-Identitätstoken (`AWS_WEB_IDENTITY_TOKEN_FILE`)
2. Gemeinsam genutzte Konfigurationsdateien.
   1. SDK verwendet standardmäßig den Ordner `credentials` Datei unter dem `.aws` Ordner, der sich im Home-Ordner auf Ihrem Computer befindet.
   2. SDK verwendet standardmäßig den Ordner `config` File unter dem `.aws` Ordner, der sich im Home-Ordner auf Ihrem Computer befindet.
3. Wenn Ihre Anwendung eine Amazon ECS-Aufgabendefinition oder einen RunTask API-Vorgang verwendet, IAM-Rolle für Aufgaben.
4. Wenn Ihre Anwendung auf einer EC2 Amazon-Instance ausgeführt wird, IAM-Rolle für Amazon EC2.

Das SDK erkennt und verwendet die integrierten Anbieter automatisch, ohne dass manuelle Konfigurationen erforderlich sind. Wenn Sie beispielsweise IAM-Rollen für EC2 Amazon-Instances verwenden, verwenden Ihre Anwendungen automatisch die Anmeldeinformationen der Instance. Sie müssen die Anmeldeinformationen in Ihrer Anwendung nicht manuell konfigurieren.

Als bewährte Methode AWS empfiehlt es sich, die Anmeldeinformationen in der folgenden Reihenfolge anzugeben:

1. Verwenden Sie IAM-Rollen für Aufgaben, wenn Ihre Anwendung eine Amazon ECS-Aufgabendefinition oder einen RunTask API-Vorgang verwendet.
2. Verwenden Sie IAM-Rollen für Amazon EC2 (wenn Ihre Anwendung auf einer EC2 Amazon-Instance läuft).

   IAM-Rollen stellen Anwendungen auf der Instance temporäre Sicherheitsanmeldedaten zur Verfügung, um Aufrufe zu tätigen AWS . IAM-Rollen bieten eine einfache Möglichkeit, Anmeldeinformationen auf mehreren EC2 Amazon-Instances zu verteilen und zu verwalten.
3. Verwenden Sie gemeinsam genutzte Anmeldeinformationen oder Konfigurationsdateien.

   Die Anmeldeinformationen und Konfigurationsdateien werden von anderen AWS SDKs und gemeinsam genutzt AWS CLI. Aus Sicherheitsgründen empfehlen wir, die Anmeldeinformationsdatei für die Festlegung sensibler Werte wie Zugriffsschlüssel IDs und

geheime Schlüssel zu verwenden. Hier sind die Formatierungsanforderungen für jede dieser Dateien.

4. Umgebungsvariablen verwenden.

   Das Setzen von Umgebungsvariablen ist nützlich, wenn Sie Entwicklungsarbeiten auf einem anderen Computer als einer EC2 Amazon-Instance durchführen.

IAM-Rollen für Aufgaben

Wenn Ihre Anwendung eine Amazon ECS-Aufgabendefinition oder einen `RunTask` Vorgang verwendet, verwenden Sie IAM-Rollen für Aufgaben, um eine IAM-Rolle anzugeben, die von den Containern in einer Aufgabe verwendet werden kann.

IAM-Rollen für Amazon-Instances EC2

Wenn Sie Ihre Anwendung auf einer EC2 Amazon-Instance ausführen, verwenden Sie die IAM-Rolle der Instance, um temporäre Sicherheitsanmeldeinformationen für Aufrufe abzurufen AWS.

Wenn Sie Ihre Instance für die Verwendung von IAM-Rollen konfiguriert haben, verwendet das SDK diese Anmeldeinformationen automatisch für Ihre Anwendung. Sie müssen diese Anmeldeinformationen nicht manuell angeben.

Gemeinsame Anmeldeinformationen und Konfiguration

Die gemeinsamen Anmeldeinformationen und Konfigurationsdateien können verwendet werden, um gemeinsame Konfigurationen AWS SDKs und andere Tools gemeinsam zu nutzen. Wenn Sie für unterschiedliche Tools und Anwendungen verschiedene Anmeldeinformationen verwenden, können Sie mithilfe von Profilen mehrere Zugriffsschlüssel innerhalb ein und derselben Konfigurationsdatei konfigurieren.

Sie können mehrere Speicherorte für Anmelde- oder Konfigurationsdateien angeben. Standardmäßig lädt das SDK Dateien`config.LoadOptions`, die an den in der Festlegen von Anmeldeinformationen genannten Standardspeicherorten gespeichert sind.

```
import (
    "context"
    "github.com/aws/aws-sdk-go-v2/config"
)

// ...
```

```
cfg , err := config.LoadDefaultConfig(context.TODO(),
    config.WithSharedCredentialsFiles(
    []string{"test/credentials", "data/credentials"},
    ),
    config.WithSharedConfigFiles(
        []string{"test/config", "data/config"},
    )
)
```

Wenn Sie mit gemeinsam genutzten Anmeldeinformationen und Konfigurationsdateien arbeiten und doppelte Profile angegeben werden, werden diese zusammengeführt, um ein Profil aufzulösen. Im Falle eines Zusammenführungskonflikts

1. Wenn doppelte Profile in derselben Anmeldeinformationen/ Konfigurationsdatei angegeben werden, haben die in letzterem Profil angegebenen Profileigenschaften Vorrang.
2. Wenn doppelte Profile entweder für mehrere Anmeldeinformationsdateien oder für mehrere Konfigurationsdateien angegeben werden, werden die Profileigenschaften gemäß der Reihenfolge der Dateieingabe in die aufgelöst. `config.LoadOptions` Die Profileigenschaften in den letztgenannten Dateien haben Vorrang.
3. Wenn ein Profil sowohl in der Anmeldeinformationsdatei als auch in der Konfigurationsdatei vorhanden ist, haben die Eigenschaften der Anmeldeinformationsdatei Vorrang.

Bei Bedarf können Sie die Schritte `LogConfigurationWarnings` zur Profilauflösung aktivieren `config.LoadOptions` und protokollieren.

Die Datei mit den Anmeldeinformationen wird erstellt

Wenn Sie keine gemeinsame Anmeldeinformationsdatei (`.aws/credentials`) haben, können Sie einen beliebigen Texteditor verwenden, um eine Datei in Ihrem Home-Verzeichnis zu erstellen. Fügen Sie Ihrer Anmeldeinformationsdatei den folgenden Inhalt hinzu *<YOUR_ACCESS_KEY_ID>* und *<YOUR_SECRET_ACCESS_KEY>* ersetzen Sie dabei Ihre Anmeldedaten.

```
[default]
aws_access_key_id = <YOUR_ACCESS_KEY_ID>
aws_secret_access_key = <YOUR_SECRET_ACCESS_KEY>
```

In der `[default]` Überschrift werden die Anmeldeinformationen für das Standardprofil definiert, die das SDK verwendet, sofern Sie es nicht für die Verwendung eines anderen Profils konfigurieren.

Sie können auch temporäre Sicherheitsanmeldedaten verwenden, indem Sie die Sitzungstoken zu Ihrem Profil hinzufügen, wie im folgenden Beispiel gezeigt:

```
[temp]
aws_access_key_id = <YOUR_TEMP_ACCESS_KEY_ID>
aws_secret_access_key = <YOUR_TEMP_SECRET_ACCESS_KEY>
aws_session_token = <YOUR_SESSION_TOKEN>
```

Der Abschnittsname für ein nicht standardmäßiges Profil in einer Anmeldeinformationsdatei darf nicht mit dem Wort `profile` beginnen. Weitere Informationen finden Sie im AWS SDKs Referenzhandbuch zu Tools.

Die Config erstellen

Wenn Sie keine gemeinsame Anmeldeinformationsdatei (`.aws/config`) haben, können Sie einen beliebigen Texteditor verwenden, um eine Datei in Ihrem Home-Verzeichnis zu erstellen. Fügen Sie Ihrer Konfigurationsdatei den folgenden Inhalt hinzu und *<REGION>* ersetzen Sie ihn durch die gewünschte Region.

```
[default]
region = <REGION>
```

Die `[default]` Überschrift definiert die Konfiguration für das Standardprofil, das das SDK verwendet, sofern Sie es nicht für die Verwendung eines anderen Profils konfigurieren.

Sie können benannte Profile verwenden, wie im folgenden Beispiel gezeigt:

```
[profile named-profile]
region = <REGION>
```

Der Abschnittsname für ein nicht standardmäßiges Profil in einer Konfigurationsdatei muss immer mit dem Wort beginnen`profile`, gefolgt vom beabsichtigten Profilnamen. Weitere Informationen finden Sie im Referenzhandbuch AWS SDKs und im Tools-Referenzhandbuch.

Profile angeben

Sie können mehrere Zugriffsschlüssel in dieselbe Konfigurationsdatei aufnehmen, indem Sie jeden Satz von Zugriffsschlüsseln einem Profil zuordnen. In Ihrer Anmeldeinformationsdatei können Sie beispielsweise mehrere Profile wie folgt deklarieren.

```
[default]
aws_access_key_id = <YOUR_DEFAULT_ACCESS_KEY_ID>
aws_secret_access_key = <YOUR_DEFAULT_SECRET_ACCESS_KEY>

[test-account]
aws_access_key_id = <YOUR_TEST_ACCESS_KEY_ID>
aws_secret_access_key = <YOUR_TEST_SECRET_ACCESS_KEY>

[prod-account]
; work profile
aws_access_key_id = <YOUR_PROD_ACCESS_KEY_ID>
aws_secret_access_key = <YOUR_PROD_SECRET_ACCESS_KEY>
```

Standardmäßig prüft das SDK die Umgebungsvariable `AWS_PROFILE`, um zu bestimmen, welches Profil verwendet werden soll. Wenn keine `AWS_PROFILE` Variable festgelegt ist, verwendet das SDK das `default` Profil.

Manchmal möchten Sie vielleicht ein anderes Profil für Ihre Anwendung verwenden. Sie möchten beispielsweise die `test-account` Anmeldeinformationen mit Ihrer `myapp` Anwendung verwenden. Sie können dieses Profil mit dem folgenden Befehl verwenden:

```
$ AWS_PROFILE=test-account myapp
```

Sie können das SDK auch anweisen, ein Profil auszuwählen`config.LoadDefaultConfig`, indem Sie es entweder `os.Setenv("AWS_PROFILE", "test-account")` vor dem Aufruf aufrufen oder indem Sie ein explizites Profil als Argument übergeben, wie im folgenden Beispiel gezeigt:

```
cfg, err := config.LoadDefaultConfig(context.TODO(),
    config.WithSharedConfigProfile("test-account"))
```

> **Note**
>
> Wenn Sie Anmeldeinformationen in Umgebungsvariablen angeben, verwendet das SDK immer diese Anmeldeinformationen, unabhängig davon, welches Profil Sie angeben.

Umgebungsvariablen

Standardmäßig erkennt das SDK die in Ihrer Umgebung festgelegten AWS Anmeldeinformationen und verwendet sie, um Anfragen zu signieren AWS. Auf diese Weise müssen Sie die Anmeldeinformationen in Ihren Anwendungen nicht verwalten.

Das SDK sucht in den folgenden Umgebungsvariablen nach Anmeldeinformationen:

- `AWS_ACCESS_KEY_ID`
- `AWS_SECRET_ACCESS_KEY`
- `AWS_SESSION_TOKEN` (optional)

Die folgenden Beispiele zeigen, wie Sie die Umgebungsvariablen konfigurieren.

Linux, OS X oder Unix

```
$ export AWS_ACCESS_KEY_ID=YOUR_AKID
$ export AWS_SECRET_ACCESS_KEY=YOUR_SECRET_KEY
$ export AWS_SESSION_TOKEN=TOKEN
```

Windows

```
> set AWS_ACCESS_KEY_ID=YOUR_AKID
> set AWS_SECRET_ACCESS_KEY=YOUR_SECRET_KEY
> set AWS_SESSION_TOKEN=TOKEN
```

Geben Sie Anmeldeinformationen programmgesteuert an

`config.LoadDefaultConfig`ermöglicht es Ihnen, eine explizite AWS anzugeben. CredentialProviderbeim Laden der gemeinsam genutzten Konfigurationsquellen. Verwenden Sie config, um beim Laden der gemeinsam genutzten Konfiguration einen expliziten Anbieter für Anmeldeinformationen zu übergeben. WithCredentialsProvider. Wenn beispielsweise `customProvider` auf eine `aws.CredentialProvider` Implementierungsinstanz verwiesen wird, kann sie beim Laden der Konfiguration wie folgt übergeben werden:

```
cfg, err := config.LoadDefaultConfig(context.TODO(),
    config.WithCredentialsProvider(customProvider))
```

Wenn Sie wie in diesem Beispiel explizit Anmeldeinformationen angeben, verwendet das SDK nur diese Anmeldeinformationen.

> **Note**
>
> Alle Anmeldeinformationsanbieter, die an übergeben oder von zurückgegeben wurden, `LoadDefaultConfig` werden CredentialsCacheautomatisch in eine eingebunden. Dies ermöglicht das Zwischenspeichern und die Rotation von Anmeldeinformationen, was die Parallelität gewährleistet. Wenn Sie einen Anbieter explizit `aws.Config` direkt konfigurieren, müssen Sie den Anbieter auch explizit mit diesem Typ umschließen, indem Sie NewCredentialsCache

Statische Anmeldeinformationen

Sie können Anmeldeinformationen in Ihrer Anwendung fest codieren, indem Sie die Anmeldeinformationen verwenden. NewStaticCredentialsProviderAnmeldeinformationsanbieter, um die zu verwendenden Zugriffsschlüssel explizit festzulegen. Zum Beispiel:

```
cfg, err := config.LoadDefaultConfig(context.TODO(),
    config.WithCredentialsProvider(credentials.NewStaticCredentialsProvider("AKID",
 "SECRET_KEY", "TOKEN")),
)
```

> **Warning**
>
> Betten Sie keine Anmeldeinformationen in eine Anwendung ein. Verwenden Sie diese Methode nur zu Testzwecken.

Anmeldeinformationen für Single Sign-On

Das SDK bietet einen Anbieter für Anmeldeinformationen zum Abrufen temporärer AWS Anmeldeinformationen mithilfe von. AWS IAM Identity Center Mit dem AWS CLI authentifizieren Sie sich beim AWS Zugriffsportal und autorisieren den Zugriff auf temporäre Anmeldeinformationen. AWS Anschließend konfigurieren Sie Ihre Anwendung so, dass das Single Sign-On (SSO) -Profil geladen wird, und das SDK verwendet Ihre SSO-Anmeldeinformationen, um temporäre AWS Anmeldeinformationen abzurufen, die bei Ablauf automatisch erneuert werden. Wenn Ihre SSO-

Anmeldeinformationen ablaufen, müssen Sie sie ausdrücklich erneuern, indem Sie sich erneut mit dem bei Ihrem IAM Identity Center-Konto anmelden. AWS CLI

Sie können beispielsweise ein Profil erstellen`dev-profile`, dieses Profil mit dem authentifizieren und autorisieren und Ihre Anwendung wie unten gezeigt konfigurieren. AWS CLI

1. Erstellen Sie zuerst das und `profile` `sso-session`

```
[profile dev-profile]
sso_session = dev-session
sso_account_id = 012345678901
sso_role_name = Developer
region = us-east-1

[sso-session dev-session]
sso_region = us-west-2
sso_start_url = https://company-sso-portal.awsapps.com/start
sso_registration_scopes = sso:account:access
```

2. Melden Sie sich mit dem AWS CLI an, um das SSO-Profil zu authentifizieren und zu autorisieren.

```
$ aws --profile dev-profile sso login
Attempting to automatically open the SSO authorization page in your default browser.
If the browser does not open or you wish to use a different device to authorize this
 request, open the following URL:

https://device.sso.us-west-2.amazonaws.com/

Then enter the code:

ABCD-EFGH
Successully logged into Start URL: https://company-sso-portal.awsapps.com/start
```

3. Als Nächstes konfigurieren Sie Ihre Anwendung so, dass sie das SSO-Profil verwendet.

```
import "github.com/aws/aws-sdk-go-v2/config"

// ...
```

```
cfg, err := config.LoadDefaultConfig(
    context.Background(),
    config.WithSharedConfigProfile("dev-profile"),
)
if err != nil {
    return err
}
```

Weitere Informationen zur Konfiguration von SSO-Profilen und zur Authentifizierung mithilfe des AWS CLI finden Sie unter Konfiguration der AWS CLI zu AWS IAM Identity Center verwendenden Profile im AWS CLI Benutzerhandbuch. Weitere Informationen zur programmgesteuerten Erstellung des SSO-Anmeldeinformationsanbieters finden Sie in der Referenzdokumentation zur ssocreds-API.

Andere Anbieter von Anmeldeinformationen

Das SDK bietet andere Methoden zum Abrufen von Anmeldeinformationen im Modul für Anmeldeinformationen. Sie können beispielsweise temporäre Sicherheitsanmeldedaten aus AWS Security Token Service oder Anmeldeinformationen aus einem verschlüsselten Speicher abrufen.

Verfügbare Anbieter von Anmeldeinformationen:

- ec2rolecreds — Rufen Sie Anmeldeinformationen von Amazon EC2 Instance-Rollen über Amazon IMDS ab. EC2
- endpointcreds — Ruft Anmeldeinformationen von einem beliebigen HTTP-Endpunkt ab.
- processcreds — Ruft Anmeldeinformationen von einem externen Prozess ab, der von der Shell der Host-Umgebung aufgerufen wird.
- stscreds — Ruft Anmeldeinformationen ab von AWS STS

Authentifizierung konfigurieren

Der AWS SDK für Go bietet die Möglichkeit, den Dienst für das Authentifizierungsverhalten zu konfigurieren. In den meisten Fällen ist die Standardkonfiguration ausreichend, aber die Konfiguration der benutzerdefinierten Authentifizierung ermöglicht zusätzliche Funktionen, wie z. B. die Verwendung von Servicefunktionen vor der Veröffentlichung.

Definitionen

Dieser Abschnitt enthält eine allgemeine Beschreibung der Authentifizierungskomponenten in der. AWS SDK für Go

AuthScheme

An AuthSchemeist die Schnittstelle, die den Workflow definiert, über den das SDK eine Anruferidentität abruft und sie an eine Betriebsanforderung anhängt.

Ein Authentifizierungsschema verwendet die folgenden Komponenten, die weiter unten ausführlich beschrieben werden:

- Eine eindeutige ID, die das Schema identifiziert
- Ein Identity Resolver, der eine beim Signieren verwendete Anruferidentität zurückgibt (z. B. Ihre AWS Anmeldeinformationen)
- Ein Unterzeichner, der die Identität des Anrufers tatsächlich in die Transportanfrage des Vorgangs einfügt (z. B. den HTTP-Header) `Authorization`

Jede Service-Client-Option enthält ein `AuthSchemes` Feld, das standardmäßig mit der Liste der von diesem Dienst unterstützten Authentifizierungsschemata gefüllt ist.

AuthSchemeResolver

Jede Service-Client-Option enthält ein `AuthSchemeResolver` Feld. Diese pro Dienst definierte Schnittstelle ist die API, die vom SDK aufgerufen wird, um die möglichen Authentifizierungsoptionen für jeden Vorgang zu ermitteln.

> **Important**
>
> Der Auth-Schema-Resolver schreibt NICHT vor, welches Authentifizierungsschema verwendet wird. Er gibt eine Liste von Schemata zurück, die verwendet werden können („Optionen“). Das endgültige Schema wird durch einen festen Algorithmus ausgewählt, der hier beschrieben wird.

Option

Eine Option, die von einem Aufruf von zurückgegeben wird`ResolverAuthSchemes`, stellt eine mögliche Authentifizierungsoption dar.

Eine Option besteht aus drei Informationssätzen:

- Eine ID, die das mögliche Schema darstellt

- Ein undurchsichtiger Satz von Eigenschaften, der dem Identity Resolver des Schemas zur Verfügung gestellt werden soll
- Ein undurchsichtiger Satz von Eigenschaften, der dem Unterzeichner des Schemas zur Verfügung gestellt werden soll

Ein Hinweis zu Eigenschaften

In 99% der Anwendungsfälle müssen sich Anrufer keine Gedanken über die undurchsichtigen Eigenschaften bei der Identitätsauflösung und beim Signieren machen. Das SDK ruft die erforderlichen Eigenschaften für jedes Schema ab und übergibt sie an die stark typisierten Schnittstellen, die im SDK verfügbar sind. Beispielsweise codiert der standardmäßige Auth Resolver für Dienste die Option Sigv4 so, dass sie über Unterzeichnereigenschaften für den Signaturnamen und die Region verfügt, deren Werte an die vom Client konfigurierte Version v4 übergeben werden. HTTPSignerImplementierung, wenn SigV4 ausgewählt ist.

Identität

Eine Identität ist eine abstrakte Darstellung dessen, wer der SDK-Aufrufer ist.

Der im SDK am häufigsten verwendete Identitätstyp ist eine Reihe von`aws.Credentials`. In den meisten Anwendungsfällen muss sich der Aufrufer nicht mit `Identity` einer Abstraktion befassen und kann direkt mit den konkreten Typen arbeiten.

> **Note**
>
> Um die Abwärtskompatibilität zu wahren und API-Verwirrung zu vermeiden, erfüllt der AWS SDK-spezifische Identitätstyp die Schnittstelle `aws.Credentials` nicht direkt. `Identity` Diese Zuordnung wird intern abgewickelt.

IdentityResolver

IdentityResolverist die Schnittstelle, über die ein abgerufen `Identity` wird.

Konkrete Versionen von `IdentityResolver` existieren im SDK in stark typisierter Form (z. B. aws. CredentialsProvider), das SDK verarbeitet diese Zuordnung intern.

Ein Aufrufer muss die `IdentityResolver` Schnittstelle nur direkt implementieren, wenn er ein externes Authentifizierungsschema definiert.

Signer

Der Unterzeichner ist die Schnittstelle, über die eine Anfrage um den abgerufenen Anrufer ergänzt wird. `Identity`

Konkrete Versionen von `Signer` existieren im SDK in stark typisierter Form (z. B. v4). HTTPSigner), das SDK verarbeitet diese Zuordnung intern.

Ein Aufrufer muss die `Signer` Schnittstelle nur direkt implementieren, wenn er ein externes Authentifizierungsschema definiert.

AuthResolverParameters

Jeder Dienst benötigt einen bestimmten Satz von Eingaben, die an seine Auflösungsfunktion übergeben werden, die in jedem Servicepaket als definiert ist. `AuthResolverParameters`

Die Basis-Resolverparameter lauten wie folgt:

Name	Typ	description
`Operation`	`string`	Der Name der Operation, die aufgerufen wird.
`Region`	`string`	Die AWS Region des Kunden. Nur für Dienste vorhanden, die SigV4 [A] verwenden.

Wenn Sie Ihren eigenen Resolver implementieren, sollten Sie niemals eine eigene Instanz seiner Parameter erstellen müssen. Das SDK bezieht diese Werte pro Anfrage und leitet sie an Ihre Implementierung weiter.

Arbeitsablauf zur Auflösung des Authentifizierungsschemas

Wenn Sie einen AWS Dienstvorgang über das SDK aufrufen, erfolgt die folgende Abfolge von Aktionen, nachdem die Anforderung serialisiert wurde:

1. Das SDK ruft die `AuthSchemeResolver.ResolveAuthSchemes()` API des Clients auf und bezieht die Eingabeparameter nach Bedarf, um eine Liste möglicher Optionen für den Vorgang zu erhalten.

2. Das SDK durchläuft diese Liste und wählt das erste Schema aus, das die folgenden Bedingungen erfüllt.
   - Ein Schema mit übereinstimmender ID ist in der eigenen Liste des Kunden vorhanden `AuthSchemes`
   - Der Identity Resolver des Schemas existiert in den Optionen des Clients (ist nicht vorhanden`nil`) (wird über die `GetIdentityResolver` Methode des Schemas überprüft, die Zuordnung zu den oben beschriebenen konkreten Identity-Resolver-Typen erfolgt intern) (1)
3. Unter der Annahme, dass ein praktikables Schema ausgewählt wurde, ruft das SDK seine `GetIdentityResolver()` API auf, um die Identität des Aufrufers abzurufen. Beispielsweise wird das integrierte SigV4-Authentifizierungsschema intern dem Anbieter des Kunden zugeordnet. `Credentials`
4. Das SDK ruft den Identity Resolver auf `GetIdentity()` (z. B. `aws.CredentialProvider.Retrieve()` für Sigv4).
5. Das SDK ruft die Endpunkt-Resolver `ResolveEndpoint()` auf, um den Endpunkt für die Anfrage zu finden. Der Endpunkt kann zusätzliche Metadaten enthalten, die den Signaturprozess beeinflussen (z. B. einen eindeutigen Signaturnamen für S3 Object Lambda).
6. Das SDK ruft die `Signer()` API des Authentifizierungsschemas auf, um den Unterzeichner abzurufen, und verwendet seine `SignRequest()` API, um die Anfrage mit der zuvor abgerufenen Anruferidentität zu signieren.

(1) Wenn das SDK auf die anonyme Option (ID`smithy.api#noAuth`) in der Liste stößt, wird sie automatisch ausgewählt, da es keinen entsprechenden Identity Resolver gibt.

Wird nativ unterstützt **`AuthScheme`**

Die folgenden Authentifizierungsschemata werden nativ unterstützt von. AWS SDK für Go

Name	Schema-ID	Identitätsauflöser	Signer	Hinweise
SigV4	`aws.auth#sigv4`	aws. CredentialsProvider	v4. HTTPSigner	Die aktuelle Standardeinstellung für die meisten AWS Serviceoperationen.

Name	Schema-ID	Identitätsauflöser	Signer	Hinweise
SigV4a	`aws.auth#sigv4a`	war. CredentialsProvider	–	Die Verwendung von SigV4a ist derzeit begrenzt, die Signer-Implementierung ist intern. In dieser Ankündigung finden Sie ein neues Opt-In-Modul aws-http-auth, das allgemeine Verwendungszwecke APIs für das Signieren von HTTP-Anfragen aufzeigt.
SIGV4Express	`com.amazonaws.s3#sigv4express`	s3. ExpressCredentials Provider	v4. HTTPSigner	Wird für Express One Zone verwendet.
HTTP-Träger	`smithy.api#httpBearerAuth`	Schmiedeträger. TokenProvider	Smithy Bearer.Signer	Wird von Codecatalyst verwendet.
Anonym	`smithy.api#noAuth`	–	–	Keine Authentifizierung — Es ist keine Identität erforderlich, und die Anfrage ist nicht signiert oder authentifiziert.

Konfiguration der Identität

AWS SDK für Go In werden die Identitätskomponenten eines Authentifizierungsschemas im SDK-Client `Options` konfiguriert. Das SDK nimmt die Werte für diese Komponenten automatisch auf und verwendet sie für das Schema, das es auswählt, wenn eine Operation aufgerufen wird.

> **Note**
>
> Aus Gründen der Abwärtskompatibilität erlaubt das SDK implizit die Verwendung des anonymen Authentifizierungsschemas, wenn keine Identitätsauflösungen konfiguriert sind. Dies kann manuell erreicht werden, indem alle Identitätsauflösungen auf dem Computer eines Clients eingerichtet werden `nil` (der Sigv4-Identitätsauflöser kann auch auf eingestellt werden). `Options` `aws.AnonymousCredentials{}`

Konfiguration des Unterzeichners

In AWS SDK für Go werden die Unterzeichnerkomponenten eines Authentifizierungsschemas im SDK-Client konfiguriert. `Options` Das SDK nimmt die Werte für diese Komponenten automatisch auf und verwendet sie für das Schema, das es auswählt, wenn eine Operation aufgerufen wird. Es ist keine zusätzliche Konfiguration erforderlich.

Benutzerdefiniertes Authentifizierungsschema

Um ein benutzerdefiniertes Authentifizierungsschema zu definieren und es für die Verwendung zu konfigurieren, muss der Aufrufer Folgendes tun:

1. Definieren Sie eine Implementierung AuthScheme
2. Registrieren Sie das Schema in der `AuthSchemes` Liste des SDK-Clients
3. Instrumentieren Sie die SDK-Clients`AuthSchemeResolver`, um gegebenenfalls eine Authentifizierung `Option` mit der ID des Schemas zurückzugeben

> **Warning**
>
> Die folgenden Dienste haben ein einzigartiges oder benutzerdefiniertes Authentifizierungsverhalten. Wir empfehlen Ihnen, an die Standardimplementierung zu delegieren und einen entsprechenden Wrapping vorzunehmen, falls Sie ein benutzerdefiniertes Authentifizierungsverhalten benötigen:

Service	Hinweise
S3	Bedingte Verwendung von SigV4a und SigV4Express, abhängig von der Operation seingabe.
EventBridge	Bedingte Verwendung von SigV4a, abhängig von der Betriebseingabe.
Cognito	Bestimmte Operationen sind nur anonym.
SSO	Bestimmte Operationen sind nur anonym.
STS	Bestimmte Operationen sind nur anonym.

Client-Endpunkte konfigurieren

> **Warning**
>
> Die Auflösung von Endpunkten ist ein SDK-Thema für Fortgeschrittene. Wenn Sie diese Einstellungen ändern, riskieren Sie, dass Ihr Code beschädigt wird. Die Standardeinstellungen sollten für die meisten Benutzer in Produktionsumgebungen gelten.

Das AWS SDK für Go bietet die Möglichkeit, einen benutzerdefinierten Endpunkt zu konfigurieren, der für einen Dienst verwendet werden soll. In den meisten Fällen reicht die Standardkonfiguration aus. Die Konfiguration benutzerdefinierter Endpunkte ermöglicht zusätzliches Verhalten, z. B. die Arbeit mit Vorabversionen eines Dienstes.

Anpassung

Es gibt zwei „Versionen“ der Konfiguration für die Endpunktauflösung im SDK.

- v2, veröffentlicht im dritten Quartal 2023, konfiguriert über:
  - `EndpointResolverV2`
  - `BaseEndpoint`

- v1, zusammen mit dem SDK veröffentlicht, konfiguriert über:
  - `EndpointResolver`

Wir empfehlen Benutzern von Version 1 Endpoint Resolution, auf Version 2 zu migrieren, um Zugriff auf neuere endpunktbezogene Servicefunktionen zu erhalten.

V2: + `EndpointResolverV2BaseEndpoint`

In Auflösung v2 `EndpointResolverV2` ist dies der endgültige Mechanismus, durch den die Endpunktauflösung erfolgt. Die `ResolveEndpoint` Methode des Resolvers wird als Teil des Workflows für jede Anfrage aufgerufen, die Sie im SDK stellen. Der vom Resolver `Endpoint` zurückgegebene Hostname wird unverändert verwendet, wenn die Anfrage gestellt wird (Operations-Serializer können jedoch weiterhin an den HTTP-Pfad angehängt werden).

Auflösung v2 beinhaltet eine zusätzliche Konfiguration auf Client-Ebene, die verwendet wird`BaseEndpoint`, um einen „Basis“ -Hostnamen für die Instanz Ihres Dienstes anzugeben. Der hier festgelegte Wert ist nicht definitiv — er wird letztendlich als Parameter an den Client übergeben, `EndpointResolverV2` wenn die endgültige Auflösung erfolgt (weitere Informationen `EndpointResolverV2` zu den Parametern finden Sie weiter). Die Resolver-Implementierung hat dann die Möglichkeit, diesen Wert zu überprüfen und möglicherweise zu ändern, um den endgültigen Endpunkt zu bestimmen.

Wenn Sie beispielsweise eine `GetObject` S3-Anfrage für einen bestimmten Bucket mit einem Client ausführen, für den Sie einen angegeben haben`BaseEndpoint`, fügt der Standard-Resolver den Bucket in den Hostnamen ein, sofern er mit virtuellen Hosts kompatibel ist (vorausgesetzt, Sie haben das virtuelle Hosting nicht in der Client-Konfiguration deaktiviert).

In der Praxis `BaseEndpoint` wird er höchstwahrscheinlich verwendet, um Ihren Kunden auf eine Entwicklungs- oder Vorschauinstanz eines Dienstes hinzuweisen.

`EndpointResolverV2`-Parameter

Jeder Dienst benötigt einen bestimmten Satz von Eingaben, die an seine Auflösungsfunktion übergeben werden, die in jedem Servicepaket als definiert ist`EndpointParameters`.

Jeder Dienst umfasst die folgenden Basisparameter, die verwendet werden, um die allgemeine Endpunktauflösung innerhalb von Diensten zu erleichtern AWS:

Name	Typ	description
`Region`	`string`	Die AWS Region des Kunden
`Endpoint`	`string`	Der `BaseEndpoint` in der Client-Konfiguration festgelegte Wert
`UseFips`	`bool`	Ob FIPS-Endpunkte in der Client-Konfiguration aktiviert sind
`UseDualStack`	`bool`	Ob Dual-Stack-Endpunkte in der Client-Konfiguration aktiviert sind

Dienste können zusätzliche Parameter angeben, die für die Auflösung erforderlich sind. Beispielsweise `EndpointParameters` enthalten S3-Dateien den Bucket-Namen sowie mehrere S3-spezifische Funktionseinstellungen, z. B. ob die virtuelle Host-Adressierung aktiviert ist.

Wenn Sie Ihre eigene Instanz implementieren`EndpointResolverV2`, sollten Sie niemals Ihre eigene Instanz von erstellen müssen. `EndpointParameters` Das SDK bezieht die Werte pro Anfrage und leitet sie an Ihre Implementierung weiter.

Ein Hinweis zu Amazon S3

Amazon S3 ist ein komplexer Service, bei dem viele seiner Funktionen durch komplexe Endpunktanpassungen wie virtuelles Bucket-Hosting, S3 MRAP und mehr modelliert wurden.

Aus diesem Grund empfehlen wir, die `EndpointResolverV2` Implementierung in Ihrem S3-Client nicht zu ersetzen. Wenn Sie das Auflösungsverhalten erweitern müssen, indem Sie beispielsweise Anfragen an einen lokalen Entwicklungsstapel mit zusätzlichen Überlegungen zu Endpunkten senden, empfehlen wir, die Standardimplementierung so zu verpacken, dass sie als Fallback an den Standard zurückdelegiert (siehe Beispiele unten).

Beispiele

Mit **`BaseEndpoint`**

Der folgende Codeausschnitt zeigt, wie Sie Ihren S3-Client auf eine lokale Instanz eines Dienstes verweisen, der in diesem Beispiel auf dem Loopback-Gerät an Port 8080 gehostet wird.

```
client := s3.NewFromConfig(cfg, func (o *svc.Options) {
    o.BaseEndpoint = aws.String("https://localhost:8080/")
})
```

Mit **`EndpointResolverV2`**

Der folgende Codeausschnitt zeigt, wie Sie mithilfe von benutzerdefiniertem Verhalten in die Endpunktauflösung von S3 einfügen. `EndpointResolverV2`

```
import (
    "context"
    "net/url"

    "github.com/aws/aws-sdk-go-v2/service/s3"
    smithyendpoints "github.com/aws/smithy-go/endpoints"
)

type resolverV2 struct {
    // you could inject additional application context here as well
}

func (*resolverV2) ResolveEndpoint(ctx context.Context, params s3.EndpointParameters) (
        smithyendpoints.Endpoint, error,
    ) {
    if /* input params or caller context indicate we must route somewhere */ {
        u, err := url.Parse("https://custom.service.endpoint/")
        if err != nil {
            return smithyendpoints.Endpoint{}, err
        }
        return smithyendpoints.Endpoint{
            URI: *u,
        }, nil
    }

    // delegate back to the default v2 resolver otherwise
    return s3.NewDefaultEndpointResolverV2().ResolveEndpoint(ctx, params)
```

```
}

func main() {
    // load config...

    client := s3.NewFromConfig(cfg, func (o *s3.Options) {
        o.EndpointResolverV2 = &resolverV2{
            // ...
        }
    })
}
```

Mit beiden

Das folgende Beispielprogramm demonstriert die Interaktion zwischen `BaseEndpoint` und`EndpointResolverV2`. Dies ist ein Anwendungsfall für Fortgeschrittene:

```
import (
    "context"
    "fmt"
    "log"
    "net/url"

    "github.com/aws/aws-sdk-go-v2"
    "github.com/aws/aws-sdk-go-v2/config"
    "github.com/aws/aws-sdk-go-v2/service/s3"
    smithyendpoints "github.com/aws/smithy-go/endpoints"
)

type resolverV2 struct {}

func (*resolverV2) ResolveEndpoint(ctx context.Context, params s3.EndpointParameters) (
        smithyendpoints.Endpoint, error,
    ) {
    // s3.Options.BaseEndpoint is accessible here:
    fmt.Printf("The endpoint provided in config is %s\n", *params.Endpoint)

    // fallback to default
    return s3.NewDefaultEndpointResolverV2().ResolveEndpoint(ctx, params)
}

func main() {
    cfg, err := config.LoadDefaultConfig(context.Background())
```

```
    if (err != nil) {
        log.Fatal(err)
    }

    client := s3.NewFromConfig(cfg, func (o *s3.Options) {
        o.BaseEndpoint = aws.String("https://endpoint.dev/")
        o.EndpointResolverV2 = &resolverV2{}
    })

    // ignore the output, this is just for demonstration
    client.ListBuckets(context.Background(), nil)
}
```

Wenn das obige Programm ausgeführt wird, gibt es Folgendes aus:

```
The endpoint provided in config is https://endpoint.dev/
```

V1: `EndpointResolver`

> **Warning**
>
> Die Endpunktauflösung v1 wird aus Gründen der Abwärtskompatibilität beibehalten und ist vom modernen Verhalten in Endpunktauflösung v2 isoliert. Es wird nur verwendet, wenn das `EndpointResolver` Feld vom Anrufer festgelegt wurde.
> Die Verwendung von Version 1 wird Sie höchstwahrscheinlich daran hindern, auf endpunktbezogene Servicefunktionen zuzugreifen, die mit oder nach der Veröffentlichung der v2-Auflösung eingeführt wurden. Anweisungen zum Upgrade finden Sie unter „Migration“.

A EndpointResolverkann so konfiguriert werden, dass es eine benutzerdefinierte Logik zur Endpunktauflösung für Service-Clients bereitstellt. Sie können einen benutzerdefinierten Endpunktresolver verwenden, um die Endpunktauflösungslogik eines Dienstes für alle Endpunkte oder nur für einen bestimmten regionalen Endpunkt zu überschreiben. Ein benutzerdefinierter Endpunktresolver kann die Endpunktauflösungslogik des Dienstes als Fallback auslösen, falls ein benutzerdefinierter Resolver einen angeforderten Endpunkt nicht auflösen möchte. EndpointResolverWithOptionsFunckann verwendet werden, um Funktionen einfach so zu verpacken, dass sie der Schnittstelle entsprechen. `EndpointResolverWithOptions`

A `EndpointResolver` kann einfach konfiguriert werden, indem der mit einem Wrapped WithEndpointResolverWithOptionsan umschlossene Resolver übergeben wird. Dadurch ist es möglich LoadDefaultConfig, Endpunkte beim Laden von Zugangsdaten zu überschreiben und das Ergebnis `aws.Config` mit Ihrem benutzerdefinierten Endpoint-Resolver zu konfigurieren.

Der Endpunkt-Resolver erhält den Dienst und die Region als Zeichenfolge, sodass der Resolver sein Verhalten dynamisch steuern kann. Jedes Service-Client-Paket hat eine exportierte `ServiceID` Konstante, anhand derer ermittelt werden kann, welcher Service-Client Ihren Endpoint-Resolver aufruft.

Ein Endpoint-Resolver kann den EndpointNotFoundErrorSentinel-Fehlerwert verwenden, um eine Fallback-Auflösung auf die Standardauflösungslogik des Service Clients auszulösen. Auf diese Weise können Sie einen oder mehrere Endpunkte nahtlos überschreiben, ohne sich mit der Fallback-Logik befassen zu müssen.

Wenn Ihre Endpoint Resolver-Implementierung einen anderen Fehler als zurückgibt`EndpointNotFoundError`, wird die Endpunktauflösung beendet und der Dienstvorgang gibt einen Fehler an Ihre Anwendung zurück.

Beispiele

Mit Fallback

Der folgende Codeausschnitt zeigt, wie ein einzelner Dienstendpunkt für DynamoDB mit Fallback-Verhalten für andere Endpunkte überschrieben werden kann:

```
customResolver := aws.EndpointResolverWithOptionsFunc(func(service, region string,
 options ...interface{}) (aws.Endpoint, error) {
    if service == dynamodb.ServiceID && region == "us-west-2" {
        return aws.Endpoint{
            PartitionID:   "aws",
            URL:           "https://test.us-west-2.amazonaws.com",
            SigningRegion: "us-west-2",
        }, nil
    }
    // returning EndpointNotFoundError will allow the service to fallback to it's
 default resolution
    return aws.Endpoint{}, &aws.EndpointNotFoundError{}
})
```

```
cfg, err := config.LoadDefaultConfig(context.TODO(),
 config.WithEndpointResolverWithOptions(customResolver))
```

Ohne Fallback

Der folgende Codeausschnitt zeigt, wie ein einzelner Dienstendpunkt für DynamoDB ohne Fallback-Verhalten für andere Endpunkte außer Kraft gesetzt werden kann:

```
customResolver := aws.EndpointResolverWithOptionsFunc(func(service, region string,
 options ...interface{}) (aws.Endpoint, error) {
    if service == dynamodb.ServiceID && region == "us-west-2" {
        return aws.Endpoint{
            PartitionID:   "aws",
            URL:           "https://test.us-west-2.amazonaws.com",
            SigningRegion: "us-west-2",
        }, nil
    }
    return aws.Endpoint{}, fmt.Errorf("unknown endpoint requested")
})

cfg, err := config.LoadDefaultConfig(context.TODO(),
 config.WithEndpointResolverWithOptions(customResolver))
```

Unveränderliche Endpunkte

> **Warning**
>
> Das Festlegen eines Endpunkts als unveränderlich kann dazu führen, dass einige Service-Client-Funktionen nicht ordnungsgemäß funktionieren, was zu undefiniertem Verhalten führen kann. Bei der Definition eines Endpunkts als unveränderlich ist Vorsicht geboten.

Einige Service-Clients, wie Amazon S3, können den vom Resolver zurückgegebenen Endpunkt für bestimmte Servicevorgänge ändern. Amazon S3 verarbeitet beispielsweise die virtuelle Bucket-Adressierung automatisch, indem der aufgelöste Endpunkt mutiert wird. Sie können verhindern, dass das SDK Ihre benutzerdefinierten Endpunkte mutiert, indem Sie auf einstellen. HostnameImmutable`true` Zum Beispiel:

```
customResolver := aws.EndpointResolverWithOptionsFunc(func(service, region string,
 options ...interface{}) (aws.Endpoint, error) {
```

```
    if service == dynamodb.ServiceID && region == "us-west-2" {
        return aws.Endpoint{
            PartitionID:   "aws",
            URL:           "https://test.us-west-2.amazonaws.com",
            SigningRegion: "us-west-2",
            HostnameImmutable: true,
        }, nil
    }
    return aws.Endpoint{}, fmt.Errorf("unknown endpoint requested")
})

cfg, err := config.LoadDefaultConfig(context.TODO(),
 config.WithEndpointResolverWithOptions(customResolver))
```

Migration

Bei der Migration von Version 1 zu Version 2 der Endpunktauflösung gelten die folgenden allgemeinen Prinzipien:

- Die Rückgabe eines Endpunkts mit der HostnameImmutableEinstellung auf `false` entspricht in etwa der Einstellung `BaseEndpoint` auf die ursprünglich von Version 1 zurückgegebene URL und der Beibehaltung `EndpointResolverV2` der Standardeinstellung.
- Die Rückgabe eines Endpunkts mit HostnameImmutable set to `true` entspricht in etwa der Implementierung eines`EndpointResolverV2`, das die ursprünglich zurückgegebene URL von v1 zurückgibt.
  - Die primäre Ausnahme bilden Operationen mit modellierten Endpunktpräfixen. Ein Hinweis dazu finden Sie weiter unten.

Beispiele für diese Fälle sind unten aufgeführt.

> **Warning**
>
> Unveränderliche V1-Endpunkte und V2-Auflösung sind im Verhalten nicht gleichwertig. Beispielsweise würden Signaturüberschreibungen für benutzerdefinierte Funktionen wie S3 Object Lambda immer noch für unveränderliche Endpunkte festgelegt, die über v1-Code zurückgegeben werden, aber dasselbe gilt nicht für v2.

Hinweis zu Host-Präfixen

Einige Operationen werden mit Host-Präfixen modelliert, die dem aufgelösten Endpunkt vorangestellt werden. Dieses Verhalten muss zusammen mit der Ausgabe von ResolveEndpoint V2 funktionieren, weshalb das Hostpräfix weiterhin auf dieses Ergebnis angewendet wird.

Sie können das Endpunkt-Host-Präfix manuell deaktivieren, indem Sie eine Middleware anwenden. Weitere Informationen finden Sie im Abschnitt mit den Beispielen.

Beispiele

Veränderbarer Endpunkt

Das folgende Codebeispiel zeigt, wie ein einfacher v1-Endpoint-Resolver migriert wird, der einen modifizierbaren Endpunkt zurückgibt:

```
// v1
client := svc.NewFromConfig(cfg, func (o *svc.Options) {
    o.EndpointResolver = svc.EndpointResolverFromURL("https://custom.endpoint.api/")
})

// v2
client := svc.NewFromConfig(cfg, func (o *svc.Options) {
    // the value of BaseEndpoint is passed to the default EndpointResolverV2
    // implementation, which will handle routing for features such as S3 accelerate,
    // MRAP, etc.
    o.BaseEndpoint = aws.String("https://custom.endpoint.api/")
})
```

Unveränderlicher Endpunkt

```
// v1
client := svc.NewFromConfig(cfg, func (o *svc.Options) {
    o.EndpointResolver = svc.EndpointResolverFromURL("https://custom.endpoint.api/",
 func (e *aws.Endpoint) {
        e.HostnameImmutable = true
    })
})

// v2
import (
    smithyendpoints "github.com/aws/smithy-go/endpoints"
```

```
)

type staticResolver struct {}

func (*staticResolver) ResolveEndpoint(ctx context.Context, params
 svc.EndpointParameters) (
        smithyendpoints.Endpoint, error,
    ) {
    // This value will be used as-is when making the request.
    u, err := url.Parse("https://custom.endpoint.api/")
    if err != nil {
        return smithyendpoints.Endpoint{}, err
    }
    return smithyendpoints.Endpoint{
        URI: *u,
    }, nil
}

client := svc.NewFromConfig(cfg, func (o *svc.Options) {
    o.EndpointResolverV2 = &staticResolver{}
})
```

Hostpräfix deaktivieren

```
import (
    "context"
    "fmt"
    "net/url"

    "github.com/aws/aws-sdk-go-v2/aws"
    "github.com/aws/aws-sdk-go-v2/config"
    "github.com/aws/aws-sdk-go-v2/service/<service>"
    smithyendpoints "github.com/aws/smithy-go/endpoints"
    "github.com/aws/smithy-go/middleware"
    smithyhttp "github.com/aws/smithy-go/transport/http"
)

// disableEndpointPrefix applies the flag that will prevent any
// operation-specific host prefix from being applied
type disableEndpointPrefix struct{}

func (disableEndpointPrefix) ID() string { return "disableEndpointPrefix" }
```

```
func (disableEndpointPrefix) HandleInitialize(
    ctx context.Context, in middleware.InitializeInput, next
 middleware.InitializeHandler,
) (middleware.InitializeOutput, middleware.Metadata, error) {
    ctx = smithyhttp.SetHostnameImmutable(ctx, true)
    return next.HandleInitialize(ctx, in)
}

func addDisableEndpointPrefix(o *<service>.Options) {
    o.APIOptions = append(o.APIOptions, (func(stack *middleware.Stack) error {
        return stack.Initialize.Add(disableEndpointPrefix{}, middleware.After)
    }))
}

type staticResolver struct{}

func (staticResolver) ResolveEndpoint(ctx context.Context, params
 <service>.EndpointParameters) (
    smithyendpoints.Endpoint, error,
) {
    u, err := url.Parse("https://custom.endpoint.api/")
    if err != nil {
        return smithyendpoints.Endpoint{}, err
    }

    return smithyendpoints.Endpoint{URI: *u}, nil
}


func main() {
    cfg, err := config.LoadDefaultConfig(context.Background())
    if err != nil {
        panic(err)
    }

    svc := <service>.NewFromConfig(cfg, func(o *<service>.Options) {
        o.EndpointResolverV2 = staticResolver{}
    })

    _, err = svc.<Operation>(context.Background(), &<service>.<OperationInput>{ /* ...
 */ },
        addDisableEndpointPrefix)
    if err != nil {
        panic(err)
```

```
    }
}
```

Passen Sie den HTTP-Client an

Der AWS SDK für Go verwendet einen Standard-HTTP-Client mit Standardkonfigurationswerten. Sie können zwar einige dieser Konfigurationswerte ändern, aber der Standard-HTTP-Client und -Transport sind nicht ausreichend für Kunden konfiguriert, die den AWS SDK für Go in einer Umgebung mit hohem Durchsatz und niedrigen Latenzanforderungen verwenden. Weitere Informationen finden Sie im, Häufig gestellte Fragen da die Konfigurationsempfehlungen je nach Arbeitslast variieren. In diesem Abschnitt wird beschrieben, wie Sie einen benutzerdefinierten HTTP-Client konfigurieren und diesen Client zum Erstellen von AWS SDK für Go Aufrufen verwenden.

Um Ihnen bei der Erstellung eines benutzerdefinierten HTTP-Clients zu helfen, wird in diesem Abschnitt beschrieben, wie Sie benutzerdefinierte Einstellungen konfigurieren und diesen Client mit einem AWS SDK für Go Service-Client verwenden. NewBuildableClient

Lassen Sie uns definieren, was wir anpassen möchten.

Überschreiben beim Laden der Konfiguration

Beim Aufrufen können benutzerdefinierte HTTP-Clients bereitgestellt werden, LoadDefaultConfigindem der Client mit With umschlossen HTTPClient und der resultierende Wert an `LoadDefaultConfig` übergeben wird. Um zum Beispiel `customClient` als unseren Client zu übergeben:

```
cfg, err := config.LoadDefaultConfig(context.TODO(),
 config.WithHTTPClient(customClient))
```

Zeitüberschreitung

Das `BuildableHTTPClient` kann mit einem Timeout-Limit für Anfragen konfiguriert werden. Dieses Timeout beinhaltet die Zeit, um eine Verbindung herzustellen, etwaige Weiterleitungen zu verarbeiten und den gesamten Antworttext zu lesen. Um beispielsweise das Client-Timeout zu ändern:

```
import "github.com/aws/aws-sdk-go-v2/aws/transport/http"

// ...
```

```
httpClient := http.NewBuildableClient().WithTimeout(time.Second*5)
```

Dialer

Der `BuildableHTTPClient` bietet eine Builder-Mechanik zum Erstellen von Clients mit modifizierten Dialer-Optionen. Das folgende Beispiel zeigt, wie die Einstellungen eines Clients `Dialer` konfiguriert werden.

```
import awshttp "github.com/aws/aws-sdk-go-v2/aws/transport/http"
import "net"

// ...

httpClient := awshttp.NewBuildableClient().WithDialerOptions(func(d *net.Dialer) {
    d.KeepAlive = -1
    d.Timeout = time.Millisecond*500
})
```

Einstellungen

Dialer. KeepAlive

Diese Einstellung stellt den Keep-Alive-Zeitraum für eine aktive Netzwerkverbindung dar.

Auf einen negativen Wert setzen, um Keep-Alives zu deaktivieren.

Auf 0 setzen, um Keep-Alives zu aktivieren, sofern sie vom Protokoll und vom Betriebssystem unterstützt werden.

Netzwerkprotokolle oder Betriebssysteme, die Keep-Alives nicht unterstützen, ignorieren dieses Feld. Standardmäßig aktiviert TCP Keep Alive.

Weitere Informationen finden Sie unter https://golang. org/pkg/net/#Dialer. KeepAlive

`KeepAlive`Als Time.Duration festlegen.

Dialer.Timeout

Diese Einstellung gibt an, wie lange ein Dial maximal darauf wartet, dass eine Verbindung hergestellt wird.

Die Standardeinstellung ist 30 Sekunden.

Weitere Informationen finden Sie unter https://golang. org/pkg/net/#Dialer .Timeout

`Timeout`Als Time.Duration festlegen.

Transport

Das `BuildableHTTPClient` bietet eine Builder-Mechanik zum Erstellen von Clients mit modifizierten Transportoptionen.

Einen Proxy konfigurieren

Wenn Sie keine direkte Verbindung zum Internet herstellen können, können Sie von GO unterstützte Umgebungsvariablen (HTTP_PROXY/HTTPS_PROXY) verwenden oder einen benutzerdefinierten HTTP-Client erstellen, um Ihren Proxy zu konfigurieren. Im folgenden Beispiel wird der Client so konfiguriert, dass er PROXY_URL als Proxy-Endpunkt verwendet wird:

```
import awshttp "github.com/aws/aws-sdk-go-v2/aws/transport/http"
import "net/http"

// ...

httpClient := awshttp.NewBuildableClient().WithTransportOptions(func(tr
 *http.Transport) {
    proxyURL, err := url.Parse("PROXY_URL")
    if err != nil {
        log.Fatal(err)
    }
    tr.Proxy = http.ProxyURL(proxyURL)
})
```

Andere Einstellungen

Im Folgenden finden Sie einige andere `Transport` Einstellungen, die geändert werden können, um den HTTP-Client zu optimieren. Alle zusätzlichen Einstellungen, die hier nicht beschrieben werden, finden Sie in der Dokumentation zum Transporttyp. Diese Einstellungen können wie im folgenden Beispiel gezeigt angewendet werden:

```
import awshttp "github.com/aws/aws-sdk-go-v2/aws/transport/http"
import "net/http"

// ...
```

```
httpClient := awshttp.NewBuildableClient().WithTransportOptions(func(tr
 *http.Transport) {
    tr.ExpectContinueTimeout = 0
    tr.MaxIdleConns = 10
})
```

Verkehr. ExpectContinueTimeout

Wenn die Anfrage einen „Expect: 100-continue“ -Header hat, stellt diese Einstellung die maximale Wartezeit auf die ersten Antwortheader eines Servers dar, nachdem die Anforderungsheader vollständig geschrieben wurden. Diese Zeit beinhaltet nicht die Zeit zum Senden des Anforderungsheaders. Der HTTP-Client sendet seine Nutzdaten, nachdem dieser Timeout abgelaufen ist.

Standard 1 Sekunde.

Setzen Sie den Wert auf 0, um kein Timeout zu vermeiden, und senden Sie die Payload der Anfrage ohne Wartezeit. Ein Anwendungsfall ist, wenn Sie auf Probleme mit Proxys oder Drittanbieterdiensten stoßen, die eine Sitzung durchführen, die der Verwendung von Amazon S3 in der später gezeigten Funktion ähnelt.

Weitere Informationen finden Sie unter https://golang. org/pkg/net/http/#Transport. ExpectContinueTimeout

`ExpectContinue`Als Time.Duration festlegen.

Verkehr. IdleConnTimeout

Diese Einstellung stellt die maximale Zeit dar, um eine inaktive Netzwerkverbindung zwischen HTTP-Anfragen aufrechtzuerhalten.

Auf 0 gesetzt, wenn kein Limit angegeben wird.

Weitere Informationen finden Sie unter https://golang. org/pkg/net/http/#Transport. IdleConnTimeout

`IdleConnTimeout`Als Time.Duration festlegen.

Verkehr. MaxIdleConns

Diese Einstellung stellt die maximale Anzahl inaktiver Verbindungen (Keep-Alive) auf allen Hosts dar. Ein Anwendungsfall für die Erhöhung dieses Werts ist, wenn Sie in kurzer Zeit viele Verbindungen von denselben Clients sehen

0 bedeutet kein Limit.

Weitere Informationen finden Sie unter https://golang. org/pkg/net/http/#Transport. MaxIdleConns

`MaxIdleConns`Als int setzen.

Verkehr. MaxIdleConnsPerHost

Diese Einstellung stellt die maximale Anzahl inaktiver Verbindungen (Keep-Alive) dar, die pro Host beibehalten werden können. Ein Anwendungsfall für die Erhöhung dieses Werts ist, wenn Sie innerhalb eines kurzen Zeitraums viele Verbindungen von denselben Clients sehen

Die Standardeinstellung sind zwei inaktive Verbindungen pro Host.

Auf 0 setzen, um DefaultMaxIdleConnsPerHost (2) zu verwenden.

Weitere Informationen finden Sie unter https://golang. org/pkg/net/http/#Transport. MaxIdleConnsPerHost

`MaxIdleConnsPerHost`Als int setzen.

Verkehr. ResponseHeaderTimeout

Diese Einstellung stellt die maximale Wartezeit dar, bis ein Client den Antwortheader gelesen hat.

Wenn der Client den Header der Antwort innerhalb dieser Dauer nicht lesen kann, schlägt die Anforderung mit einem Timeout-Fehler fehl.

Seien Sie vorsichtig bei der Einstellung dieses Werts, wenn Sie Lambda-Funktionen mit langer Laufzeit verwenden, da der Vorgang keine Antwortheader zurückgibt, bis die Lambda-Funktion abgeschlossen ist oder das Timeout abgelaufen ist. Sie können diese Option jedoch weiterhin mit der API-Operation ** ** verwenden. InvokeAsync

Die Standardeinstellung ist kein Timeout. Warte ewig.

Weitere Informationen finden Sie unter https://golang. org/pkg/net/http/#Transport. ResponseHeaderTimeout

`ResponseHeaderTimeout`Als Time.Duration festlegen.

Verkehr. TLSHandshakeAuszeit

Diese Einstellung stellt die maximale Wartezeit bis zum Abschluss eines TLS-Handshakes dar.

Der Standardwert beträgt 10 Sekunden.

Null bedeutet kein Timeout.

Weitere Informationen finden Sie unter https://golang. org/pkg/net/http/#Transport. TLSHandshakeAuszeit

`TLSHandshakeTimeout`Als Time.Duration festlegen.

Protokollierung

Der AWS SDK für Go verfügt über Protokollierungsfunktionen, mit denen Ihre Anwendung Debugging-Informationen zum Debuggen und Diagnostizieren von Anforderungsproblemen oder -fehlern aktivieren kann. Die Logger-Schnittstelle und ClientLogModesind die wichtigsten Komponenten, mit denen Sie festlegen können, wie und was von Clients protokolliert werden soll.

Logger

Beim Erstellen einer Config `Logger` wird standardmäßig so konfiguriert, dass Protokollnachrichten an den Standardfehler des Prozesses (stderr) gesendet werden. LoadDefaultConfig Ein benutzerdefinierter Logger, der die Logger-Schnittstelle erfüllt, kann als Argument übergeben werden, `LoadDefaultConfig` indem er mit der Konfiguration umschlossen wird. WithLogger.

Zum Beispiel, um unsere Kunden so zu konfigurieren, dass sie unsere verwenden`applicationLogger`:

```
cfg, err := config.LoadDefaultConfig(context.TODO(),
 config.WithLogger(applicationLogger))
```

Jetzt senden Clients, die mit dem `aws.Config` Constructed konfiguriert wurden, Protokollnachrichten an`applicationLogger`.

Kontextsensitive Logger

Eine Logger-Implementierung kann die optionale Schnittstelle implementieren. ContextLogger Bei Loggern, die diese Schnittstelle implementieren, werden ihre `WithContext` Methoden im aktuellen Kontext aufgerufen. Auf diese Weise können Ihre Protokollierungsimplementierungen eine neue Datei zurückgeben`Logger`, die zusätzliche Logging-Metadaten auf der Grundlage der im Kontext vorhandenen Werte schreiben kann.

ClientLogMode

Standardmäßig erzeugen Service-Clients keine Protokollnachrichten. Um Clients so zu konfigurieren, dass sie Protokollnachrichten zu Debugging-Zwecken senden, verwenden Sie das ClientLogModeMitglied on`Config`. `ClientLogMode`kann so eingestellt werden, dass Debugging-Messaging aktiviert wird für:

- Signatur Version 4 (Sigv4) Signierung
- Wiederholungen anfordern
- HTTP-Anforderungen
- HTTP-Antworten

Um beispielsweise die Protokollierung von HTTP-Anfragen und -Wiederholungen zu aktivieren:

```
cfg, err := config.LoadDefaultConfig(context.TODO(),
 config.WithClientLogMode(aws.LogRetries | aws.LogRequest))
```

Informationen zu ClientLogModeden verschiedenen verfügbaren Client-Protokollmodi finden Sie unter.

Wiederholversuche und Zeitüberschreitungen

AWS SDK für Go Damit können Sie das Wiederholungsverhalten von Anfragen an HTTP-Dienste konfigurieren. Standardmäßig verwenden Service-Clients Retry.STANDARD als Standard-Wiederholungsversuch. Wenn die Standardkonfiguration oder das Standardverhalten nicht Ihren Anwendungsanforderungen entspricht, können Sie die Retryer-Konfiguration anpassen oder Ihre eigene Retryer-Implementierung bereitstellen.

Das AWS SDK für Go stellt eine AWS.Retryer-Schnittstelle bereit, die den Methodensatz definiert, der für die Implementierung einer Retry-Implementierung erforderlich ist. Das SDK bietet zwei Implementierungen für Wiederholungen: retry.STANDARD und aws. NoOpRetryer.

Standardtrockner

Der Retry.Standard Retryer ist die Standardimplementierung, die von SDK-Clients verwendet wird. `aws.Retryer` Der Standard-Retryer ist ein Retryer mit begrenzter Geschwindigkeit, einer konfigurierbaren Anzahl von Maximalversuchen und der Möglichkeit, die Rücknahme von Anfragen zu deaktivieren.

In der folgenden Tabelle sind die Standardwerte für diesen Wiederholungsversuch definiert:

Eigenschaft	Standard
Max. Anzahl der Versuche	3
Maximale Back-Off-Verzögerung	20 Sekunden

Wenn beim Aufrufen Ihrer Anfrage ein Fehler auftritt, der wiederholt werden kann, verwendet der Standard-Wiederholungsversuch die bereitgestellte Konfiguration, um die Anfrage zu verzögern und anschließend erneut zu versuchen. Wiederholungen erhöhen die Gesamtlatenz Ihrer Anfrage, und Sie müssen den Wiederholungsversuch konfigurieren, falls die Standardkonfiguration nicht Ihren Anwendungsanforderungen entspricht.

In der Dokumentation zum Retry-Paket finden Sie Einzelheiten dazu, welche Fehler bei der Standardimplementierung von Retryern als wiederholbar angesehen werden.

NopRetryer

Das war. NopRetryerist eine `aws.Retryer` Implementierung, die bereitgestellt wird, wenn Sie alle Wiederholungsversuche deaktivieren möchten. Beim Aufrufen eines Service-Client-Vorgangs erlaubt dieser Wiederholungsversuch nur, dass die Anfrage einmal versucht wird, und alle daraus resultierenden Fehler werden an die aufrufende Anwendung zurückgegeben.

Verhalten anpassen

Das SDK stellt eine Reihe von Hilfsprogrammen bereit, die eine `aws.Retryer` Implementierung umschließen, und gibt den bereitgestellten Wiederholungsversuch mit dem gewünschten Wiederholungsverhalten zurück. Sie können den Standard-Retryer für alle Clients, pro Client oder pro Vorgang überschreiben, je nach den Anforderungen Ihrer Anwendung. Weitere Beispiele zur Vorgehensweise finden Sie in den Dokumentationsbeispielen zum Retry-Paket.

> **Warning**
>
> Wenn Sie eine globale `aws.Retryer` Implementierung mit angeben`config.WithRetryer`, müssen Sie sicherstellen, dass Sie für `aws.Retryer` jeden Aufruf eine neue Instanz zurückgeben. Dadurch wird sichergestellt, dass Sie nicht für alle Service-Clients einen globalen Token-Bucket für Wiederholungsversuche erstellen.

Begrenzung der maximalen Anzahl von Versuchen

Sie verwenden Wiederholungsversuch. AddWithMaxAttemptsum eine `aws.Retryer` Implementierung zu beenden und die maximale Anzahl der Versuche auf den gewünschten Wert zu setzen. Wenn die maximale Anzahl der Versuche auf Null gesetzt wird, kann das SDK alle wiederholbaren Fehler wiederholen, bis die Anfrage erfolgreich ist oder ein Fehler zurückgegeben wird, der nicht wiederholt werden kann.

Sie können beispielsweise den folgenden Code verwenden, um den standardmäßigen Client-Wiederholungsversuch mit maximal fünf Versuchen zu umschließen:

```
import "context"
import "github.com/aws/aws-sdk-go-v2/aws/retry"
import "github.com/aws/aws-sdk-go-v2/config"
import "github.com/aws/aws-sdk-go-v2/service/s3"

// ...

cfg, err := config.LoadDefaultConfig(context.TODO(), config.WithRetryer(func()
 aws.Retryer {
    return retry.AddWithMaxAttempts(retry.NewStandard(), 5)
}))
if err != nil {
    return err
}

client := s3.NewFromConfig(cfg)
```

Begrenzung der maximalen Back-Off-Verzögerung

Sie verwenden Retry. AddWithMaxBackoffDelayum eine `aws.Retryer` Implementierung abzuschließen und die maximale Back-Off-Verzögerung zu begrenzen, die zwischen erneuten Versuchen einer fehlgeschlagenen Anfrage auftreten kann.

Sie können beispielsweise den folgenden Code verwenden, um den standardmäßigen Client-Wiederholungsversuch mit einer gewünschten maximalen Verzögerung von fünf Sekunden zu beenden:

```
import "context"
import "time"
```

```
import "github.com/aws/aws-sdk-go-v2/aws/retry"
import "github.com/aws/aws-sdk-go-v2/config"
import "github.com/aws/aws-sdk-go-v2/service/s3"

// ...

cfg, err := config.LoadDefaultConfig(context.TODO(), config.WithRetryer(func()
 aws.Retryer {
    return retry.AddWithMaxBackoffDelay(retry.NewStandard(), time.Second*5)
}))
if err != nil {
    return err
}

client := s3.NewFromConfig(cfg)
```

Versuchen Sie es erneut mit weiteren API-Fehlercodes

Sie verwenden Retry. AddWithErrorCodesum eine `aws.Retryer` Implementierung zu verpacken und zusätzliche API-Fehlercodes einzufügen, die als wiederholbar erachtet werden sollten.

Sie können beispielsweise den folgenden Code verwenden, um den Standard-Client-Wiederholungsversuch so zu umschließen, dass die Amazon S3 `NoSuchBucketException` S3-Ausnahme als wiederholbar gilt.

```
import "context"
import "time"
import "github.com/aws/aws-sdk-go-v2/aws/retry"
import "github.com/aws/aws-sdk-go-v2/config"
import "github.com/aws/aws-sdk-go-v2/service/s3"
import "github.com/aws/aws-sdk-go-v2/service/s3/types"

// ...

cfg, err := config.LoadDefaultConfig(context.TODO(), config.WithRetryer(func()
 aws.Retryer {
    return retry.AddWithErrorCodes(retry.NewStandard(), (*types.NoSuchBucketException)
(nil).ErrorCode())
}))
if err != nil {
    return err
}
```

```
client := s3.NewFromConfig(cfg)
```

Clientseitige Ratenbegrenzung

Damit AWS SDK für Go wird ein neuer Mechanismus zur kundenseitigen Ratenbegrenzung in die Standardrichtlinie für Wiederholungsversuche aufgenommen, der dem heutigen Verhalten entspricht. SDKs Dieses Verhalten wird durch das RateLimiterFeld in den Optionen eines Wiederholungsversuchs gesteuert.

A RateLimiter arbeitet als Token-Bucket mit einer festgelegten Kapazität, in dem fehlgeschlagene Betriebsversuche Tokens verbrauchen. Ein erneuter Versuch, mehr Token zu verbrauchen, als verfügbar sind, führt zu einem Betriebsfehler mit einer QuotaExceededError.

Die Standardimplementierung ist wie folgt parametrisiert (wie man die einzelnen Einstellungen ändert):

- eine Kapazität von 500 (stellen Sie den Wert RateLimiter on StandardOptions using ein) NewTokenRateLimit
- Ein durch einen Timeout verursachter Wiederholungsversuch kostet 10 Token (aktiviert) RetryTimeoutCost StandardOptions
- ein durch andere Fehler verursachter Wiederholungsversuch kostet 5 Token (aktiviert) RetryCost StandardOptions
- Ein Vorgang, der beim ersten Versuch erfolgreich ist, fügt 1 Token hinzu (aktiviert) NoRetryIncrement StandardOptions
  - Operationen, die beim zweiten oder späteren Versuch erfolgreich sind, fügen keine Token zurück

Wenn Sie feststellen, dass das Standardverhalten nicht den Anforderungen Ihrer Anwendung entspricht, können Sie es mit RateLimit.NONE deaktivieren.

Beispiel: modifizierter Ratenbegrenzer

```
import (
    "context"

    "github.com/aws/aws-sdk-go-v2/aws"
    "github.com/aws/aws-sdk-go-v2/aws/ratelimit"
    "github.com/aws/aws-sdk-go-v2/aws/retry"
    "github.com/aws/aws-sdk-go-v2/config"
)
```

```
// ...

cfg, err := config.LoadDefaultConfig(context.Background(), config.WithRetryer(func()
 aws.Retryer {
    return retry.NewStandard(func(o *retry.StandardOptions) {
        // Makes the rate limiter more permissive in general. These values are
        // arbitrary for demonstration and may not suit your specific
        // application's needs.
        o.RateLimiter = ratelimit.NewTokenRateLimit(1000)
        o.RetryCost = 1
        o.RetryTimeoutCost = 3
        o.NoRetryIncrement = 10
    })
}))
```

Beispiel: kein Ratenlimit mit RateLimit.None

```
import (
    "context"

    "github.com/aws/aws-sdk-go-v2/aws"
    "github.com/aws/aws-sdk-go-v2/aws/ratelimit"
    "github.com/aws/aws-sdk-go-v2/aws/retry"
    "github.com/aws/aws-sdk-go-v2/config"
)

// ...

cfg, err := config.LoadDefaultConfig(context.Background(), config.WithRetryer(func()
 aws.Retryer {
    return retry.NewStandard(func(o *retry.StandardOptions) {
        o.RateLimiter = ratelimit.None
    })
}))
```

Timeouts

Sie verwenden das Kontextpaket, um beim Aufrufen eines Service-Client-Vorgangs Timeouts oder Deadlines festzulegen. Verwenden Sie den Kontext. WithDeadlineum den Kontext Ihrer Anwendungen zu verschließen und eine Frist bis zu einem bestimmten Zeitpunkt festzulegen, bis zu dem der aufgerufene Vorgang abgeschlossen sein muss. Um ein Timeout nach einem

bestimmten `time.Duration` Nutzungskontext festzulegen. WithTimeout. Das SDK übergibt die bereitgestellten Daten `context.Context` an den HTTP-Transportclient, wenn eine Service-API aufgerufen wird. Wenn der an das SDK übergebene Kontext beim Aufrufen des Vorgangs storniert wird oder abgebrochen wird, wiederholt das SDK die Anfrage nicht weiter und kehrt zur aufrufenden Anwendung zurück. In Fällen, in denen der dem SDK bereitgestellte Kontext storniert wurde, müssen Sie die Kontextstornierung in Ihrer Anwendung entsprechend handhaben.

Ein Timeout festlegen

Das folgende Beispiel zeigt, wie ein Timeout für einen Service-Client-Vorgang festgelegt wird.

```
import "context"
import "time"

// ...

ctx := context.TODO() // or appropriate context.Context value for your application

client := s3.NewFromConfig(cfg)

// create a new context from the previous ctx with a timeout, e.g. 5 seconds
ctx, cancel := context.WithTimeout(ctx, 5*time.Second)
defer cancel()

resp, err := client.GetObject(ctx, &s3.GetObjectInput{
    // input parameters
})
if err != nil {
    // handle error
}
```

Mit dem AWS SDK für Go

Lernen Sie gängige und empfohlene Methoden zur Programmierung mit dem AWS SDK für Go in Ihren Anwendungen kennen.

Themen

- Einen Service-Client erstellen
- Service Operations aufrufen
- Gleichzeitige Verwendung von Service-Clients
- Verwendung von Operation Paginators
- Verwenden von Waiter-Objekten
- Umgang mit Fehlern in der AWS SDK für Go V2

Einen Service-Client erstellen

Service-Clients können entweder mithilfe der `NewFromConfig` Funktionen `New` oder erstellt werden, die im Go-Paket des Service-Clients verfügbar sind. Jede Funktion gibt einen `Client` Strukturtyp zurück, der die Methoden zum Aufrufen des Dienstes APIs enthält. `NewFromConfig`Beide bieten dieselben konfigurierbaren Optionen für die Erstellung eines Service-Clients, bieten jedoch leicht unterschiedliche Konstruktionsmuster, die wir in den folgenden Abschnitten betrachten werden. `New`

NewFromConfig

`NewFromConfig`Die Funktion bietet eine konsistente Schnittstelle für die Erstellung von Service-Clients mithilfe von AWS.config. Eine `aws.Config` kann mit der Konfiguration geladen werden. LoadDefaultConfig. Weitere Informationen zum Konstruieren eines finden Sie `aws.Config` unterDas SDKs konfigurieren. Das folgende Beispiel zeigt, wie ein Amazon S3 S3-Serviceclient mit der `NewFromConfig` Funktion `aws.Config` und erstellt wird:

```
import "context"
    import "github.com/aws/aws-sdk-go-v2/config"
    import "github.com/aws/aws-sdk-go-v2/service/s3"

    // ...

    cfg, err := config.LoadDefaultConfig(context.TODO())
```

```
    if err != nil {
        panic(err)
    }

    client := s3.NewFromConfig(cfg)
```

Konfiguration überschreiben

`NewFromConfig`kann ein oder mehrere funktionale Argumente annehmen, die die `Options` Konfigurationsstruktur eines Clients verändern können. Auf diese Weise können Sie bestimmte Überschreibungen vornehmen, z. B. die Region ändern oder dienstspezifische Optionen wie die Amazon S3 `UseAccelerate` S3-Option ändern. Zum Beispiel:

```
import "context"
    import "github.com/aws/aws-sdk-go-v2/config"
    import "github.com/aws/aws-sdk-go-v2/service/s3"

    // ...

    cfg, err := config.LoadDefaultConfig(context.TODO())
    if err != nil {
        panic(err)
    }

    client := s3.NewFromConfig(cfg, func(o *s3.Options) {
        o.Region = "us-west-2"
        o.UseAccelerate = true
    })
```

Das Überschreiben des `Options` Client-Werts hängt von der Reihenfolge ab, in der die Funktionsargumente angegeben werden. `NewFromConfig`

Neu

> **Note**
>
> `New`wird als eine fortgeschrittenere Form der Client-Konstruktion angesehen. Wir empfehlen die Verwendung `NewFromConfig` für die Konstruktion auf Kundenbasis, da es die Konstruktion unter Verwendung der `aws.Config` Struktur ermöglicht. Dadurch entfällt

die Notwendigkeit, für jeden Service-Client, den Ihre Anwendung benötigt, eine `Options` Strukturinstanz zu erstellen.

`New`Die Funktion ist ein Client-Konstruktor, der eine Schnittstelle zum Konstruieren von Clients bereitstellt, wobei nur die `Options` Struktur des Client-Pakets verwendet wird, um die Konfigurationsoptionen des Clients zu definieren. Um beispielsweise einen Amazon S3 S3-Client zu erstellen mit`New`:

```
import "github.com/aws/aws-sdk-go-v2/aws"
    import "github.com/aws/aws-sdk-go-v2/credentials"
    import "github.com/aws/aws-sdk-go-v2/service/s3"

    // ...

    client := s3.New(s3.Options{
        Region:      "us-west-2",
        Credentials:
 aws.NewCredentialsCache(credentials.NewStaticCredentialsProvider(accessKey, secretKey,
 "")),
    })
```

Konfiguration überschreiben

`New`kann ein oder mehrere funktionale Argumente annehmen, die die `Options` Konfigurationsstruktur eines Clients verändern können. Auf diese Weise können Sie bestimmte Überschreibungen vornehmen, z. B. die Region ändern oder dienstspezifische Optionen wie die Amazon S3 `UseAccelerate` S3-Option ändern. Zum Beispiel:

```
import "github.com/aws/aws-sdk-go-v2/aws"
    import "github.com/aws/aws-sdk-go-v2/credentials"
    import "github.com/aws/aws-sdk-go-v2/service/s3"

    // ...

    options := s3.Options{
        Region:      "us-west-2",
        Credentials:
 aws.NewCredentialsCache(credentials.NewStaticCredentialsProvider(accessKey, secretKey,
 "")),
```

```
    }

    client := s3.New(options, func(o *s3.Options) {
        o.Region = "us-east-1"
        o.UseAccelerate = true
    })
```

Das Überschreiben des `Options` Client-Werts hängt von der Reihenfolge ab, in der die Funktionsargumente angegeben werden. `New`

Service Operations aufrufen

Nachdem Sie eine Service-Client-Instanz eingerichtet haben, können Sie sie verwenden, um die Operationen eines Dienstes aufzurufen. Um beispielsweise den Amazon S3 `GetObject` S3-Vorgang aufzurufen:

```
response, err := client.GetObject(context.TODO(), &s3.GetObjectInput{
        Bucket: aws.String("amzn-s3-demo-bucket"),
        Key:    aws.String("obj-key"),
    })
```

Wenn Sie einen Servicevorgang aufrufen, validiert das SDK synchron die Eingabe, serialisiert die Anfrage, signiert sie mit Ihren Anmeldeinformationen, sendet sie an AWS und deserialisiert dann eine Antwort oder einen Fehler. In den meisten Fällen können Sie Service Operations direkt aufrufen. Jede Client-Methode für Dienstoperationen gibt eine Operationsantwort-Struktur und einen Fehlerschnittstellentyp zurück. Sie sollten immer den `error` Typ überprüfen, um festzustellen, ob ein Fehler aufgetreten ist, bevor Sie versuchen, auf die Antwortstruktur der Serviceoperation zuzugreifen.

Übergabe von Parametern an einen Dienstvorgang

Jede Dienstvorgangsmethode verwendet einen Context.Context-Wert, der zur Festlegung von Anforderungsfristen verwendet werden kann, die vom SDK eingehalten werden. Darüber hinaus verwendet jeder Dienstvorgang eine `<OperationName>Input` Struktur, die sich im jeweiligen Go-Paket des Dienstes befindet. Sie übergeben API-Eingabeparameter mithilfe der Eingabestruktur der Operation.

Eingabestrukturen für Operationen können Eingabeparameter wie die Standardtypen Go-Numerik, Boolean, String, Map und List haben. Bei komplexeren API-Vorgängen kann ein Dienst eine

komplexere Modellierung von Eingabeparametern aufweisen. Diese anderen Typen, wie z. B. dienstspezifische Strukturen und Aufzählungswerte, sind im `types` Go-Paket des Dienstes enthalten.

Darüber hinaus können Dienste zwischen dem Standardwert eines Go-Typs und der Frage, ob der Wert vom Benutzer festgelegt wurde oder nicht, unterscheiden. In diesen Fällen müssen Sie für Eingabeparameter möglicherweise eine Zeigerreferenz auf den betreffenden Typ übergeben. Für Standard-Go-Typen wie Numerik, Boolean und String stehen `<Type>` in der AWS `From<Type>` praktische Funktionen zur Verfügung, um diese Konvertierung zu vereinfachen. AWS.String kann beispielsweise verwendet werden, um a in einen `*string` Typ für Eingabeparameter `string` zu konvertieren, die einen Zeiger auf eine Zeichenfolge erfordern. Umgekehrt, aws. ToStringkann verwendet werden, um a in a `*string` zu transformieren `string` und schützt so vor der Dereferenzierung eines Nullzeigers. Die `To<Type>` Funktionen sind hilfreich bei der Bearbeitung von Serviceantworten.

Schauen wir uns ein Beispiel an, wie wir einen Amazon S3 S3-Client verwenden können, um die `GetObject` API aufzurufen, und unsere Eingabe mithilfe des `types` Pakets und der `aws.<Type>` Helfer erstellen können.

```
import "context"
    import "github.com/aws/aws-sdk-go-v2/config"
    import "github.com/aws/aws-sdk-go-v2/service/s3"
    import "github.com/aws/aws-sdk-go-v2/service/s3/types"

    // ...

    cfg, err := config.LoadDefaultConfig(context.TODO())
    if err != nil {
        panic(err)
    }

    client := s3.NewFromConfig(cfg)

    resp, err := client.GetObject(context.TODO(), &s3.GetObjectInput{
        Bucket:       aws.String("amzn-s3-demo-bucket"),
        Key:          aws.String("keyName"),
        RequestPayer: types.RequestPayerRequester,
    })
```

Client-Optionen für Operation Call überschreiben

Ähnlich wie Client-Betriebsoptionen während der Erstellung eines Clients mithilfe von funktionalen Argumenten geändert werden können, können die Client-Optionen beim Aufruf der Operationsmethode geändert werden, indem der Service-Operationsmethode ein oder mehrere funktionale Argumente zur Verfügung gestellt werden. Diese Aktion ist parallelitätssicher und wirkt sich nicht auf andere gleichzeitige Operationen auf dem Client aus.

Um beispielsweise die Client-Region von „us-west-2" auf „us-east-1" zu überschreiben:

```
cfg, err := config.LoadDefaultConfig(context.TODO(), config.WithRegion("us-west-2"))
    if err != nil {
        log.Printf("error: %v", err)
        return
    }

    client := s3.NewFromConfig(cfg)

    params := &s3.GetObjectInput{
        // ...
    }

    resp, err := client.GetObject(context.TODO(), params, func(o *Options) {
        o.Region = "us-east-1"
    })
```

Umgang mit Betriebsantworten

Jeder Dienstoperation ist eine Ausgabestruktur zugeordnet, die die Operation-Response-Elemente des Dienstes enthält. Die Ausgabestruktur folgt dem folgenden Benennungsmuster`<OperationName>Output`. Bei einigen Operationen sind möglicherweise keine Mitglieder für ihre Operationsausgabe definiert. Nach dem Aufrufen einer Dienstoperation sollte der Typ des `error` Rückgabearguments immer überprüft werden, um festzustellen, ob beim Aufrufen der Dienstoperation ein Fehler aufgetreten ist. Die zurückgegebenen Fehler können von Fehlern bei der clientseitigen Eingabeüberprüfung bis hin zu dienstseitigen Fehlerantworten reichen, die an den Client zurückgegeben werden. Auf die Ausgabestruktur der Operation sollte nicht zugegriffen werden, falls der Client einen Fehler zurückgibt, der ungleich Null ist.

Um beispielsweise einen Operationsfehler zu protokollieren und vorzeitig von der aufrufenden Funktion zurückzukehren:

```
response, err := client.GetObject(context.TODO())
    if err != nil {
        log.Printf("GetObject error: %v", err)
        return
    }
```

Weitere Informationen zur Fehlerbehandlung, einschließlich der Suche nach bestimmten Fehlertypen, finden Sie unter TODO

Antworten mit `io.ReadCloser`

Einige API-Operationen geben eine Antwortstruktur zurück, die ein Ausgabeelement enthält, bei dem es sich um einen handelt`io.ReadCloser`. Dies wird bei API-Operationen der Fall sein, bei denen ein Teil ihrer Ausgabe im Hauptteil der HTTP-Antwort selbst verfügbar gemacht wird.

Beispielsweise gibt der Amazon S3 `GetObject` S3-Vorgang eine Antwort zurück, deren `Body` Mitglied ein `io.ReadCloser` für den Zugriff auf die Objektnutzlast ist.

> **Warning**
>
> Sie MÜSSEN IMMER `Close()` beliebige `io.ReadCloser` Output-Elemente angeben, unabhängig davon, ob Sie den Inhalt konsumiert haben. Geschieht dies nicht, können Ressourcen verloren gehen und es kann zu Problemen beim Lesen der Antwortstellen für Operationen führen, die in future einberufen werden.

```
resp, err := s3svc.GetObject(context.TODO(), &s3.GetObjectInput{...})
    if err != nil {
        // handle error
        return
    }
    // Make sure to always close the response Body when finished
    defer resp.Body.Close()

    decoder := json.NewDecoder(resp.Body)
    if err := decoder.Decode(&myStruct); err != nil {
        // handle error
```

```
        return
    }
```

Metadaten der Antwort

Alle Ausgabestrukturen für Dienstoperationen enthalten ein `ResultMetadata` Element des Typs Middleware.Metadata. `middleware.Metadata`wird von der SDK-Middleware verwendet, um zusätzliche Informationen aus einer Dienstantwort bereitzustellen, die nicht vom Dienst modelliert wird. Dazu gehören Metadaten wie die. `RequestID` Um beispielsweise die mit einer Serviceanfrage `RequestID` verknüpfte Antwort abzurufen, um den AWS Support bei der Behebung einer Anfrage zu unterstützen:

```
import "fmt"
    import "log"
    import "github.com/aws/aws-sdk-go-v2/aws/middleware"
    import "github.com/aws/aws-sdk-go-v2/service/s3"

    // ..

    resp, err := client.GetObject(context.TODO(), &s3.GetObjectInput{
        // ...
    })
    if err != nil {
        log.Printf("error: %v", err)
        return
    }

    requestID, ok := middleware.GetRequestIDMetadata(resp.ResultMetadata)
    if !ok {
        fmt.Println("RequestID not included with request")
    }

    fmt.Printf("RequestID: %s\n", requestID)
```

Gleichzeitige Verwendung von Service-Clients

Sie können Goroutinen erstellen, die denselben Service-Client gleichzeitig verwenden, um mehrere Anfragen zu senden. Sie können einen Service-Client mit so vielen Goroutinen verwenden, wie Sie möchten.

Im folgenden Beispiel wird ein Amazon S3 S3-Serviceclient in mehreren Goroutinen verwendet. In diesem Beispiel werden zwei Objekte gleichzeitig in einen Amazon S3 S3-Bucket hochgeladen.

```
import "context"
    import "log"
    import "strings"
    import "github.com/aws/aws-sdk-go-v2/config"
    import "github.com/aws/aws-sdk-go-v2/service/s3"

    // ...

    cfg, err := config.LoadDefaultConfig(context.TODO())
    if err != nil {
        log.Printf("error: %v", err)
        return
    }

    client := s3.NewFromConfig(cfg)

    type result struct {
        Output *s3.PutObjectOutput
        Err    error
    }

    results := make(chan result, 2)

    var wg sync.WaitGroup
    wg.Add(2)

    go func() {
    defer wg.Done()
        output, err := client.PutObject(context.TODO(), &s3.PutObjectInput{
            Bucket: aws.String("amzn-s3-demo-bucket"),
            Key:    aws.String("foo"),
            Body:   strings.NewReader("foo body content"),
        })
        results <- result{Output: output, Err: err}
    }()

    go func() {
        defer wg.Done()
        output, err := client.PutObject(context.TODO(), &s3.PutObjectInput{
            Bucket: aws.String("amzn-s3-demo-bucket"),
```

```
            Key:    aws.String("bar"),
            Body:   strings.NewReader("bar body content"),
        })
        results <- result{Output: output, Err: err}
    }()

    wg.Wait()

    close(results)

    for result := range results {
        if result.Err != nil {
            log.Printf("error: %v", result.Err)
            continue
        }
        fmt.Printf("etag: %v", aws.ToString(result.Output.ETag))
    }
```

Verwendung von Operation Paginators

Wenn Sie eine Liste von Elementen abrufen, müssen Sie in der Regel in der Ausgabestruktur nach einem Token oder einer Markierung suchen, um zu überprüfen, ob der AWS Dienst alle Ergebnisse Ihrer Anfrage zurückgegeben hat. Wenn das Token oder die Markierung vorhanden ist, verwenden Sie sie, um die nächste Ergebnisseite anzufordern. Anstatt diese Token oder Markierungen zu verwalten, können Sie die verfügbaren Paginatortypen des Servicepakets verwenden.

Paginator-Helfer sind für unterstützte Serviceoperationen verfügbar und befinden sich im Go-Paket des Service-Clients. Verwenden Sie die Funktion, um einen Paginator für eine unterstützte Operation zu erstellen. `New<OperationName>Paginator` Paginator-Konstruktfunktionen verwenden den Dienst`Client`, die `<OperationName>Input` Eingabeparameter der Operation und einen optionalen Satz von Funktionsargumenten, mit denen Sie weitere optionale Paginatoreinstellungen konfigurieren können.

Der Paginatortyp für zurückgegebene Operationen bietet eine bequeme Möglichkeit, über eine paginierte Operation zu iterieren, bis Sie die letzte Seite erreicht haben oder die Elemente gefunden haben, nach denen Ihre Anwendung gesucht hat. Ein Paginatortyp hat zwei Methoden: und. `HasMorePages` `NextPage` `HasMorePages`gibt einen booleschen Wert zurück, `true` wenn die erste Seite nicht abgerufen wurde oder ob weitere Seiten verfügbar sind, die mit der Operation abgerufen werden können. Um die ersten oder die nachfolgenden Seiten der Operation abzurufen,

muss die `NextPage` Operation aufgerufen werden. `NextPage`nimmt die Operationsausgabe `context.Context` und alle entsprechenden Fehler auf und gibt sie zurück. Wie bei den Rückgabeparametern der Client-Operationsmethode sollte der Rückgabefehler immer überprüft werden, bevor versucht wird, die zurückgegebene Antwortstruktur zu verwenden. Siehe Umgang mit Betriebsantworten.

Im folgenden Beispiel wird der `ListObjectsV2` Paginator verwendet, um bis zu drei Seiten mit Objektschlüsseln aus der `ListObjectV2` Operation aufzulisten. Jede Seite besteht aus bis zu 10 Schlüsseln, was durch die `Limit` Paginator-Option definiert wird.

```
import "context"
    import "log"
    import "github.com/aws/aws-sdk-go-v2/config"
    import "github.com/aws/aws-sdk-go-v2/aws"
    import "github.com/aws/aws-sdk-go-v2/service/s3"

    // ...

    cfg, err := config.LoadDefaultConfig(context.TODO())
    if err != nil {
        log.Printf("error: %v", err)
        return
    }

    client := s3.NewFromConfig(cfg)

    params := &s3.ListObjectsV2Input{
        Bucket: aws.String("amzn-s3-demo-bucket"),
    }

    paginator := s3.NewListObjectsV2Paginator(client, params, func(o
 *s3.ListObjectsV2PaginatorOptions) {
        o.Limit = 10
    })

    pageNum := 0
    for paginator.HasMorePages() && pageNum < 3 {
        output, err := paginator.NextPage(context.TODO())
        if err != nil {
            log.Printf("error: %v", err)
            return
        }
```

```
        for _, value := range output.Contents {
            fmt.Println(*value.Key)
        }
        pageNum++
    }
```

Ähnlich wie bei der Client-Betriebsmethode können die Client-Optionen wie die Anforderungsregion geändert werden, indem ein oder mehrere funktionale Argumente für angegeben werden. `NextPage` Weitere Hinweise zum Überschreiben von Client-Optionen beim Aufrufen einer Operation finden Sie unterClient-Optionen für Operation Call überschreiben.

Verwenden von Waiter-Objekten

Bei der asynchronen Interaktion mit AWS APIs diesen Ressourcen müssen Sie oft warten, bis eine bestimmte Ressource verfügbar ist, um weitere Aktionen darauf ausführen zu können.

Beispielsweise kehrt die Amazon DynamoDB `CreateTable` DynamoDB-API sofort mit dem Wert CREATING TableStatus zurück, und Sie können keine Lese- oder Schreiboperationen aufrufen, bis der Tabellenstatus auf den Tabellenstatus umgestellt wurde. `ACTIVE`

Das Schreiben von Logik zur kontinuierlichen Abfrage des Tabellenstatus kann umständlich und fehleranfällig sein. Die Kellner helfen dabei, die Komplexität zu verringern, und sind einfach und erledigen APIs die Abfrageaufgabe für Sie.

Sie können beispielsweise waiters verwenden, um abzufragen, ob eine DynamoDB-Tabelle erstellt wurde und für einen Schreibvorgang bereit ist.

```
import "context"
    import "fmt"
    import "log"
    import "time"
    import "github.com/aws/aws-sdk-go-v2/aws"
    import "github.com/aws/aws-sdk-go-v2/config"
    import "github.com/aws/aws-sdk-go-v2/service/dynamodb"

    // ...

    cfg, err := config.LoadDefaultConfig(context.TODO())
    if err != nil {
        log.Printf("error: %v", err)
```

```
        return
    }

    client := dynamodb.NewFromConfig(cfg)

    // we create a waiter instance by directly passing in a client
    // that satisfies the waiters client Interface.
    waiter :=  dynamodb.NewTableExistsWaiter(client)

    // params is the input to api operation used by the waiter
    params := &dynamodb.DescribeTableInput {
        TableName: aws.String("test-table")
    }

    // maxWaitTime is the maximum wait time, the waiter will wait for
    // the resource status.
    maxWaitTime := 5 * time.Minutes

    // Wait will poll until it gets the resource status, or max wait time
    // expires.
    err := waiter.Wait(context.TODO(), params, maxWaitTime)
    if err != nil {
        log.Printf("error: %v", err)
        return
    }
    fmt.Println("Dynamodb table is now ready for write operations")
```

Die Kellner-Konfiguration wird außer Kraft gesetzt

Standardmäßig verwendet das SDK die Werte für die minimale Verzögerung und die maximale Verzögerung, die mit den optimalen Werten konfiguriert sind, die von den AWS Diensten für verschiedene Dienste definiert werden. APIs Sie können die Kellnerkonfiguration überschreiben, indem Sie beim Aufbau des Kellners oder beim Aufrufen eines Kellnervorgangs Funktionsoptionen angeben.

Zum Beispiel, um die Kellnerkonfiguration während der Installation des Kellners zu überschreiben

```
import "context"
    import "fmt"
    import "log"
    import "time"
```

```
    import "github.com/aws/aws-sdk-go-v2/aws"
    import "github.com/aws/aws-sdk-go-v2/config"
    import "github.com/aws/aws-sdk-go-v2/service/dynamodb"

    // ...

    cfg, err := config.LoadDefaultConfig(context.TODO())
    if err != nil {
        log.Printf("error: %v", err)
        return
    }

    client := dynamodb.NewFromConfig(cfg)

    // we create a waiter instance by directly passing in a client
    // that satisfies the waiters client Interface.
    waiter :=  dynamodb.NewTableExistsWaiter(client, func (o
 *dynamodb.TableExistsWaiterOptions) {

        // override minimum delay to 10 seconds
        o.MinDelay = 10 * time.Second

        // override maximum default delay to 300 seconds
        o.MaxDelay = 300 * time.Second
    })
```

Die `Wait` Funktion für jeden Kellner umfasst auch funktionale Optionen. Ähnlich wie im obigen Beispiel können Sie die Kellnerkonfiguration pro `Wait` Anfrage überschreiben.

```
// params is the input to api operation used by the waiter
    params := &dynamodb.DescribeTableInput {
        TableName: aws.String("test-table")
    }

    // maxWaitTime is the maximum wait time, the waiter will wait for
    // the resource status.
    maxWaitTime := 5 * time.Minutes

    // Wait will poll until it gets the resource status, or max wait time
    // expires.
    err := waiter.Wait(context.TODO(), params, maxWaitTime, func (o
 *dynamodb.TableExistsWaiterOptions) {
```

```
        // override minimum delay to 5 seconds
        o.MinDelay = 5 * time.Second

        // override maximum default delay to 120 seconds
        o.MaxDelay = 120 * time.Second
    })
    if err != nil {
        log.Printf("error: %v", err)
        return
    }
    fmt.Println("Dynamodb table is now ready for write operations")
```

Die erweiterte Kellnerkonfiguration hat Vorrang

Sie können das Standardverhalten des Kellners zusätzlich anpassen, indem Sie eine benutzerdefinierte Wiederholungsfunktion bereitstellen. Die kellnerspezifischen Optionen ermöglichen auch die Anpassung von Betriebs-Middlewares. `APIOptions`

Zum Beispiel, um erweiterte Einstellungen für Kellner zu konfigurieren.

```
import "context"
    import "fmt"
    import "log"
    import "time"
    import "github.com/aws/aws-sdk-go-v2/aws"
    import "github.com/aws/aws-sdk-go-v2/config"
    import "github.com/aws/aws-sdk-go-v2/service/dynamodb"
    import "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
    // ...

    cfg, err := config.LoadDefaultConfig(context.TODO())
    if err != nil {
        log.Printf("error: %v", err)
        return
    }

    client := dynamodb.NewFromConfig(cfg)

    // custom retryable defines if a waiter state is retryable or a terminal state.
    // For example purposes, we will configure the waiter to not wait
    // if table status is returned as `UPDATING`
```

```
    customRetryable := func(ctx context.Context, params *dynamodb.DescribeTableInput,
        output *dynamodb.DescribeTableOutput, err error) (bool, error) {
        if output.Table != nil {
            if output.Table.TableStatus == types.TableStatusUpdating {
                // if table status is `UPDATING`, no need to wait
                return false, nil
            }
        }
    }

    // we create a waiter instance by directly passing in a client
    // that satisfies the waiters client Interface.
    waiter :=  dynamodb.NewTableExistsWaiter(client, func (o
 *dynamodb.TableExistsWaiterOptions) {

        // override the service defined waiter-behavior
        o.Retryable = customRetryable
    })
```

Umgang mit Fehlern in der AWS SDK für Go V2

Die AWS SDK für Go Rückgabefehler, die dem `error` Go-Schnittstellentyp entsprechen. Sie können die `Error()` Methode verwenden, um eine formatierte Zeichenfolge der SDK-Fehlermeldung ohne besondere Behandlung abzurufen. Fehler, die vom SDK zurückgegeben werden, können eine `Unwrap` Methode implementieren. Die `Unwrap` Methode wird vom SDK verwendet, um zusätzliche Kontextinformationen zu Fehlern bereitzustellen und gleichzeitig Zugriff auf den zugrunde liegenden Fehler oder die Fehlerkette zu erhalten. Die `Unwrap` Methode sollte zusammen mit Errors.as verwendet werden, um das Entpacken von Fehlerketten zu behandeln.

Es ist wichtig, dass Ihre Anwendung überprüft, ob nach dem Aufrufen einer Funktion oder Methode, die einen Schnittstellentyp zurückgeben kann, ein Fehler aufgetreten ist. `error` Die einfachste Form der Fehlerbehandlung ähnelt dem folgenden Beispiel:

```
if err != nil {
    // Handle error
    return
}
```

Fehler protokollieren

Die einfachste Form der Fehlerbehandlung besteht traditionell darin, die Fehlermeldung zu protokollieren oder auszudrucken, bevor Sie zur Anwendung zurückkehren oder sie verlassen.

```
import "log"

// ...

if err != nil {
    log.Printf("error: %s", err.Error())
    return
}
```

Fehler beim Service Client

Das SDK fasst alle von Service Clients zurückgegebenen Fehler mit dem Smithy zusammen. OperationErrorArt des Fehlers. `OperationError`stellt kontextbezogene Informationen über den Dienstnamen und den Vorgang bereit, der mit einem zugrunde liegenden Fehler verknüpft ist. Diese Informationen können für Anwendungen nützlich sein, die stapelweise Operationen für einen oder mehrere Dienste ausführen und über einen zentralen Fehlerbehandlungsmechanismus verfügen. Ihre Anwendung kann für den Zugriff `errors.As` auf diese `OperationError` Metadaten verwendet werden.

```
import "log"
import "github.com/aws/smithy-go"

// ...

if err != nil {
    var oe *smithy.OperationError
    if errors.As(err, &oe) {
        log.Printf("failed to call service: %s, operation: %s, error: %v",
 oe.Service(), oe.Operation(), oe.Unwrap())
    }
    return
}
```

Antworten auf API-Fehler

Serviceoperationen können modellierte Fehlertypen zurückgeben, um auf bestimmte Fehler hinzuweisen. Diese modellierten Typen können verwendet werden`errors.As`, um herauszufinden, ob der Ausfall des Vorgangs auf einen bestimmten Fehler zurückzuführen ist. Amazon S3 `CreateBucket` kann beispielsweise einen BucketAlreadyExistsFehler zurückgeben, wenn bereits ein Bucket mit demselben Namen existiert.

Um beispielsweise zu überprüfen, ob ein Fehler ein `BucketAlreadyExists` Fehler war:

```
import "log"
import "github.com/aws/aws-sdk-go-v2/service/s3/types"

// ...

if err != nil {
    var bne *types.BucketAlreadyExists
    if errors.As(err, &bne) {
        log.Println("error:", bne)
    }
    return
}
```

Alle Antwortfehler der Service-API implementieren den Smithy. APIErrorSchnittstellentyp.
Diese Schnittstelle kann verwendet werden, um sowohl modellierte als auch nicht modellierte Servicefehlerantworten zu verarbeiten. Dieser Typ ermöglicht den Zugriff auf den Fehlercode und die vom Dienst zurückgegebene Meldung. Darüber hinaus gibt dieser Typ an, ob der Fehler auf den Client oder Server zurückzuführen ist, sofern bekannt.

```
import "log"
import "github.com/aws/smithy-go"

// ...

if err != nil {
    var ae smithy.APIError
    if errors.As(err, &ae) {
        log.Printf("code: %s, message: %s, fault: %s", ae.ErrorCode(),
 ae.ErrorMessage(), ae.ErrorFault().String())
    }
    return
```

```
}
```

Anforderungskennungen werden abgerufen

Wenn Sie mit dem AWS Support zusammenarbeiten, werden Sie möglicherweise aufgefordert, die Anfrage-ID anzugeben, die die Anfrage identifiziert, die Sie zu beheben versuchen. Sie können http verwenden. ResponseErrorund verwenden Sie die `ServiceRequestID()` Methode, um die Anforderungskennung abzurufen, die der Fehlerantwort zugeordnet ist.

```
import "log"
import awshttp "github.com/aws/aws-sdk-go-v2/aws/transport/http"

// ...

if err != nil {
    var re *awshttp.ResponseError
    if errors.As(err, &re) {
        log.Printf("requestID: %s, error: %v", re.ServiceRequestID(), re.Unwrap());
    }
    return
}
```

Amazon S3 S3-Anforderungskennungen

Amazon S3 S3-Anfragen enthalten zusätzliche Identifikatoren, die verwendet werden können, um den AWS Support bei der Behebung Ihrer Anfrage zu unterstützen. Sie können s3 verwenden. ResponseErrorund rufen Sie an `ServiceRequestID()` und `ServiceHostID()` um die Anfrage-ID und die Host-ID abzurufen.

```
import "log"
import "github.com/aws/aws-sdk-go-v2/service/s3"

// ...

if err != nil {
    var re s3.ResponseError
    if errors.As(err, &re) {
        log.Printf("requestID: %s, hostID: %s request failure", re.ServiceRequestID(),
 re.ServiceHostID());
    }
    return
```

```
}
```

Verwenden Sie die AWS SDK für Go Version 2 mit AWS Diensten

Um einen AWS Dienst aufzurufen, müssen Sie zunächst eine Service-Client-Instanz erstellen. Ein Service-Client bietet Low-Level-Zugriff auf jede API-Aktion für diesen Dienst. Sie erstellen beispielsweise einen Amazon S3-Serviceclient, um Amazon S3 aufzurufen APIs.

Wenn Sie Service Operations aufrufen, übergeben Sie Eingabeparameter als Struktur. Ein erfolgreicher Aufruf führt zu einer Ausgabestruktur, die die Antwort der Service-API enthält. Wenn Sie beispielsweise erfolgreich eine Amazon S3 S3-Aktion zum Erstellen eines Buckets aufgerufen haben, gibt die Aktion eine Ausgabestruktur mit dem Speicherort des Buckets zurück.

Eine Liste der Service-Clients, einschließlich ihrer Methoden und Parameter, finden Sie in der AWS SDK für Go API-Referenz.

Schutz der Datenintegrität mit Prüfsummen

Amazon Simple Storage Service (Amazon S3) bietet die Möglichkeit, beim Hochladen eines Objekts eine Prüfsumme anzugeben. Wenn Sie eine Prüfsumme angeben, wird diese zusammen mit dem Objekt gespeichert und kann beim Herunterladen des Objekts überprüft werden.

Prüfsummen bieten eine zusätzliche Ebene der Datenintegrität bei der Übertragung von Dateien. Mit Prüfsummen können Sie die Datenkonsistenz überprüfen, indem Sie sicherstellen, dass die empfangene Datei mit der Originaldatei übereinstimmt. Weitere Informationen zu Prüfsummen mit Amazon S3 finden Sie im Amazon Simple Storage Service-Benutzerhandbuch, einschließlich der unterstützten Algorithmen.

Sie haben die Flexibilität, den Algorithmus auszuwählen, der Ihren Anforderungen am besten entspricht, und das SDK die Prüfsumme berechnen zu lassen. Alternativ können Sie mithilfe eines der unterstützten Algorithmen einen vorab berechneten Prüfsummenwert angeben.

> **Note**
>
> Ab Version v1.74.1 des Amazon S3 S3-Moduls bietet das SDK standardmäßigen Integritätsschutz, indem es automatisch eine `CRC32` Prüfsumme für Uploads berechnet. Das SDK berechnet diese Prüfsumme, wenn Sie keinen vorab berechneten Prüfsummenwert angeben oder wenn Sie keinen Algorithmus angeben, den das SDK zur Berechnung einer Prüfsumme verwenden soll.

Das SDK bietet auch globale Einstellungen für den Schutz der Datenintegrität, die Sie extern festlegen können. Weitere Informationen finden Sie im Referenzhandbuch und im Tools-Referenzhandbuch.AWS SDKs

Wir behandeln Prüfsummen in zwei Anforderungsphasen: beim Hochladen eines Objekts und beim Herunterladen eines Objekts.

Hochladen eines Objekts

Wenn Sie ein Objekt mit der `putObject` Methode hochladen und einen Prüfsummenalgorithmus bereitstellen, berechnet das SDK die Prüfsumme für den angegebenen Algorithmus.

Der folgende Codeausschnitt zeigt eine Anfrage zum Hochladen eines Objekts mit einer Prüfsumme. `CRC32` Wenn das SDK die Anfrage sendet, berechnet es die `CRC32` Prüfsumme und lädt das Objekt hoch. Amazon S3 validiert die Integrität des Inhalts, indem es die Prüfsumme berechnet und mit der vom SDK bereitgestellten Prüfsumme vergleicht. Amazon S3 speichert dann die Prüfsumme mit dem Objekt.

```
out, err := s3Client.PutObject(context.Background(), &s3.PutObjectInput{
     Bucket:            aws.String("bucket"),
     Key:               aws.String("key"),
     ChecksumAlgorithm: types.ChecksumAlgorithmCrc32,
     Body:              strings.NewReader("Hello World"),
 })
```

Wenn Sie mit der Anfrage keinen Prüfsummenalgorithmus angeben, variiert das Prüfsummenverhalten je nach der Version des verwendeten SDK, wie in der folgenden Tabelle dargestellt.

Prüfsummenverhalten, wenn kein Prüfsummenalgorithmus bereitgestellt wird

Amazon S3 S3-Modulversion von AWS SDK für Go	Verhalten der Prüfsumme
Früher als v1.74.1	Das SDK berechnet nicht automatisch eine CRC-basierte Prüfsumme und gibt sie in der Anfrage an.

Amazon S3 S3-Modulversion von AWS SDK für Go	Verhalten der Prüfsumme
v1.74.1 oder höher	Das SDK verwendet den CRC32 Algorithm us zur Berechnung der Prüfsumme und stellt sie in der Anfrage bereit. Amazon S3 validiert die Integrität der Übertragung, indem es seine eigene CRC32 Prüfsumme berechnet und sie mit der vom SDK bereitgestellten Prüfsumme vergleicht. Wenn die Prüfsummen übereinst immen, wird die Prüfsumme zusammen mit dem Objekt gespeichert.

Verwenden Sie einen vorberechneten Prüfsummenwert

Ein mit der Anfrage bereitgestellter vorberechneter Prüfsummenwert deaktiviert die automatische Berechnung durch das SDK und verwendet stattdessen den angegebenen Wert.

Das folgende Beispiel zeigt eine Anfrage mit einer vorberechneten Prüfsumme. SHA256

```
out, err := s3Client.PutObject(context.Background(), &s3.PutObjectInput{
     Bucket:        aws.String("bucket"),
     Key:           aws.String("key"),
     ChecksumCRC32: aws.String("checksumvalue"),
     Body:          strings.NewReader("Hello World"),
 })
```

Wenn Amazon S3 feststellt, dass der Prüfsummenwert für den angegebenen Algorithmus falsch ist, gibt der Service eine Fehlerantwort zurück.

Mehrteilige Uploads

Sie können Prüfsummen auch bei mehrteiligen Uploads verwenden.

Das AWS SDK für Go bietet zwei Optionen zur Verwendung von Prüfsummen bei mehrteiligen Uploads. Die erste Option verwendet den Transfermanager, der den CRC32 Algorithmus für den Upload spezifiziert.

```
s3Client := s3.NewFromConfig(cfg)
```

```
    transferManager := manager.NewUploader(s3Client)
    out, err := transferManager.Upload(context.Background(), &s3.PutObjectInput{
            Bucket:            aws.String("bucket"),
            Key:               aws.String("key"),
            Body:              large file to trigger multipart upload,
            ChecksumAlgorithm: types.ChecksumAlgorithmCrc32,
    })
```

Wenn Sie bei der Verwendung des Transfer-Managers für Uploads keinen Prüfsummenalgorithmus angeben, berechnet das SDK automatisch eine Prüfsumme auf der Grundlage des Algorithmus. `CRC32` Das SDK führt diese Berechnung für alle Versionen des SDK durch.

Die zweite Option verwendet den Amazon S3 S3-Client, um den mehrteiligen Upload durchzuführen. Wenn Sie bei diesem Ansatz eine Prüfsumme angeben, müssen Sie den Algorithmus angeben, der bei der Initiierung des Uploads verwendet werden soll. Sie müssen auch den Algorithmus für jede Teilanforderung angeben und die für jedes Teil nach dem Hochladen berechnete Prüfsumme bereitstellen.

```
s3Client := s3.NewFromConfig(cfg)
    createMultipartUploadOutput, err :=
 s3Client.CreateMultipartUpload(context.Background(), &s3.CreateMultipartUploadInput{
            Bucket:            aws.String("bucket"),
            Key:               aws.String("key"),
            ChecksumAlgorithm: types.ChecksumAlgorithmCrc32,
         })
    if err != nil {
            log.Fatal("err create multipart upload ", err)
    }

    var partsBody []io.Reader // this is just an example parts content, you should
 load your target file in your code
    partNum := int32(1)
    var completedParts []types.CompletedPart
    for _, body := range partsBody {
        uploadPartOutput, err := s3Client.UploadPart(context.Background(),
 &s3.UploadPartInput{
            Bucket:            aws.String("bucket"),
            Key:               aws.String("key"),
            ChecksumAlgorithm: types.ChecksumAlgorithmCrc32,
            Body:              body,
            PartNumber:        aws.Int32(partNum),
            UploadId:          createMultipartUploadOutput.UploadId,
```

```
        })
        if err != nil {
            log.Fatal("err upload part ", err)
        }

        completedParts = append(completedParts, types.CompletedPart{
            PartNumber:    aws.Int32(partNum),
            ETag:          uploadPartOutput.ETag,
            ChecksumCRC32: uploadPartOutput.ChecksumCRC32,
        })
        partNum++
    }

    completeMultipartUploadOutput, err :=
s3Client.CompleteMultipartUpload(context.Background(),
&s3.CompleteMultipartUploadInput{
        Bucket:   aws.String("bucket"),
        Key:      aws.String("key"),
        UploadId: createMultipartUploadOutput.UploadId,
        MultipartUpload: &types.CompletedMultipartUpload{
            Parts: completedParts,
        },
    })
    if err != nil {
        log.Fatal("err complete multipart upload ", err)
    }
```

Herunterladen eines Objekts

Wenn Sie die GetObjectMethode verwenden, um ein Objekt herunterzuladen, validiert das SDK automatisch die Prüfsumme, wenn das `ChecksumMode` Feld von auf gesetzt `GetObjectInput` ist. `types.ChecksumModeEnabled`

Die Anfrage im folgenden Codeausschnitt weist das SDK an, die Prüfsumme in der Antwort zu validieren, indem es die Prüfsumme berechnet und die Werte vergleicht.

```
out, err := s3Client.GetObject(context.Background(), &s3.GetObjectInput{
    Bucket:       aws.String("bucket"),
    Key:          aws.String("key"),
    ChecksumMode: types.ChecksumModeEnabled,
 })
```

Wenn das Objekt nicht mit einer Prüfsumme hochgeladen wurde, findet keine Überprüfung statt.

Verwenden der AWS SDK für Go Dienstprogramme

Das AWS SDK für Go beinhaltet die folgenden Dienstprogramme, mit denen Sie AWS Dienste einfacher nutzen können. Die SDK-Dienstprogramme finden Sie in den entsprechenden AWS Servicepaketen.

Amazon RDS-Dienstprogramme

IAM-Authentifizierung

Das Auth-Paket bietet Hilfsprogramme zum Generieren von Authentifizierungstoken für die Verbindung mit Amazon RDS-MySQL- und PostgreSQL-Datenbank-Instances. Mithilfe BuildAuthTokendieser Methode generieren Sie ein Datenbank-Autorisierungstoken, indem Sie den Datenbank-Endpunkt, die AWS Region, den Benutzernamen und eine AWS-Datei angeben. CredentialProviderImplementierung, die IAM-Anmeldeinformationen mit der Berechtigung zurückgibt, mithilfe der IAM-Datenbankauthentifizierung eine Verbindung zur Datenbank herzustellen. Weitere Informationen zur Konfiguration von Amazon RDS mit IAM-Authentifizierung finden Sie in den folgenden Ressourcen des Amazon RDS Developer Guide:

- Aktivierung und Deaktivierung der IAM-Datenbankauthentifizierung
- Erstellen und Verwenden einer IAM-Richtlinie für den IAM-Datenbankzugriff
- Erstellen eines Datenbankkontos mithilfe der IAM-Authentifizierung

Das folgende Beispiel zeigt, wie ein Authentifizierungstoken generiert wird, um eine Verbindung zu einer Amazon RDS-Datenbank herzustellen:

```
import "context"
import "github.com/aws/aws-sdk-go-v2/config"
import "github.com/aws/aws-sdk-go-v2/feature/rds/auth"

// ...

cfg, err := config.LoadDefaultConfig(context.TODO())
if err != nil {
    panic("configuration error: " + err.Error())
}

authenticationToken, err := auth.BuildAuthToken(
```

```
    context.TODO(),
    "mydb.123456789012.us-east-1.rds.amazonaws.com:3306", // Database Endpoint (With
 Port)
    "us-east-1", // AWS Region
    "jane_doe", // Database Account
    cfg.Credentials,
)
if err != nil {
    panic("failed to create authentication token: " + err.Error())
}
```

Amazon CloudFront Utilities

CloudFront Amazon-URL-Signaturer

Der CloudFront Amazon-URL-Signaturer vereinfacht den Prozess der Erstellung signierter URLs URLs. Eine signierte URL enthält Informationen wie ein Ablaufdatum und eine Ablaufzeit, mit denen Sie den Zugriff auf Ihre Inhalte kontrollieren können. Signierte Inhalte URLs sind nützlich, wenn Sie Inhalte über das Internet verteilen, den Zugriff jedoch auf bestimmte Benutzer beschränken möchten (z. B. auf Benutzer, die eine Gebühr bezahlt haben).

Um eine URL zu signieren, erstellen Sie eine `URLSigner` Instanz mit Ihrer CloudFront Schlüsselpaar-ID und dem zugehörigen privaten Schlüssel. Rufen Sie dann die `SignWithPolicy` Methode `Sign` or auf und geben Sie die URL an, die signiert werden soll. Weitere Informationen zu CloudFront Amazon-Schlüsselpaaren finden Sie unter Creating CloudFront Key Pairs for Your Trusted Signers im CloudFront Developer Guide.

Im folgenden Beispiel wird eine signierte URL erstellt, die nach ihrer Erstellung noch eine Stunde gültig ist.

```
import "github.com/aws/aws-sdk-go-v2/feature/cloudfront/sign"

// ...

signer := sign.NewURLSigner(keyID, privKey)

signedURL, err := signer.Sign(rawURL, time.Now().Add(1*time.Hour))
if err != nil {
    log.Fatalf("Failed to sign url, err: %s\n", err.Error())
    return
```

```
}
```

Weitere Informationen zum Signaturprogramm finden Sie im Abschnitt zum Sign-Paket in der AWS SDK für Go API-Referenz.

Metadatenservice für EC2 Amazon-Instances

Sie können den verwenden AWS SDK für Go , um auf den Amazon EC2 Instance Metadata Service zuzugreifen. Das feature/ec2/imdsGo-Paket bietet einen Client-Typ, der für den Zugriff auf den Amazon EC2 Instance Metadata Service verwendet werden kann. Die `Client` und die zugehörigen Operationen können ähnlich wie die anderen vom SDK bereitgestellten AWS Service-Clients verwendet werden. Weitere Informationen zur Konfiguration des SDK und zur Verwendung von Service-Clients finden Sie unter Das SDKs konfigurieren undVerwenden Sie die AWS SDK für Go Version 2 mit AWS Diensten.

Mit dem Client können Sie auf einfache Weise Informationen über Instances abrufen, auf denen Ihre Anwendungen ausgeführt werden, z. B. die AWS Region oder die lokale IP-Adresse. In der Regel müssen Sie HTTP-Anfragen erstellen und einreichen, um Instanz-Metadaten abzurufen. Erstellen Sie stattdessen einen, `imds.Client` um wie andere AWS Services mithilfe eines programmatischen Clients auf den Amazon EC2 Instance Metadata Service zuzugreifen.

Um beispielsweise einen Client zu erstellen:

```
import "context"
import "github.com/aws/aws-sdk-go-v2/config"
import "github.com/aws/aws-sdk-go-v2/feature/ec2/imds"

// ...

cfg, err := config.LoadDefaultConfig(context.TODO())
if err != nil {
    log.Printf("error: %v", err)
    return
}

client := imds.NewFromConfig(cfg)
```

Verwenden Sie dann den Service-Client, um Informationen aus einer Metadatenkategorie wie `local-ipv4` (der privaten IP-Adresse der Instanz) abzurufen.

```
localIp, err := client.GetMetadata(context.TODO(), &imds.GetMetadataInput{
    Path: "local-ipv4",
})
if err != nil {
    log.Printf("Unable to retrieve the private IP address from the EC2 instance: %s\n",
 err)
    return
}
content, _ := io.ReadAll(localIp.Content)
fmt.Printf("local-ip: %v\n", string(content))
```

Eine Liste aller Metadatenkategorien finden Sie unter Kategorien von Instance-Metadaten im EC2 Amazon-Benutzerhandbuch.

Amazon S3 S3-Dienstprogramme

Amazon S3 S3-Übertragungsmanager

Die Upload- und Download-Manager von Amazon S3 können große Objekte aufteilen, sodass sie in mehreren Teilen parallel übertragen werden können. Dies macht es einfach, unterbrochene Übertragungen wieder aufzunehmen.

Amazon S3 S3-Uploadmanager

Der Amazon S3 S3-Upload-Manager bestimmt, ob eine Datei in kleinere Teile aufgeteilt und parallel hochgeladen werden kann. Sie können die Anzahl der parallel Uploads und die Größe der hochgeladenen Teile anpassen.

Das folgende Beispiel verwendet Amazon S3`Uploader`, um eine Datei hochzuladen. `Uploader`Die Verwendung ähnelt der `s3.PutObject()` Operation.

```
import "context"
import "github.com/aws/aws-sdk-go-v2/config"
import "github.com/aws/aws-sdk-go-v2/service/s3"
import "github.com/aws/aws-sdk-go-v2/feature/s3/manager"

// ...

cfg, err := config.LoadDefaultConfig(context.TODO())
if err != nil {
    log.Printf("error: %v", err)
```

```
    return
}

client := s3.NewFromConfig(cfg)

uploader := manager.NewUploader(client)
result, err := uploader.Upload(context.TODO(), &s3.PutObjectInput{
    Bucket: aws.String("amzn-s3-demo-bucket"),
    Key:    aws.String("my-object-key"),
    Body:   uploadFile,
})
```

Konfigurationsoptionen

Wenn Sie eine `Uploader` Instanz mithilfe instanziieren NewUploader, können Sie verschiedene Konfigurationsoptionen angeben, um anzupassen, wie Objekte hochgeladen werden. Optionen werden überschrieben, indem Sie ein oder mehrere Argumente für angeben. `NewUploader` Sie haben u. a. folgende Möglichkeiten:

- `PartSize`— Gibt die Puffergröße jedes hochzuladenden Teils in Byte an. Die Mindestgröße pro Teil beträgt 5 MiB.
- `Concurrency`— Gibt die Anzahl der Teile an, die parallel hochgeladen werden sollen.
- `LeavePartsOnError`— Gibt an, ob erfolgreich hochgeladene Teile in Amazon S3 belassen werden sollen.

Der `Concurrency` Wert begrenzt die Anzahl gleichzeitiger Teil-Uploads, die bei einem bestimmten `Upload` Anruf durchgeführt werden können. Dies ist kein globales Limit für die Parallelität von Clients. Passen Sie die Werte `PartSize` und die `Concurrency` Konfiguration an, um die optimale Konfiguration zu finden. Beispielsweise können Systeme mit Verbindungen mit hoher Bandbreite größere Teile und mehr Uploads parallel senden.

Ihre Anwendung wird beispielsweise `Uploader` mit der Einstellung `Concurrency` of von konfiguriert. `5` Wenn Ihre Anwendung dann `Upload` von zwei verschiedenen Goroutinen aus aufruft, sind das Ergebnis `10` gleichzeitige Teil-Uploads (2 Goroutinen * 5). `Concurrency`

> **Warning**
>
> Es wird erwartet, dass Ihre Anwendung die gleichzeitigen Aufrufe begrenzt, um eine Erschöpfung der Anwendungsressourcen zu verhindern. `Upload`

Im Folgenden finden Sie ein Beispiel für die Festlegung der Standardgröße für Bauteile bei der `Uploader` Erstellung:

```
uploader := manager.NewUploader(client, func(u *Uploader) {
    u.PartSize = 10 * 1024 * 1024, // 10 MiB
})
```

Weitere Informationen zu `Uploader` und seinen Konfigurationen finden Sie unter Uploader in der AWS SDK für Go API-Referenz.

PutObjectInput Feld „Körper“ (io). ReadSeeker gegen io.Reader)

Das `Body` Feld der `s3.PutObjectInput` Struktur ist ein `io.Reader` Typ. Dieses Feld kann jedoch mit einem Typ gefüllt werden, der sowohl die `io.ReaderAt` Schnittstelle als auch erfüllt, um die `io.ReadSeeker` Nutzung der Anwendungsressourcen in der Host-Umgebung zu verbessern. Im folgenden Beispiel wird der Typ erstelltReadSeekerAt, der beide Schnittstellen erfüllt:

```
type ReadSeekerAt interface {
    io.ReadSeeker
    io.ReaderAt
}
```

Bei `io.Reader` Typen müssen die Bytes des Lesegeräts im Speicher gepuffert werden, bevor der Teil hochgeladen werden kann. Wenn Sie den `Concurrency` Wert `PartSize` oder erhöhen, `Uploader` erhöht sich der benötigte Arbeitsspeicher (RAM) für erheblich. Der benötigte Arbeitsspeicher beträgt ungefähr *`PartSize`* `Concurrency`*. Wenn Sie beispielsweise 100 MB für `PartSize` und 10 für angeben`Concurrency`, sind mindestens 1 GB erforderlich.

Da ein `io.Reader` Typ seine Größe nicht vor dem Lesen seiner Byte bestimmen `Uploader` kann, kann nicht berechnet werden, wie viele Teile hochgeladen werden. Folglich `Uploader` kann das Amazon S3 S3-Upload-Limit von 10.000 Teilen für große Dateien erreicht werden, wenn Sie es `PartSize` zu niedrig einstellen. Wenn Sie versuchen, mehr als 10.000 Teile hochzuladen, stoppt der Upload und es wird ein Fehler zurückgegeben.

Bei `body` Werten, die den `ReadSeekerAt` Typ implementieren, `Uploader` puffert der den Hauptinhalt nicht im Speicher, bevor er an Amazon S3 gesendet wird. `Uploader`berechnet die erwartete Anzahl von Teilen, bevor die Datei auf Amazon S3 hochgeladen wird. Wenn der aktuelle Wert von mehr als 10.000 Teile zum Hochladen der Datei `PartSize` erfordert, wird der Wert für die Bauteilgröße `Uploader` erhöht, sodass weniger Teile benötigt werden.

Umgang mit fehlgeschlagenen Uploads

Wenn ein Upload zu Amazon S3 fehlschlägt, `Uploader` verwendet standardmäßig den Amazon S3 `AbortMultipartUpload` S3-Vorgang, um die hochgeladenen Teile zu entfernen. Diese Funktion stellt sicher, dass fehlgeschlagene Uploads keinen Amazon S3 S3-Speicherplatz verbrauchen.

Sie können `LeavePartsOnError` den Wert auf true setzen, damit die erfolgreich hochgeladenen Teile `Uploader` nicht gelöscht werden. Dies ist nützlich, um teilweise abgeschlossene Uploads wieder aufzunehmen. Um mit hochgeladenen Teilen arbeiten zu können, müssen Sie die Informationen über den `UploadID` fehlgeschlagenen Upload erhalten. Das folgende Beispiel zeigt, wie Sie den Schnittstellentyp `manager.MultiUploadFailure` Fehler verwenden, um den zu ermittelnUploadID.

```
result, err := uploader.Upload(context.TODO(), &s3.PutObjectInput{
    Bucket: aws.String("amzn-s3-demo-bucket"),
    Key:    aws.String("my-object-key"),
    Body:   uploadFile,
})
output, err := u.upload(input)
if err != nil {
    var mu manager.MultiUploadFailure
    if errors.As(err, &mu) {
        // Process error and its associated uploadID
        fmt.Println("Error:", mu)
        _ = mu.UploadID() // retrieve the associated UploadID
    } else {
        // Process error generically
        fmt.Println("Error:", err.Error())
    }
    return
}
```

Uploader-Optionen pro Upload überschreiben

Sie können die `Uploader` Optionen beim Aufrufen überschreiben, `Upload` indem Sie der Methode ein oder mehrere Argumente zur Verfügung stellen. Bei diesen Überschreibungen handelt es sich um Änderungen, bei denen die Parallelität gewährleistet ist. Sie wirken sich nicht auf laufende Uploads oder nachfolgende Aufrufe an den Manager aus. `Upload` Um beispielsweise die `PartSize` Konfiguration für eine bestimmte Upload-Anfrage zu überschreiben:

```
params := &s3.PutObjectInput{
```

```
    Bucket: aws.String("amzn-s3-demo-bucket"),
    Key:    aws.String("my-key"),
    Body:   myBody,
}
resp, err := uploader.Upload(context.TODO(), params, func(u *manager.Uploader) {
    u.PartSize = 10 * 1024 * 1024, // 10 MiB
})
```

Beispiele

Einen Ordner auf Amazon S3 hochladen

Das folgende Beispiel verwendet das `path/filepath` Paket, um rekursiv eine Liste von Dateien zu sammeln und sie in den angegebenen Amazon S3 S3-Bucket hochzuladen. Den Schlüsseln der Amazon S3 S3-Objekte wird der relative Pfad der Datei vorangestellt.

```
package main

import (
    "context"
    "log"
    "os"
    "path/filepath"

    "github.com/aws/aws-sdk-go-v2/aws"
    "github.com/aws/aws-sdk-go-v2/config"
    "github.com/aws/aws-sdk-go-v2/feature/s3/manager"
    "github.com/aws/aws-sdk-go-v2/service/s3"
)

var (
    localPath string
    bucket    string
    prefix    string
)

func init() {
    if len(os.Args) != 4 {
        log.Fatalln("Usage:", os.Args[0], "<local path> <bucket> <prefix>")
    }
    localPath = os.Args[1]
    bucket = os.Args[2]
    prefix = os.Args[3]
```

```
}

func main() {
    walker := make(fileWalk)
    go func() {
        // Gather the files to upload by walking the path recursively
        if err := filepath.Walk(localPath, walker.Walk); err != nil {
            log.Fatalln("Walk failed:", err)
        }
        close(walker)
    }()

    cfg, err := config.LoadDefaultConfig(context.TODO())
    if err != nil {
        log.Fatalln("error:", err)
    }

    // For each file found walking, upload it to Amazon S3
    uploader := manager.NewUploader(s3.NewFromConfig(cfg))
    for path := range walker {
        rel, err := filepath.Rel(localPath, path)
        if err != nil {
            log.Fatalln("Unable to get relative path:", path, err)
        }
        file, err := os.Open(path)
        if err != nil {
            log.Println("Failed opening file", path, err)
            continue
        }
        defer file.Close()
        result, err := uploader.Upload(context.TODO(), &s3.PutObjectInput{
            Bucket: &bucket,
            Key:    aws.String(filepath.Join(prefix, rel)),
            Body:   file,
        })
        if err != nil {
            log.Fatalln("Failed to upload", path, err)
        }
        log.Println("Uploaded", path, result.Location)
    }
}

type fileWalk chan string
```

```
func (f fileWalk) Walk(path string, info os.FileInfo, err error) error {
    if err != nil {
        return err
    }
    if !info.IsDir() {
        f <- path
    }
    return nil
}
```

Laden Sie den Manager herunter

Der Amazon S3 Downloader-Manager bestimmt, ob eine Datei in kleinere Teile aufgeteilt und parallel heruntergeladen werden kann. Sie können die Anzahl der parallel Downloads und die Größe der heruntergeladenen Teile anpassen.

Beispiel: Laden Sie eine Datei herunter

Im folgenden Beispiel wird Amazon S3 verwendet`Downloader`, um eine Datei herunterzuladen. Die Verwendung `Downloader` ähnelt der von S3. GetObjectBetrieb.

```
import "context"
import "github.com/aws/aws-sdk-go-v2/aws"
import "github.com/aws/aws-sdk-go-v2/config"
import "github.com/aws/aws-sdk-go-v2/service/s3"
import "github.com/aws/aws-sdk-go-v2/feature/s3/manager"

// ...

cfg, err := config.LoadDefaultConfig(context.TODO())
if err != nil {
    log.Println("error:", err)
    return
}

client := s3.NewFromConfig(cfg)

downloader := manager.NewDownloader(client)
numBytes, err := downloader.Download(context.TODO(), downloadFile, &s3.GetObjectInput{
    Bucket: aws.String("amzn-s3-demo-bucket"),
    Key:    aws.String("my-key"),
})
```

Der `downloadFile` Parameter ist ein `io.WriterAt` Typ. Die `WriterAt` Schnittstelle ermöglicht es`Downloader`, mehrere Teile der Datei parallel zu schreiben.

Konfigurationsoptionen

Wenn Sie eine `Downloader` Instanz instanziieren, können Sie Konfigurationsoptionen angeben, um anzupassen, wie Objekte heruntergeladen werden:

- `PartSize`— Gibt die Puffergröße jedes herunterzuladenden Teils in Byte an. Die Mindestgröße pro Teil beträgt 5 MB.
- `Concurrency`— Gibt die Anzahl der Teile an, die parallel heruntergeladen werden sollen.

Der `Concurrency` Wert begrenzt die Anzahl der gleichzeitigen Downloads von Teilen, die bei einem bestimmten `Download` Anruf ausgeführt werden können. Dies ist kein globales Limit für die Parallelität von Clients. Passen Sie die Werte `PartSize` und die `Concurrency` Konfiguration an, um die optimale Konfiguration zu finden. Beispielsweise können Systeme mit Verbindungen mit hoher Bandbreite größere Teile und mehr Downloads parallel empfangen.

Ihre Anwendung wird beispielsweise `Downloader` mit einem `Concurrency` von konfiguriert. `5` Ihre Anwendung ruft dann `Download` von zwei verschiedenen Goroutinen aus auf. Das Ergebnis sind `10` gleichzeitige Downloads von Teilen (2 Goroutinen * 5). `Concurrency`

> **Warning**
>
> Es wird erwartet, dass Ihre Anwendung die Anzahl gleichzeitiger Aufrufe begrenzt, um eine Erschöpfung der Anwendungsressourcen zu verhindern`Download`.

Weitere Informationen zu `Downloader` und den anderen Konfigurationsoptionen finden Sie unter Manager.Downloader in der API-Referenz. AWS SDK für Go

Überschreiben der Downloader-Optionen pro Download

Sie können die `Downloader` Optionen beim Aufrufen überschreiben, `Download` indem Sie der Methode ein oder mehrere funktionale Argumente zur Verfügung stellen. Bei diesen Überschreibungen handelt es sich um Änderungen, die gleichzeitig funktionieren und sich nicht auf laufende Uploads oder nachfolgende `Download` Aufrufe an den Manager auswirken. Um beispielsweise die `PartSize` Konfiguration für eine bestimmte Upload-Anfrage zu überschreiben:

```
params := &s3.GetObjectInput{
```

```
    Bucket: aws.String("amzn-s3-demo-bucket"),
    Key:    aws.String("my-key"),
}
resp, err := downloader.Download(context.TODO(), targetWriter, params, func(u
 *manager.Downloader) {
    u.PartSize = 10 * 1024 * 1024, // 10 MiB
})
```

Beispiele

Alle Objekte in einem Bucket herunterladen

Das folgende Beispiel verwendet Paginierung, um eine Liste von Objekten aus einem Amazon S3 S3-Bucket zu sammeln. Anschließend wird jedes Objekt in eine lokale Datei heruntergeladen.

```
package main

import (
    "context"
    "fmt"
    "log"
    "os"
    "path/filepath"

    "github.com/aws/aws-sdk-go-v2/aws"
    "github.com/aws/aws-sdk-go-v2/config"
    "github.com/aws/aws-sdk-go-v2/feature/s3/manager"
    "github.com/aws/aws-sdk-go-v2/service/s3"
)

var (
    Bucket         = "amzn-s3-demo-bucket" // Download from this bucket
    Prefix         = "logs/"    // Using this key prefix
    LocalDirectory = "s3logs"   // Into this directory
)

func main() {
    cfg, err := config.LoadDefaultConfig(context.TODO())
    if err != nil {
        log.Fatalln("error:", err)
    }

    client := s3.NewFromConfig(cfg)
    manager := manager.NewDownloader(client)
```

```
    paginator := s3.NewListObjectsV2Paginator(client, &s3.ListObjectsV2Input{
        Bucket: &Bucket,
        Prefix: &Prefix,
    })

    for paginator.HasMorePages() {
        page, err := paginator.NextPage(context.TODO())
        if err != nil {
            log.Fatalln("error:", err)
        }
        for _, obj := range page.Contents {
            if err := downloadToFile(manager, LocalDirectory, Bucket,
 aws.ToString(obj.Key)); err != nil {
                log.Fatalln("error:", err)
            }
        }
    }
}

func downloadToFile(downloader *manager.Downloader, targetDirectory, bucket, key
 string) error {
    // Create the directories in the path
    file := filepath.Join(targetDirectory, key)
    if err := os.MkdirAll(filepath.Dir(file), 0775); err != nil {
        return err
    }

    // Set up the local file
    fd, err := os.Create(file)
    if err != nil {
        return err
    }
    defer fd.Close()

    // Download the file using the AWS SDK for Go
    fmt.Printf("Downloading s3://%s/%s to %s...\n", bucket, key, file)
    _, err = downloader.Download(context.TODO(), fd, &s3.GetObjectInput{Bucket:
 &bucket, Key: &key})

    return err
}
```

GetBucketRegion

GetBucketRegionist eine Hilfsfunktion zur Bestimmung des Standorts der AWS Region eines Amazon S3 S3-Buckets. `GetBucketRegion`nimmt einen Amazon S3 S3-Client und verwendet ihn, um den Speicherort des angeforderten Buckets innerhalb der AWS Partition zu ermitteln, die der konfigurierten Region des Clients zugeordnet ist.

Um beispielsweise die Region für den Bucket zu finden*amzn-s3-demo-bucket*:

```
cfg, err := config.LoadDefaultConfig(context.TODO())
if err != nil {
    log.Println("error:", err)
    return
}

bucket := "amzn-s3-demo-bucket"
region, err := manager.GetBucketRegion(ctx, s3.NewFromConfig(cfg), bucket)
if err != nil {
    var bnf manager.BucketNotFound
    if errors.As(err, &bnf) {
        log.Printf("unable to find bucket %s's Region\n", bucket)
    } else {
        log.Println("error:", err)
    }
    return
}
fmt.Printf("Bucket %s is in %s region\n", bucket, region)
```

Wenn `GetBucketRegion` die Position eines Buckets nicht aufgelöst werden kann, gibt die Funktion einen BucketNotFoundFehlertyp zurück, wie im Beispiel gezeigt.

Streaming-Eingabe nicht auffindbar

Für API-Operationen wie `PutObject` und erwartet der Amazon S3 S3-Client`UploadPart`, dass der Wert des `Body` Eingabeparameters standardmäßig die io.Seeker-Schnittstelle implementiert. Die `io.Seeker` Schnittstelle wird vom Client verwendet, um die Länge des hochzuladenden Werts zu bestimmen und den Payload-Hash für die Anforderungssignatur zu berechnen. Wenn der `Body` Eingabeparameterwert nicht implementiert wird`io.Seeker`, erhält Ihre Anwendung einen Fehler.

```
operation error S3: PutObject, failed to compute payload hash: failed to seek
body to start, request stream is not seekable
```

Sie können dieses Verhalten ändern, indem Sie die Betriebsmethode Middleware mithilfe von Funktionsoptionen ändern. Der APIOptionsWith-Helper gibt eine funktionale Option für null oder mehr Middleware-Mutatoren zurück. Um den Client zu deaktivieren, der den Payload-Hash berechnet und die Signatur für unsignierte Payload-Anfragen verwendet, fügen Sie v4 hinzu. SwapComputePayloadSHA256ForUnsignedPayloadMiddleware.

```
resp, err := client.PutObject(context.TODO(), &s3.PutObjectInput{
    Bucket: &bucketName,
    Key: &objectName,
    Body: bytes.NewBuffer([]byte(`example object!`)),
    ContentLength: 15, // length of body
}, s3.WithAPIOptions(
    v4.SwapComputePayloadSHA256ForUnsignedPayloadMiddleware,
))
```

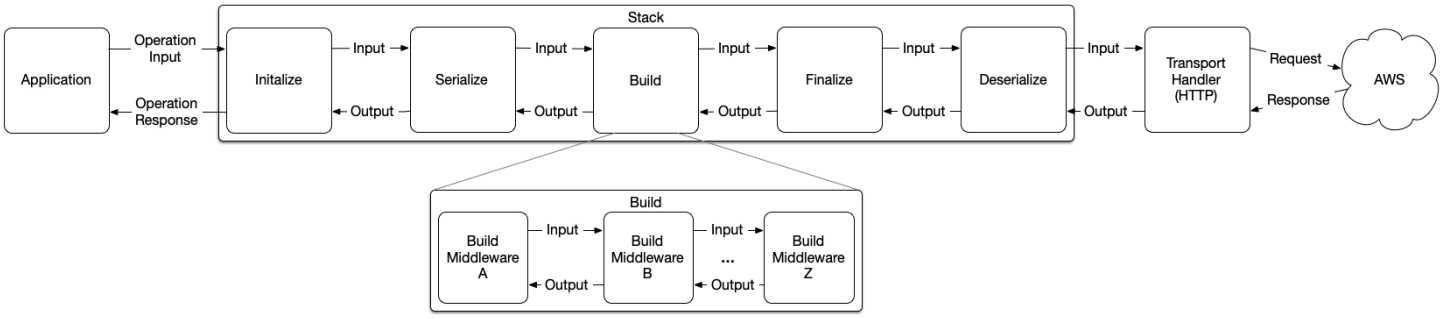
Warning

Amazon S3 erfordert, dass die Inhaltslänge für alle Objekte angegeben wird, die in einen Bucket hochgeladen wurden. Da der `Body` Eingabeparameter keine `io.Seeker` Schnittstelle implementiert, kann der Client den `ContentLength` Parameter für die Anfrage nicht berechnen. Der Parameter muss von der Anwendung bereitgestellt werden. Die Anfrage schlägt fehl, wenn der `ContentLength` Parameter nicht angegeben wird.
Verwenden Sie die SDKs Amazon S3 S3-Uploadmanager für Uploads, die nicht durchsuchbar sind und deren Länge nicht bekannt ist.

Anpassen der AWS SDK für Go v2-Client-Anfragen mit Middleware

> **Warning**
>
> Das Ändern der Client-Anforderungspipeline kann zu fehlerhaften/ungültigen Anfragen oder zu unerwarteten Anwendungsfehlern führen. Diese Funktionalität ist für erweiterte Anwendungsfälle vorgesehen, die nicht standardmäßig von der SDK-Schnittstelle bereitgestellt werden.

Sie können AWS SDK für Go Client-Anfragen anpassen, indem Sie eine oder mehrere Middleware im Stack eines Dienstvorgangs registrieren. Der Stack besteht aus einer Reihe von Schritten: Initialisieren, Serialisieren, Erstellen, Finalisieren und Deserialisieren. Jeder Schritt enthält null oder mehr Middleware, die mit den Eingabe- und Ausgabetypen dieses Schritts arbeitet. Das folgende Diagramm und die folgende Tabelle bieten einen Überblick darüber, wie die Anforderung und Antwort einer Operation den Stack durchläuft.



Schritt „Stapel“	Beschreibung
Initialisieren	Bereitet die Eingabe vor und legt alle Standardparameter nach Bedarf fest.
Serialisieren	Serialisiert die Eingabe in ein Protokollformat, das für die Zieltransportschicht geeignet ist.
Entwicklung	Hängt zusätzliche Metadaten an die serialisierte Eingabe an, z. B. HTTP Content-Length.

Schritt „Stapel“	Beschreibung
Finalisieren	Endgültige Nachrichtenvorbereitung, einschließlich Wiederholungsversuche und Authentifizierung (SigV4-Signatur).
Deserialisieren	Deserialisieren Sie Antworten aus dem Protokollformat in einen strukturierten Typ oder Fehler.

Jede Middleware innerhalb eines bestimmten Schritts muss über eine eindeutige Kennung verfügen, die durch die Methode der Middleware bestimmt wird. `ID` Middleware-Identifikatoren stellen sicher, dass nur eine Instanz einer bestimmten Middleware für einen Schritt registriert ist, und ermöglichen das Einfügen anderer Step-Middleware relativ zu diesem Schritt.

Sie hängen Step-Middleware an, indem Sie die Methoden eines Schritts oder einer Methode verwenden. `Insert` `Add` Sie fügen eine Middleware `Add` an den Anfang eines Schritts an, indem Sie `Middleware.before` als und `Middleware.After` angeben `RelativePosition`, um sie am Ende des Schritts anzufügen. Sie verwenden dies, um eine Middleware an einen `Insert` Schritt anzuhängen, indem Sie die Middleware relativ zu einer anderen Step-Middleware einfügen.

> **Warning**
>
> Sie müssen die `Add` Methode verwenden, um benutzerdefinierte Step-Middleware sicher einzufügen. Durch die Verwendung `Insert` entsteht eine Abhängigkeit zwischen Ihrer benutzerdefinierten Middleware und der Middleware, zu der Sie relativ etwas einfügen. Die Middleware innerhalb eines Stack-Schritts muss als undurchsichtig betrachtet werden, um zu verhindern, dass an Ihrer Anwendung schädliche Änderungen vorgenommen werden.

Eine benutzerdefinierte Middleware schreiben

Jeder Stack-Schritt hat eine Schnittstelle, die Sie erfüllen müssen, um eine Middleware an einen bestimmten Schritt anzuhängen. Sie können eine der bereitgestellten *`Step`*`MiddlewareFunc` Funktionen verwenden, um diese Schnittstelle schnell zu erfüllen. In der folgenden Tabelle werden die Schritte, ihre Schnittstelle und die Hilfsfunktion beschrieben, die verwendet werden kann, um die Anforderungen der Schnittstelle zu erfüllen.

Schritt	Schnittstelle	Helper-Funktion
Initialisieren	InitializeMiddleware	InitializeMiddlewareFunc
Entwicklung	BuildMiddleware	BuildMiddlewareFunc
Serialisieren	SerializeMiddleware	SerializeMiddlewareFunc
Finalisieren	FinalizeMiddleware	FinalizeMiddlewareFunc
Deserialisieren	DeserializeMiddleware	DeserializeMiddlewareFunc

Die folgenden Beispiele zeigen, wie Sie eine benutzerdefinierte Middleware schreiben können, um das Bucket-Mitglied der Amazon S3 `GetObject` S3-API-Aufrufe zu füllen, falls keiner bereitgestellt wird. In den nachfolgenden Beispielen wird auf diese Middleware verwiesen, um zu zeigen, wie Step-Middleware an den Stack angehängt wird.

```
import "github.com/aws/smithy-go/aws"
import "github.com/aws/smithy-go/middleware"
import "github.com/aws/aws-sdk-go-v2/service/s3"

// ...

var defaultBucket = middleware.InitializeMiddlewareFunc("DefaultBucket", func(
    ctx context.Context, in middleware.InitializeInput, next
 middleware.InitializeHandler,
) (
    out middleware.InitializeOutput, metadata middleware.Metadata, err error,
) {
    // Type switch to check if the input is s3.GetObjectInput, if so and the bucket is
 not set, populate it with
    // our default.
    switch v := in.Parameters.(type) {
    case *s3.GetObjectInput:
        if v.Bucket == nil {
            v.Bucket = aws.String("amzn-s3-demo-bucket")
        }
    }

    // Middleware must call the next middleware to be executed in order to continue
 execution of the stack.
```

```
    // If an error occurs, you can return to prevent further execution.
    return next.HandleInitialize(ctx, in)
})
```

Middleware an alle Clients anhängen

Sie können Ihre benutzerdefinierte Step-Middleware an jeden Client anhängen, indem Sie die Middleware mithilfe eines Mitglieds des Typs AWS.config hinzufügen. `APIOptions` In den folgenden Beispielen wird die `defaultBucket` Middleware an jeden Client angehängt, der mit Ihrem Anwendungsobjekt erstellt wurde: `aws.Config`

```
import "context"
import "github.com/aws/aws-sdk-go-v2/aws"
import "github.com/aws/aws-sdk-go-v2/config"
import "github.com/aws/aws-sdk-go-v2/service/s3"
import "github.com/aws/smithy-go/middleware"

// ...

cfg, err := config.LoadDefaultConfig(context.TODO())
if err != nil {
    // handle error
}

cfg.APIOptions = append(cfg.APIOptions, func(stack *middleware.Stack) error {
    // Attach the custom middleware to the beginning of the Initialize step
    return stack.Initialize.Add(defaultBucket, middleware.Before)
})

client := s3.NewFromConfig(cfg)
```

Middleware an einen bestimmten Vorgang anhängen

Sie können Ihre benutzerdefinierte Step-Middleware an einen bestimmten Client-Vorgang anhängen, indem Sie das `APIOptions` Mitglied des Clients mithilfe der variadischen Argumentliste für einen Vorgang ändern. In den folgenden Beispielen wird die `defaultBucket` Middleware an einen bestimmten Amazon S3 `GetObject` S3-Operationsaufruf angehängt:

```
import "context"
import "github.com/aws/aws-sdk-go-v2/aws"
```

```
import "github.com/aws/aws-sdk-go-v2/config"
import "github.com/aws/aws-sdk-go-v2/service/s3"
import "github.com/aws/smithy-go/middleware"

// ...

// registerDefaultBucketMiddleware registers the defaultBucket middleware with the
 provided stack.
func registerDefaultBucketMiddleware(stack *middleware.Stack) error {
    // Attach the custom middleware to the beginning of the Initialize step
    return stack.Initialize.Add(defaultBucket, middleware.Before)
}

// ...

cfg, err := config.LoadDefaultConfig(context.TODO())
if err != nil {
    // handle error
}

client := s3.NewFromConfig(cfg)

object, err := client.GetObject(context.TODO(), &s3.GetObjectInput{
    Key: aws.String("my-key"),
}, func(options *s3.Options) {
    // Register the defaultBucketMiddleware for this operation only
    options.APIOptions = append(options.APIOptions, registerDefaultBucketMiddleware)
})
```

Metadaten im Stapel weiterleiten

In bestimmten Situationen stellen Sie möglicherweise fest, dass Sie zwei oder mehr Middlewares benötigen, um gemeinsam Informationen oder Status auszutauschen. Sie können Context.Context verwenden, um diese Metadaten mithilfe von Middleware zu übergeben. WithStackValue. `middleware.WithStackValue`hängt das angegebene Schlüssel-Wert-Paar an den angegebenen Kontext an und schränkt den Bereich sicher auf den aktuell ausgeführten Stack ein. Diese Werte im Stack-Bereich können mithilfe von Middleware aus einem Kontext abgerufen werden. GetStackValueund Bereitstellung des Schlüssels, der zum Speichern des entsprechenden Werts verwendet wurde. Schlüssel müssen vergleichbar sein, und Sie müssen Ihre eigenen Typen als Kontextschlüssel definieren, um Kollisionen zu vermeiden. Die folgenden Beispiele zeigen, wie zwei Middlewares Informationen über `context.Context` den Stack weiterleiten können.

```
import "context"
import "github.com/aws/smithy-go/middleware"

// ...

type customKey struct {}

func GetCustomKey(ctx context.Context) (v string) {
    v, _ = middleware.GetStackValue(ctx, customKey{}).(string)
    return v
}

func SetCustomKey(ctx context.Context, value string) context.Context {
    return middleware.WithStackValue(ctx, customKey{}, value)
}

// ...

var customInitalize = middleware.InitializeMiddlewareFunc("customInitialize", func(
    ctx context.Context, in middleware.InitializeInput, next
 middleware.InitializeHandler,
) (
    out middleware.InitializeOutput, metadata middleware.Metadata, err error,
) {
    ctx = SetCustomKey(ctx, "my-custom-value")

    return next.HandleInitialize(ctx, in)
})

var customBuild = middleware.BuildMiddlewareFunc("customBuild", func(
    ctx context.Context, in middleware.BuildInput, next middleware.BuildHandler,
) (
    out middleware.BuildOutput, metadata middleware.Metadata, err error,
) {
    customValue := GetCustomKey(ctx)

    // use customValue

    return next.HandleBuild(ctx, in)
})
```

Vom SDK bereitgestellte Metadaten

Das AWS SDK für Go stellt mehrere Metadatenwerte bereit, die aus dem bereitgestellten Kontext abgerufen werden können. Diese Werte können verwendet werden, um eine dynamischere Middleware zu aktivieren, die ihr Verhalten je nach ausgeführtem Dienst, Vorgang oder Zielregion ändert. Einige der verfügbaren Schlüssel sind in der folgenden Tabelle aufgeführt:

Schlüssel	Retriever	Beschreibung
Dienst-ID	GetServiceID (ID)	Ruft die Dienst-ID für den ausführenden Stack ab. Dies kann mit der `ServiceID` Konstante des Service-Client-Pakets verglichen werden.
OperationName	GetOperationName	Ruft den Operationsnamen für den ausführenden Stack ab.
Logger	GetLogger	Ruft den Logger ab, der für die Protokollierung von Nachrichten aus der Middleware verwendet werden kann.

Metadaten im Stapel nach oben weiterleiten

Sie können Metadaten durch den Stack weiterleiten, indem Sie Metadaten-Schlüssel- und Wertepaare mithilfe von Middleware.metadata hinzufügen. Jeder Middleware-Schritt gibt eine Ausgabestruktur, Metadaten und einen Fehler zurück. Ihre benutzerdefinierte Middleware muss die Metadaten zurückgeben, die Sie beim Aufrufen des nächsten Handlers im Schritt erhalten haben. Dadurch wird sichergestellt, dass von der Downstream-Middleware hinzugefügte Metadaten an die Anwendung weitergegeben werden, die den Dienstvorgang aufruft. Auf die resultierenden Metadaten kann die aufrufende Anwendung entweder über die Ausgabeform der Operation über das Strukturelement zugreifen. `ResultMetadata`

Die folgenden Beispiele zeigen, wie eine benutzerdefinierte Middleware Metadaten hinzufügen kann, die als Teil der Operationsausgabe zurückgegeben werden.

```
import "context"
import "github.com/aws/aws-sdk-go-v2/service/s3"
import "github.com/aws/smithy-go/middleware"

// ...

type customKey struct{}

func GetCustomKey(metadata middleware.Metadata) (v string) {
    v, _ = metadata.Get(customKey{}).(string)
    return v
}

func SetCustomKey(metadata *middleware.Metadata, value string) {
    metadata.Set(customKey{}, value)
}

// ...

var customInitalize = middleware.InitializeMiddlewareFunc("customInitialize", func (
    ctx context.Context, in middleware.InitializeInput, next
 middleware.InitializeHandler,
) (
    out middleware.InitializeOutput, metadata middleware.Metadata, err error,
) {
    out, metadata, err = next.HandleInitialize(ctx, in)
    if err != nil {
        return out, metadata, err
    }

    SetCustomKey(&metadata, "my-custom-value")

    return out, metadata, nil
})

// ...

client := s3.NewFromConfig(cfg, func (options *s3.Options) {
    options.APIOptions = append(options.APIOptions, func(stack *middleware.Stack) error
 {
        return stack.Initialize.Add(customInitalize, middleware.After)
    })
})
```

```
out, err := client.GetObject(context.TODO(), &s3.GetObjectInput{
    // input parameters
})
if err != nil {
    // handle error
}

customValue := GetCustomKey(out.ResponseMetadata)
```

Häufig gestellte Fragen

Wie konfiguriere ich den HTTP-Client meines SDK? Gibt es Richtlinien oder bewährte Verfahren?

Wir sind nicht in der Lage, Kunden zu beraten, wie sie ihren HTTP-Workflow so konfigurieren können, dass sie für ihren speziellen Workload am effektivsten sind. Die Antwort darauf ist das Produkt einer multivariaten Gleichung, zu deren Eingabefaktoren unter anderem gehören, aber nicht darauf beschränkt sind:

- die Netzwerkauslastung der Anwendung (TPS, Durchsatz usw.)
- die Dienste, die genutzt werden
- die Recheneigenschaften der Bereitstellung
- der geografische Charakter des Einsatzes
- das gewünschte Anwendungsverhalten oder die Anforderungen der Anwendung selbst (SLAs, Zeitpläne usw.)

Wie sollte ich Betriebs-Timeouts konfigurieren?

Ähnlich wie bei der vorherigen Frage kommt es darauf an. Zu den hier zu berücksichtigenden Elementen gehören die folgenden:

- Alle oben genannten Faktoren betreffen die HTTP-Client-Konfiguration
- Ihre eigenen Anwendungszeiten oder SLA-Einschränkungen (z. B. wenn Sie selbst Traffic für andere Verbraucher bereitstellen)

Die Antwort auf diese Frage sollte fast NIEMALS auf rein empirischer Beobachtung des Upstream-Verhaltens beruhen — z. B. „Ich habe 1000 Aufrufe für diesen Vorgang getätigt, er hat höchstens 5 Sekunden gedauert, also werde ich den Timeout auf dieser Grundlage mit einem Sicherheitsfaktor von 2 bis 10 Sekunden festlegen“. Die Umgebungsbedingungen können sich ändern, Dienste können sich vorübergehend verschlechtern, und solche Annahmen können sich ohne Vorwarnung als falsch erweisen.

Anfragen des SDK haben eine Zeitüberschreitung oder dauern zu lange. Wie behebe ich das?

Wir können Ihnen bei längeren Anrufen oder bei Betriebsunterbrechungen nicht weiterhelfen, da wir zu viel Zeit in der Leitung verbracht haben. „Wire Time“ ist im SDK wie folgt definiert:

- Für die `HTTPClient.Do()` Methode eines SDK-Clients aufgewendete Zeit
- Zeit, die in `Read()` s für einen HTTP-Antworttext verbracht wurde, der an den Aufrufer weitergeleitet wurde (z. B. `GetObject`

Wenn Sie aufgrund von Betriebslatenz oder Timeouts Probleme haben, sollten Sie zunächst Telemetriedaten des SDK-Betriebszyklus abrufen, um die zeitliche Trennung zwischen der auf der Leitung verbrachten Zeit und dem damit verbundenen Overhead des Vorgangs zu ermitteln. Weitere Informationen finden Sie im Leitfaden zum Timing von SDK-Vorgängen, der einen wiederverwendbaren Codeausschnitt enthält, mit dem Sie dies erreichen können.

Wie behebe ich einen **`read: connection reset`** Fehler?

Das SDK wiederholt standardmäßig alle Fehler, die dem `connection reset` Muster entsprechen. Dies deckt die Fehlerbehandlung für die meisten Operationen ab, bei denen die HTTP-Antwort der Operation vollständig verarbeitet und in ihren modellierten Ergebnistyp deserialisiert wird.

Dieser Fehler kann jedoch immer noch in einem Kontext außerhalb der Wiederholungsschleife auftreten: Bestimmte Dienstoperationen leiten den HTTP-Antworttext der API direkt an den Aufrufer weiter, sodass er direkt über die Leitung abgerufen wird `io.ReadCloser` (z. B. `GetObject` die Objektnutzlast). Dieser Fehler kann auftreten, wenn Sie eine Antwort `Read` auf den Antworttext ausführen.

Dieser Fehler weist darauf hin, dass Ihr Host, der Dienst oder eine zwischengeschaltete Partei (z. B. NAT-Gateways, Proxys, Load Balancer) die Verbindung geschlossen hat, während Sie versucht haben, die Antwort zu lesen.

Dies kann aus verschiedenen Gründen auftreten:

- Sie haben den Antworttext einige Zeit lang nicht verwendet, nachdem die Antwort selbst empfangen wurde (nachdem der Dienstvorgang aufgerufen wurde). Wir empfehlen Ihnen, den HTTP-Antworttext für diese Art von Vorgängen so schnell wie möglich zu verwenden.

- Sie haben einen zuvor empfangenen Antworttext nicht geschlossen. Dies kann auf bestimmten Plattformen zum Zurücksetzen der Verbindung führen. Sie MÜSSEN alle **`io.ReadCloser`** Instanzen schließen, die in der Antwort einer Operation angegeben werden, unabhängig davon, ob Sie deren Inhalt verwenden.

Versuchen Sie außerdem, eine `tcpdump` für eine betroffene Verbindung am Rand Ihres Netzwerks auszuführen (z. B. hinter allen Proxys, die Sie kontrollieren). Wenn Sie feststellen, dass der AWS Endpunkt anscheinend TCP-RST sendet, sollten Sie die AWS Support-Konsole verwenden, um ein Verfahren gegen den betreffenden Dienst einzuleiten. Seien Sie bereit, eine Anfrage IDs und genaue Zeitstempel anzugeben, wann das Problem aufgetreten ist.

Warum erhalte ich die Fehlermeldung „Ungültige Signatur“, wenn ich einen HTTP-Proxy mit dem SDK verwende?

Der Signaturalgorithmus für AWS Dienste (im Allgemeinen Sigv4) ist an die Header der serialisierten Anfrage gebunden, genauer gesagt an die meisten Header mit dem Präfix. `X-` Proxys neigen dazu, die ausgehende Anfrage zu modifizieren, indem sie zusätzliche Weiterleitungsinformationen hinzufügen (oft über einen `X-Forwarded-For` Header), wodurch die vom SDK berechnete Signatur effektiv gebrochen wird.

Wenn Sie einen HTTP-Proxy verwenden und Signaturfehler auftreten, sollten Sie versuchen, die Anfrage so zu erfassen, wie sie vom Proxy ausgeht, und feststellen, ob es sich um eine andere Anfrage handelt.

Timing von SDK-Vorgängen

Beim Debuggen von Timeout-/Latenzproblemen im SDK ist es wichtig, die Komponenten des Betriebslebenszyklus zu identifizieren, deren Ausführung länger dauert als erwartet. Als Ausgangspunkt müssen Sie in der Regel die zeitliche Verteilung zwischen dem gesamten Operationsaufruf und dem HTTP-Aufruf selbst überprüfen.

Das folgende Beispielprogramm implementiert eine grundlegende Instrumentierung im Hinblick auf `smithy-go` Middleware für SQS-Clients und demonstriert, wie sie verwendet wird. Die Probe gibt für jeden Operationsaufruf die folgenden Informationen aus:

- AWS ID anfordern
- Dienst-ID

- Name des Vorgangs
- Uhrzeit des Aufrufs der Operation
- Zeit des HTTP-Anrufs

Jeder ausgegebenen Nachricht wird eine eindeutige (für eine einzelne Operation) „Aufruf-ID“ vorangestellt, die am Anfang des Handler-Stacks gesetzt wird.

Der Einstiegspunkt für die Instrumentierung wird als bereitgestelltWithOperationTiming, der so parametrisiert ist, dass er eine Nachrichtenbehandlungsfunktion akzeptiert, die Instrumentierungs-„Ereignisse“ als formatierte Zeichenketten empfängt. PrintfMSGHandlerwird aus praktischen Gründen zur Verfügung gestellt, sodass Nachrichten einfach auf die Standardausgabe ausgegeben werden.

Der hier verwendete Dienst ist austauschbar - ALLE Service-Client-Optionen akzeptieren APIOptions und eine AS-Konfiguration. HTTPClient WithOperationTimingKönnte zum Beispiel stattdessen wie folgt deklariert werden:

```
func WithOperationTiming(msgHandler func(string)) func(*s3.Options)
func WithOperationTiming(msgHandler func(string)) func(*dynamodb.Options)
// etc.
```

Wenn Sie es ändern, stellen Sie sicher, dass Sie auch die Signatur der Funktion ändern, die zurückgegeben wird.

```
import (
    "context"
    "fmt"
    "log"
    "net/http"
    "sync"
    "time"

    awsmiddleware "github.com/aws/aws-sdk-go-v2/aws/middleware"
    awshttp "github.com/aws/aws-sdk-go-v2/aws/transport/http"
    "github.com/aws/aws-sdk-go-v2/config"
    "github.com/aws/aws-sdk-go-v2/service/sqs"
    "github.com/aws/smithy-go/middleware"
    smithyrand "github.com/aws/smithy-go/rand"
)

// WithOperationTiming instruments an SQS client to dump timing information for
```

```
// the following spans:
//   - overall operation time
//   - HTTPClient call time
//
// This instrumentation will also emit the request ID, service name, and
// operation name for each invocation.
//
// Accepts a message "handler" which is invoked with formatted messages to be
// handled externally, you can use the declared PrintfMSGHandler to simply dump
// these values to stdout.
func WithOperationTiming(msgHandler func(string)) func(*sqs.Options) {
    return func(o *sqs.Options) {
        o.APIOptions = append(o.APIOptions, addTimingMiddlewares(msgHandler))
        o.HTTPClient = &timedHTTPClient{
            client:     awshttp.NewBuildableClient(),
            msgHandler: msgHandler,
        }
    }
}

// PrintfMSGHandler writes messages to stdout.
func PrintfMSGHandler(msg string) {
    fmt.Printf("%s\n", msg)
}

type invokeIDKey struct{}

func setInvokeID(ctx context.Context, id string) context.Context {
    return middleware.WithStackValue(ctx, invokeIDKey{}, id)
}

func getInvokeID(ctx context.Context) string {
    id, _ := middleware.GetStackValue(ctx, invokeIDKey{}).(string)
    return id
}

// Records the current time, and returns a function to be called when the
// target span of events is completed. The return function will emit the given
// span name and time elapsed to the given message consumer.
func timeSpan(ctx context.Context, name string, consumer func(string)) func() {
    start := time.Now()
    return func() {
        elapsed := time.Now().Sub(start)
        consumer(fmt.Sprintf("[%s] %s: %s", getInvokeID(ctx), name, elapsed))
```

```
    }
}

type timedHTTPClient struct {
    client     *awshttp.BuildableClient
    msgHandler func(string)
}

func (c *timedHTTPClient) Do(r *http.Request) (*http.Response, error) {
    defer timeSpan(r.Context(), "http", c.msgHandler)()

    resp, err := c.client.Do(r)
    if err != nil {
        return nil, fmt.Errorf("inner client do: %v", err)
    }

    return resp, nil
}

type addInvokeIDMiddleware struct {
    msgHandler func(string)
}

func (*addInvokeIDMiddleware) ID() string { return "addInvokeID" }

func (*addInvokeIDMiddleware) HandleInitialize(ctx context.Context, in
 middleware.InitializeInput, next middleware.InitializeHandler) (
    out middleware.InitializeOutput, md middleware.Metadata, err error,
) {
    id, err := smithyrand.NewUUID(smithyrand.Reader).GetUUID()
    if err != nil {
        return out, md, fmt.Errorf("new uuid: %v", err)
    }

    return next.HandleInitialize(setInvokeID(ctx, id), in)
}

type timeOperationMiddleware struct {
    msgHandler func(string)
}

func (*timeOperationMiddleware) ID() string { return "timeOperation" }
```

```
func (m *timeOperationMiddleware) HandleInitialize(ctx context.Context, in
 middleware.InitializeInput, next middleware.InitializeHandler) (
    middleware.InitializeOutput, middleware.Metadata, error,
) {
    defer timeSpan(ctx, "operation", m.msgHandler)()
    return next.HandleInitialize(ctx, in)
}

type emitMetadataMiddleware struct {
    msgHandler func(string)
}

func (*emitMetadataMiddleware) ID() string { return "emitMetadata" }

func (m *emitMetadataMiddleware) HandleInitialize(ctx context.Context, in
 middleware.InitializeInput, next middleware.InitializeHandler) (
    middleware.InitializeOutput, middleware.Metadata, error,
) {
    out, md, err := next.HandleInitialize(ctx, in)

    invokeID := getInvokeID(ctx)
    requestID, _ := awsmiddleware.GetRequestIDMetadata(md)
    service := awsmiddleware.GetServiceID(ctx)
    operation := awsmiddleware.GetOperationName(ctx)
    m.msgHandler(fmt.Sprintf(`[%s] requestID = "%s"`, invokeID, requestID))
    m.msgHandler(fmt.Sprintf(`[%s] service   = "%s"`, invokeID, service))
    m.msgHandler(fmt.Sprintf(`[%s] operation = "%s"`, invokeID, operation))

    return out, md, err
}

func addTimingMiddlewares(mh func(string)) func(*middleware.Stack) error {
    return func(s *middleware.Stack) error {
        if err := s.Initialize.Add(&timeOperationMiddleware{msgHandler: mh},
 middleware.Before); err != nil {
            return fmt.Errorf("add time operation middleware: %v", err)
        }
        if err := s.Initialize.Add(&addInvokeIDMiddleware{msgHandler: mh},
 middleware.Before); err != nil {
            return fmt.Errorf("add invoke id middleware: %v", err)
        }
        if err := s.Initialize.Insert(&emitMetadataMiddleware{msgHandler: mh},
 "RegisterServiceMetadata", middleware.After); err != nil {
            return fmt.Errorf("add emit metadata middleware: %v", err)
```

```
        }
        return nil
    }
}

func main() {
    cfg, err := config.LoadDefaultConfig(context.Background())
    if err != nil {
        log.Fatal(fmt.Errorf("load default config: %v", err))
    }

    svc := sqs.NewFromConfig(cfg, WithOperationTiming(PrintfMSGHandler))

    var wg sync.WaitGroup

    for i := 0; i < 6; i++ {
        wg.Add(1)
        go func() {
            defer wg.Done()

            _, err = svc.ListQueues(context.Background(), nil)
            if err != nil {
                fmt.Println(fmt.Errorf("list queues: %v", err))
            }
        }()
    }
    wg.Wait()
}
```

Eine Beispielausgabe dieses Programms:

```
[e9a801bb-c51d-45c8-8e9f-a202e263fde8] http: 192.24067ms
[e9a801bb-c51d-45c8-8e9f-a202e263fde8] requestID = "dbee3082-96a3-5b23-
adca-6d005696fa94"
[e9a801bb-c51d-45c8-8e9f-a202e263fde8] service   = "SQS"
[e9a801bb-c51d-45c8-8e9f-a202e263fde8] operation = "ListQueues"
[e9a801bb-c51d-45c8-8e9f-a202e263fde8] operation: 193.098393ms
[0740f0e0-953e-4328-94fc-830a5052e763] http: 195.185732ms
[0740f0e0-953e-4328-94fc-830a5052e763] requestID = "48b301fa-
fc9f-5f1f-9007-5c783caa9322"
[0740f0e0-953e-4328-94fc-830a5052e763] service   = "SQS"
[0740f0e0-953e-4328-94fc-830a5052e763] operation = "ListQueues"
[0740f0e0-953e-4328-94fc-830a5052e763] operation: 195.725491ms
```

```
[c0589832-f351-4cc7-84f1-c656eb79dbd7] http: 200.52383ms
[444030d0-6743-4de5-bd91-bc40b2b94c55] http: 200.525919ms
[c0589832-f351-4cc7-84f1-c656eb79dbd7] requestID = "4a73cc82-b47b-56e1-
b327-9100744e1b1f"
[c0589832-f351-4cc7-84f1-c656eb79dbd7] service   = "SQS"
[c0589832-f351-4cc7-84f1-c656eb79dbd7] operation = "ListQueues"
[c0589832-f351-4cc7-84f1-c656eb79dbd7] operation: 201.214365ms
[444030d0-6743-4de5-bd91-bc40b2b94c55] requestID = "ca1523ed-1879-5610-
bf5d-7e6fd84cabee"
[444030d0-6743-4de5-bd91-bc40b2b94c55] service   = "SQS"
[444030d0-6743-4de5-bd91-bc40b2b94c55] operation = "ListQueues"
[444030d0-6743-4de5-bd91-bc40b2b94c55] operation: 201.197071ms
[079e8dbd-bb93-43ab-89e5-a7bb392b86a5] http: 206.449568ms
[12b2b39d-df86-4648-a436-ff0482d13340] http: 206.526603ms
[079e8dbd-bb93-43ab-89e5-a7bb392b86a5] requestID = "64229710-b552-56ed-8f96-
ca927567ec7b"
[079e8dbd-bb93-43ab-89e5-a7bb392b86a5] service   = "SQS"
[079e8dbd-bb93-43ab-89e5-a7bb392b86a5] operation = "ListQueues"
[079e8dbd-bb93-43ab-89e5-a7bb392b86a5] operation: 207.252357ms
[12b2b39d-df86-4648-a436-ff0482d13340] requestID =
 "76d9cbc0-07aa-58aa-98b7-9642c79f9851"
[12b2b39d-df86-4648-a436-ff0482d13340] service   = "SQS"
[12b2b39d-df86-4648-a436-ff0482d13340] operation = "ListQueues"
[12b2b39d-df86-4648-a436-ff0482d13340] operation: 207.360621ms
```

Unit-Tests mit dem AWS SDK für Go v2

Wenn Sie das SDK in Ihrer Anwendung verwenden, sollten Sie das SDK für den Komponententest Ihrer Anwendung nachahmen. Wenn Sie das SDK nachahmen, kann sich Ihr Test auf das konzentrieren, was Sie testen möchten, und nicht auf die internen Komponenten des SDK.

Um Mocking zu unterstützen, verwenden Sie Go-Benutzeroberflächen anstelle von konkreten Service-Client-, Paginatoren- und Kellnertypen, wie z. `s3.Client` Auf diese Weise kann Ihre Anwendung Muster wie Dependency Injection verwenden, um Ihre Anwendungslogik zu testen.

Verspottung von Client-Vorgängen

In diesem Beispiel `S3GetObjectAPI` handelt es sich um eine Schnittstelle, die die für die `GetObjectFromS3` Funktion erforderlichen Amazon S3 S3-API-Operationen definiert. `S3GetObjectAPI`ist mit der GetObjectMethode des Amazon S3 S3-Clients zufrieden.

```
import "context"
import "github.com/aws/aws-sdk-go-v2/service/s3"

// ...

type S3GetObjectAPI interface {
    GetObject(ctx context.Context, params *s3.GetObjectInput,
 optFns ...func(*s3.Options)) (*s3.GetObjectOutput, error)
}

func GetObjectFromS3(ctx context.Context, api S3GetObjectAPI, bucket, key string)
 ([]byte, error) {
    object, err := api.GetObject(ctx, &s3.GetObjectInput{
        Bucket: &bucket,
        Key:    &key,
    })
    if err != nil {
        return nil, err
    }
    defer object.Body.Close()

    return ioutil.ReadAll(object.Body)
}
```

Um die `GetObjectFromS3` Funktion zu testen, verwenden Sie den, `mockGetObjectAPI` um die `S3GetObjectAPI` Schnittstellendefinition zu erfüllen. Verwenden Sie dann den `mockGetObjectAPI` Typ, um die Ausgabe und die vom Service-Client zurückgegebenen Fehlerantworten nachzuahmen.

```
import "testing"
import "github.com/aws/aws-sdk-go-v2/service/s3"

// ...

type mockGetObjectAPI func(ctx context.Context, params *s3.GetObjectInput,
 optFns ...func(*s3.Options)) (*s3.GetObjectOutput, error)

func (m mockGetObjectAPI) GetObject(ctx context.Context, params *s3.GetObjectInput,
 optFns ...func(*s3.Options)) (*s3.GetObjectOutput, error) {
    return m(ctx, params, optFns...)
}

func TestGetObjectFromS3(t *testing.T) {
    cases := []struct {
        client func(t *testing.T) S3GetObjectAPI
        bucket string
        key string
        expect []byte
    }{
        {
            client: func(t *testing.T) S3GetObjectAPI {
                return mockGetObjectAPI(func(ctx context.Context, params
 *s3.GetObjectInput, optFns ...func(*s3.Options)) (*s3.GetObjectOutput, error) {
                    t.Helper()
                    if params.Bucket == nil {
                        t.Fatal("expect bucket to not be nil")
                    }
                    if e, a := "fooBucket", *params.Bucket; e != a {
                        t.Errorf("expect %v, got %v", e, a)
                    }
                    if params.Key == nil {
                        t.Fatal("expect key to not be nil")
                    }
                    if e, a := "barKey", *params.Key; e != a {
                        t.Errorf("expect %v, got %v", e, a)
                    }
```

```
                    return &s3.GetObjectOutput{
                        Body: ioutil.NopCloser(bytes.NewReader([]byte("this is the body
 foo bar baz"))),
                    }, nil
                })
            },
            bucket: "amzn-s3-demo-bucket>",
            key:    "barKey",
            expect: []byte("this is the body foo bar baz"),
        },
    }

    for i, tt := range cases {
        t.Run(strconv.Itoa(i), func(t *testing.T) {
            ctx := context.TODO()
            content, err := GetObjectFromS3(ctx, tt.client(t), tt.bucket, tt.key)
            if err != nil {
                t.Fatalf("expect no error, got %v", err)
            }
            if e, a := tt.expect, content; bytes.Compare(e, a) != 0 {
                t.Errorf("expect %v, got %v", e, a)
            }
        })
    }
}
```

Spöttische Paginatoren

Ähnlich wie Service-Clients können Paginatoren verspottet werden, indem eine Go-Schnittstelle für den Paginator definiert wird. Diese Schnittstelle würde vom Code Ihrer Anwendung verwendet werden. Auf diese Weise kann die SDK-Implementierung verwendet werden, wenn Ihre Anwendung ausgeführt wird, und eine simulierte Implementierung zum Testen.

Im folgenden Beispiel `ListObjectsV2Pager` ist eine Schnittstelle, die das für die Funktion erforderliche Verhalten für den Amazon S3 ListObjectsV2Paginator definiert. `CountObjects`

```
import "context"
import "github.com/aws/aws-sdk-go-v2/service/s3"

// ...

type ListObjectsV2Pager interface {
```

```
    HasMorePages() bool
    NextPage(context.Context, ...func(*s3.Options)) (*s3.ListObjectsV2Output, error)
}

func CountObjects(ctx context.Context, pager ListObjectsV2Pager) (count int, err error)
 {
    for pager.HasMorePages() {
        var output *s3.ListObjectsV2Output
        output, err = pager.NextPage(ctx)
        if err != nil {
            return count, err
        }
        count += int(output.KeyCount)
    }
    return count, nil
}
```

Erstellen Sie zum Testen den `mockListObjectsV2Pager` Typ`CountObjects`, der der Schnittstellendefinition entspricht. `ListObjectsV2Pager` Verwenden Sie dann`mockListObjectsV2Pager`, um das Paging-Verhalten der Ausgabe- und Fehlerantworten des Paginators für Dienstoperationen zu replizieren.

```
import "context"
import  "fmt"
import "testing"
import "github.com/aws/aws-sdk-go-v2/service/s3"

// ...

type mockListObjectsV2Pager struct {
    PageNum int
    Pages   []*s3.ListObjectsV2Output
}

func (m *mockListObjectsV2Pager) HasMorePages() bool {
    return m.PageNum < len(m.Pages)
}

func (m *mockListObjectsV2Pager) NextPage(ctx context.Context, f ...func(*s3.Options))
 (output *s3.ListObjectsV2Output, err error) {
    if m.PageNum >= len(m.Pages) {
        return nil, fmt.Errorf("no more pages")
    }
```

```
    output = m.Pages[m.PageNum]
    m.PageNum++
    return output, nil
}

func TestCountObjects(t *testing.T) {
    pager := &mockListObjectsV2Pager{
        Pages: []*s3.ListObjectsV2Output{
            {
                KeyCount: 5,
            },
            {
                KeyCount: 10,
            },
            {
                KeyCount: 15,
            },
        },
    }
    objects, err := CountObjects(context.TODO(), pager)
    if err != nil {
        t.Fatalf("expect no error, got %v", err)
    }
    if expect, actual := 30, objects; expect != actual {
        t.Errorf("expect %v, got %v", expect, actual)
    }
}
```

SDK for Go V2-Codebeispiele

Die Codebeispiele in diesem Thema zeigen Ihnen, wie Sie AWS SDK für Go V2 mit verwenden AWS.

Bei Grundlagen handelt es sich um Code-Beispiele, die Ihnen zeigen, wie Sie die wesentlichen Vorgänge innerhalb eines Services ausführen.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Service-Funktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarios anzeigen.

Szenarien sind Code-Beispiele, die Ihnen zeigen, wie Sie bestimmte Aufgaben ausführen, indem Sie mehrere Funktionen innerhalb eines Services aufrufen oder mit anderen AWS-Services kombinieren.

Einige Dienste enthalten zusätzliche Beispielkategorien, die zeigen, wie Sie Bibliotheken oder Funktionen nutzen können, die für den Dienst spezifisch sind.

Services

- API Gateway Gateway-Beispiele mit SDK for Go V2
- Aurora-Beispiele mit SDK for Go V2
- Beispiele für Amazon Bedrock mit SDK for Go V2
- Amazon Bedrock Runtime-Beispiele mit SDK for Go V2
- AWS CloudFormation Beispiele mit SDK for Go V2
- CloudWatch Log-Beispiele mit SDK for Go V2
- Beispiele für Amazon Cognito Identity Provider mit SDK for Go V2
- Amazon DocumentDB DocumentDB-Beispiele mit SDK for Go V2
- DynamoDB-Beispiele mit SDK for Go V2
- IAM-Beispiele mit SDK for Go V2
- Kinesis-Beispiele mit SDK for Go V2
- Lambda-Beispiele mit SDK for Go V2
- Amazon MSK-Beispiele mit SDK for Go V2
- Partner Central-Beispiele mit SDK for Go V2
- Amazon RDS-Beispiele mit SDK for Go V2
- Amazon Redshift Redshift-Beispiele mit SDK for Go V2

- Amazon S3 S3-Beispiele mit SDK for Go V2
- Amazon SNS SNS-Beispiele mit SDK for Go V2
- Amazon SQS SQS-Beispiele mit SDK for Go V2

API Gateway Gateway-Beispiele mit SDK for Go V2

Die folgenden Codebeispiele zeigen Ihnen, wie Sie mithilfe von AWS SDK für Go V2 mit API Gateway Aktionen ausführen und allgemeine Szenarien implementieren.

AWS Community-Beiträge sind Beispiele, die von mehreren Teams erstellt wurden und verwaltet werden AWS. Um Feedback zu geben, verwende den Mechanismus, der in den verlinkten Repositorien zur Verfügung steht.

Jedes Beispiel enthält einen Link zum vollständigen Quellcode, in dem Sie Anweisungen zum Einrichten und Ausführen des Codes im Kontext finden.

Themen

- AWS Beiträge der Community

AWS Beiträge der Community

Erstellen und testen Sie eine serverlose Anwendung

Das folgende Codebeispiel zeigt, wie eine serverlose Anwendung mithilfe von API Gateway mit Lambda und DynamoDB erstellt und getestet wird.

SDK für Go V2

> Zeigt, wie eine serverlose Anwendung, die aus einem API Gateway mit Lambda und DynamoDB besteht, mithilfe des Go SDK erstellt und getestet wird.
>
> Den vollständigen Quellcode und Anweisungen zur Einrichtung und Ausführung finden Sie im vollständigen Beispiel unter. GitHub
>
> In diesem Beispiel verwendete Dienste
>
> - API Gateway
> - DynamoDB
> - Lambda

Aurora-Beispiele mit SDK for Go V2

Die folgenden Codebeispiele zeigen Ihnen, wie Sie mithilfe von AWS SDK für Go V2 mit Aurora Aktionen ausführen und allgemeine Szenarien implementieren.

Bei Grundlagen handelt es sich um Code-Beispiele, die Ihnen zeigen, wie Sie die wesentlichen Vorgänge innerhalb eines Services ausführen.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Service-Funktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarios anzeigen.

Jedes Beispiel enthält einen Link zum vollständigen Quellcode, in dem Sie Anweisungen zum Einrichten und Ausführen des Codes im Kontext finden.

Erste Schritte

Hello Aurora

Die folgenden Codebeispiele veranschaulichen die ersten Schritte mit Aurora.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
package main

import (
 "context"
 "fmt"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/config"
 "github.com/aws/aws-sdk-go-v2/service/rds"
)
```

```
// main uses the AWS SDK for Go V2 to create an Amazon Aurora client and list up to
 20
// DB clusters in your account.
// This example uses the default settings specified in your shared credentials
// and config files.
func main() {
 ctx := context.Background()
 sdkConfig, err := config.LoadDefaultConfig(ctx)
 if err != nil {
  fmt.Println("Couldn't load default configuration. Have you set up your AWS
 account?")
  fmt.Println(err)
  return
 }
 auroraClient := rds.NewFromConfig(sdkConfig)
 const maxClusters = 20
 fmt.Printf("Let's list up to %v DB clusters.\n", maxClusters)
 output, err := auroraClient.DescribeDBClusters(
  ctx, &rds.DescribeDBClustersInput{MaxRecords: aws.Int32(maxClusters)})
 if err != nil {
  fmt.Printf("Couldn't list DB clusters: %v\n", err)
  return
 }
 if len(output.DBClusters) == 0 {
  fmt.Println("No DB clusters found.")
 } else {
  for _, cluster := range output.DBClusters {
   fmt.Printf("DB cluster %v has database %v.\n", *cluster.DBClusterIdentifier,
    *cluster.DatabaseName)
  }
 }
}
```

- Einzelheiten zur API finden Sie unter Describe DBClusters in der AWS SDK für Go API-Referenz.

Themen

- Grundlagen
- Aktionen

Grundlagen

Erlernen der Grundlagen

Wie das aussehen kann, sehen Sie am nachfolgenden Beispielcode:

- Erstellen Sie eine benutzerdefinierte Aurora-DB-Cluster-Parametergruppe und legen Sie Parameterwerte fest.
- Erstellen Sie einen DB-Cluster, der die Parametergruppe verwendet.
- Erstellen Sie eine DB-Instance, die eine Datenbank enthält.
- Erstellen Sie einen Snapshot des DB-Clusters und bereinigen Sie dann die Ressourcen.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Führen Sie ein interaktives Szenario an einem Prompt aus.

```
import (
 "aurora/actions"
 "context"
 "fmt"
 "log"
 "slices"
 "sort"
 "strconv"
 "strings"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/rds"
 "github.com/aws/aws-sdk-go-v2/service/rds/types"
 "github.com/awsdocs/aws-doc-sdk-examples/gov2/demotools"
 "github.com/google/uuid"
)
```

```
// GetStartedClusters is an interactive example that shows you how to use the AWS
 SDK for Go
// with Amazon Aurora to do the following:
//
// 1. Create a custom DB cluster parameter group and set parameter values.
// 2. Create an Aurora DB cluster that is configured to use the parameter group.
// 3. Create a DB instance in the DB cluster that contains a database.
// 4. Take a snapshot of the DB cluster.
// 5. Delete the DB instance, DB cluster, and parameter group.
type GetStartedClusters struct {
 sdkConfig  aws.Config
 dbClusters actions.DbClusters
 questioner demotools.IQuestioner
 helper     IScenarioHelper
 isTestRun  bool
}

// NewGetStartedClusters constructs a GetStartedClusters instance from a
 configuration.
// It uses the specified config to get an Amazon Relational Database Service (Amazon
 RDS)
// client and create wrappers for the actions used in the scenario.
func NewGetStartedClusters(sdkConfig aws.Config, questioner demotools.IQuestioner,
 helper IScenarioHelper) GetStartedClusters {
 auroraClient := rds.NewFromConfig(sdkConfig)
 return GetStartedClusters{
  sdkConfig:  sdkConfig,
  dbClusters: actions.DbClusters{AuroraClient: auroraClient},
  questioner: questioner,
  helper:     helper,
 }
}

// Run runs the interactive scenario.
func (scenario GetStartedClusters) Run(ctx context.Context, dbEngine string,
 parameterGroupName string,
 clusterName string, dbName string) {
 defer func() {
  if r := recover(); r != nil {
   log.Println("Something went wrong with the demo.")
  }
 }()
```

```
 log.Println(strings.Repeat("-", 88))
 log.Println("Welcome to the Amazon Aurora DB Cluster demo.")
 log.Println(strings.Repeat("-", 88))

 parameterGroup := scenario.CreateParameterGroup(ctx, dbEngine, parameterGroupName)
 scenario.SetUserParameters(ctx, parameterGroupName)
 cluster := scenario.CreateCluster(ctx, clusterName, dbEngine, dbName,
 parameterGroup)
 scenario.helper.Pause(5)
 dbInstance := scenario.CreateInstance(ctx, cluster)
 scenario.DisplayConnection(cluster)
 scenario.CreateSnapshot(ctx, clusterName)
 scenario.Cleanup(ctx, dbInstance, cluster, parameterGroup)

 log.Println(strings.Repeat("-", 88))
 log.Println("Thanks for watching!")
 log.Println(strings.Repeat("-", 88))
}

// CreateParameterGroup shows how to get available engine versions for a specified
// database engine and create a DB cluster parameter group that is compatible with a
// selected engine family.
func (scenario GetStartedClusters) CreateParameterGroup(ctx context.Context,
 dbEngine string,
 parameterGroupName string) *types.DBClusterParameterGroup {

 log.Printf("Checking for an existing DB cluster parameter group named %v.\n",
  parameterGroupName)
 parameterGroup, err := scenario.dbClusters.GetParameterGroup(ctx,
 parameterGroupName)
 if err != nil {
  panic(err)
 }
 if parameterGroup == nil {
  log.Printf("Getting available database engine versions for %v.\n", dbEngine)
  engineVersions, err := scenario.dbClusters.GetEngineVersions(ctx, dbEngine, "")
  if err != nil {
   panic(err)
  }

  familySet := map[string]struct{}{}
  for _, family := range engineVersions {
   familySet[*family.DBParameterGroupFamily] = struct{}{}
  }
```

```
  var families []string
  for family := range familySet {
   families = append(families, family)
  }
  sort.Strings(families)
  familyIndex := scenario.questioner.AskChoice("Which family do you want to use?\n",
 families)
  log.Println("Creating a DB cluster parameter group.")
  _, err = scenario.dbClusters.CreateParameterGroup(
   ctx, parameterGroupName, families[familyIndex], "Example parameter group.")
  if err != nil {
   panic(err)
  }
  parameterGroup, err = scenario.dbClusters.GetParameterGroup(ctx,
 parameterGroupName)
  if err != nil {
   panic(err)
  }
 }
 log.Printf("Parameter group %v:\n", *parameterGroup.DBParameterGroupFamily)
 log.Printf("\tName: %v\n", *parameterGroup.DBClusterParameterGroupName)
 log.Printf("\tARN: %v\n", *parameterGroup.DBClusterParameterGroupArn)
 log.Printf("\tFamily: %v\n", *parameterGroup.DBParameterGroupFamily)
 log.Printf("\tDescription: %v\n", *parameterGroup.Description)
 log.Println(strings.Repeat("-", 88))
 return parameterGroup

}

// SetUserParameters shows how to get the parameters contained in a custom parameter
// group and update some of the parameter values in the group.
func (scenario GetStartedClusters) SetUserParameters(ctx context.Context,
 parameterGroupName string) {
 log.Println("Let's set some parameter values in your parameter group.")
 dbParameters, err := scenario.dbClusters.GetParameters(ctx, parameterGroupName, "")
 if err != nil {
  panic(err)
 }
 var updateParams []types.Parameter
 for _, dbParam := range dbParameters {
  if strings.HasPrefix(*dbParam.ParameterName, "auto_increment") &&
   *dbParam.IsModifiable && *dbParam.DataType == "integer" {
   log.Printf("The %v parameter is described as:\n\t%v",
    *dbParam.ParameterName, *dbParam.Description)
```

```
    rangeSplit := strings.Split(*dbParam.AllowedValues, "-")
    lower, _ := strconv.Atoi(rangeSplit[0])
    upper, _ := strconv.Atoi(rangeSplit[1])
    newValue := scenario.questioner.AskInt(
     fmt.Sprintf("Enter a value between %v and %v:", lower, upper),
     demotools.InIntRange{Lower: lower, Upper: upper})
    dbParam.ParameterValue = aws.String(strconv.Itoa(newValue))
    updateParams = append(updateParams, dbParam)
   }
  }
  err = scenario.dbClusters.UpdateParameters(ctx, parameterGroupName, updateParams)
  if err != nil {
   panic(err)
  }
  log.Println("You can get a list of parameters you've set by specifying a source of
  'user'.")
  userParameters, err := scenario.dbClusters.GetParameters(ctx, parameterGroupName,
  "user")
  if err != nil {
   panic(err)
  }
  log.Println("Here are the parameters you've set:")
  for _, param := range userParameters {
   log.Printf("\t%v: %v\n", *param.ParameterName, *param.ParameterValue)
  }
  log.Println(strings.Repeat("-", 88))
}

// CreateCluster shows how to create an Aurora DB cluster that contains a database
// of a specified type. The database is also configured to use a custom DB cluster
// parameter group.
func (scenario GetStartedClusters) CreateCluster(ctx context.Context, clusterName
 string, dbEngine string,
 dbName string, parameterGroup *types.DBClusterParameterGroup) *types.DBCluster {

 log.Println("Checking for an existing DB cluster.")
 cluster, err := scenario.dbClusters.GetDbCluster(ctx, clusterName)
 if err != nil {
  panic(err)
 }
 if cluster == nil {
  adminUsername := scenario.questioner.Ask(
   "Enter an administrator user name for the database: ", demotools.NotEmpty{})
  adminPassword := scenario.questioner.Ask(
```

```
    "Enter a password for the administrator (at least 8 characters): ",
  demotools.NotEmpty{})
   engineVersions, err := scenario.dbClusters.GetEngineVersions(ctx, dbEngine,
  *parameterGroup.DBParameterGroupFamily)
   if err != nil {
    panic(err)
   }
   var engineChoices []string
   for _, engine := range engineVersions {
    engineChoices = append(engineChoices, *engine.EngineVersion)
   }
   log.Println("The available engines for your parameter group are:")
   engineIndex := scenario.questioner.AskChoice("Which engine do you want to use?\n",
  engineChoices)
   log.Printf("Creating DB cluster %v and database %v.\n", clusterName, dbName)
   log.Printf("The DB cluster is configured to use\nyour custom parameter group %v
 \n",
    *parameterGroup.DBClusterParameterGroupName)
   log.Printf("and selected engine %v.\n", engineChoices[engineIndex])
   log.Println("This typically takes several minutes.")
   cluster, err = scenario.dbClusters.CreateDbCluster(
    ctx, clusterName, *parameterGroup.DBClusterParameterGroupName, dbName, dbEngine,
    engineChoices[engineIndex], adminUsername, adminPassword)
   if err != nil {
    panic(err)
   }
   for *cluster.Status != "available" {
    scenario.helper.Pause(30)
    cluster, err = scenario.dbClusters.GetDbCluster(ctx, clusterName)
    if err != nil {
     panic(err)
    }
    log.Println("Cluster created and available.")
   }
  }
  log.Println("Cluster data:")
  log.Printf("\tDBClusterIdentifier: %v\n", *cluster.DBClusterIdentifier)
  log.Printf("\tARN: %v\n", *cluster.DBClusterArn)
  log.Printf("\tStatus: %v\n", *cluster.Status)
  log.Printf("\tEngine: %v\n", *cluster.Engine)
  log.Printf("\tEngine version: %v\n", *cluster.EngineVersion)
  log.Printf("\tDBClusterParameterGroup: %v\n", *cluster.DBClusterParameterGroup)
  log.Printf("\tEngineMode: %v\n", *cluster.EngineMode)
  log.Println(strings.Repeat("-", 88))
```

```
 return cluster
}

// CreateInstance shows how to create a DB instance in an existing Aurora DB
 cluster.
// A new DB cluster contains no DB instances, so you must add one. The first DB
 instance
// that is added to a DB cluster defaults to a read-write DB instance.
func (scenario GetStartedClusters) CreateInstance(ctx context.Context, cluster
 *types.DBCluster) *types.DBInstance {
 log.Println("Checking for an existing database instance.")
 dbInstance, err := scenario.dbClusters.GetInstance(ctx,
 *cluster.DBClusterIdentifier)
 if err != nil {
  panic(err)
 }
 if dbInstance == nil {
  log.Println("Let's create a database instance in your DB cluster.")
  log.Println("First, choose a DB instance type:")
  instOpts, err := scenario.dbClusters.GetOrderableInstances(
   ctx, *cluster.Engine, *cluster.EngineVersion)
  if err != nil {
   panic(err)
  }
  var instChoices []string
  for _, opt := range instOpts {
   instChoices = append(instChoices, *opt.DBInstanceClass)
  }
  slices.Sort(instChoices)
  instChoices = slices.Compact(instChoices)
  instIndex := scenario.questioner.AskChoice(
   "Which DB instance class do you want to use?\n", instChoices)
  log.Println("Creating a database instance. This typically takes several minutes.")
  dbInstance, err = scenario.dbClusters.CreateInstanceInCluster(
   ctx, *cluster.DBClusterIdentifier, *cluster.DBClusterIdentifier, *cluster.Engine,
   instChoices[instIndex])
  if err != nil {
   panic(err)
  }
  for *dbInstance.DBInstanceStatus != "available" {
   scenario.helper.Pause(30)
   dbInstance, err = scenario.dbClusters.GetInstance(ctx,
 *cluster.DBClusterIdentifier)
   if err != nil {
```

```
    panic(err)
   }
  }
 }
 log.Println("Instance data:")
 log.Printf("\tDBInstanceIdentifier: %v\n", *dbInstance.DBInstanceIdentifier)
 log.Printf("\tARN: %v\n", *dbInstance.DBInstanceArn)
 log.Printf("\tStatus: %v\n", *dbInstance.DBInstanceStatus)
 log.Printf("\tEngine: %v\n", *dbInstance.Engine)
 log.Printf("\tEngine version: %v\n", *dbInstance.EngineVersion)
 log.Println(strings.Repeat("-", 88))
 return dbInstance
}

// DisplayConnection displays connection information about an Aurora DB cluster and
 tips
// on how to connect to it.
func (scenario GetStartedClusters) DisplayConnection(cluster *types.DBCluster) {
 log.Println(
  "You can now connect to your database using your favorite MySql client.\n" +
   "One way to connect is by using the 'mysql' shell on an Amazon EC2 instance\n" +
   "that is running in the same VPC as your database cluster. Pass the endpoint,\n"
 +
   "port, and administrator user name to 'mysql' and enter your password\n" +
   "when prompted:")
 log.Printf("\n\tmysql -h %v -P %v -u %v -p\n",
  *cluster.Endpoint, *cluster.Port, *cluster.MasterUsername)
 log.Println("For more information, see the User Guide for Aurora:\n" +
  "\thttps://docs.aws.amazon.com/AmazonRDS/latest/AuroraUserGuide/
CHAP_GettingStartedAurora.CreatingConnecting.Aurora.html#CHAP_GettingStartedAurora.Aurora.Co
 log.Println(strings.Repeat("-", 88))
}

// CreateSnapshot shows how to create a DB cluster snapshot and wait until it's
 available.
func (scenario GetStartedClusters) CreateSnapshot(ctx context.Context, clusterName
 string) {
 if scenario.questioner.AskBool(
  "Do you want to create a snapshot of your DB cluster (y/n)? ", "y") {
  snapshotId := fmt.Sprintf("%v-%v", clusterName, scenario.helper.UniqueId())
  log.Printf("Creating a snapshot named %v. This typically takes a few minutes.\n",
 snapshotId)
  snapshot, err := scenario.dbClusters.CreateClusterSnapshot(ctx, clusterName,
 snapshotId)
```

```
  if err != nil {
   panic(err)
  }
  for *snapshot.Status != "available" {
   scenario.helper.Pause(30)
   snapshot, err = scenario.dbClusters.GetClusterSnapshot(ctx, snapshotId)
   if err != nil {
    panic(err)
   }
  }
  log.Println("Snapshot data:")
  log.Printf("\tDBClusterSnapshotIdentifier: %v\n",
 *snapshot.DBClusterSnapshotIdentifier)
  log.Printf("\tARN: %v\n", *snapshot.DBClusterSnapshotArn)
  log.Printf("\tStatus: %v\n", *snapshot.Status)
  log.Printf("\tEngine: %v\n", *snapshot.Engine)
  log.Printf("\tEngine version: %v\n", *snapshot.EngineVersion)
  log.Printf("\tDBClusterIdentifier: %v\n", *snapshot.DBClusterIdentifier)
  log.Printf("\tSnapshotCreateTime: %v\n", *snapshot.SnapshotCreateTime)
  log.Println(strings.Repeat("-", 88))
 }
}

// Cleanup shows how to clean up a DB instance, DB cluster, and DB cluster parameter
 group.
// Before the DB cluster parameter group can be deleted, all associated DB instances
 and
// DB clusters must first be deleted.
func (scenario GetStartedClusters) Cleanup(ctx context.Context, dbInstance
 *types.DBInstance, cluster *types.DBCluster,
 parameterGroup *types.DBClusterParameterGroup) {

 if scenario.questioner.AskBool(
  "\nDo you want to delete the database instance, DB cluster, and parameter group
 (y/n)? ", "y") {
  log.Printf("Deleting database instance %v.\n", *dbInstance.DBInstanceIdentifier)
  err := scenario.dbClusters.DeleteInstance(ctx, *dbInstance.DBInstanceIdentifier)
  if err != nil {
   panic(err)
  }
  log.Printf("Deleting database cluster %v.\n", *cluster.DBClusterIdentifier)
  err = scenario.dbClusters.DeleteDbCluster(ctx, *cluster.DBClusterIdentifier)
  if err != nil {
   panic(err)
```

```
  }
  log.Println(
   "Waiting for the DB instance and DB cluster to delete. This typically takes
 several minutes.")
  for dbInstance != nil || cluster != nil {
   scenario.helper.Pause(30)
   if dbInstance != nil {
    dbInstance, err = scenario.dbClusters.GetInstance(ctx,
 *dbInstance.DBInstanceIdentifier)
    if err != nil {
     panic(err)
    }
   }
   if cluster != nil {
    cluster, err = scenario.dbClusters.GetDbCluster(ctx,
 *cluster.DBClusterIdentifier)
    if err != nil {
     panic(err)
    }
   }
  }
  log.Printf("Deleting parameter group %v.",
 *parameterGroup.DBClusterParameterGroupName)
  err = scenario.dbClusters.DeleteParameterGroup(ctx,
 *parameterGroup.DBClusterParameterGroupName)
  if err != nil {
   panic(err)
  }
 }
}

// IScenarioHelper abstracts the function from a scenario so that it
// can be mocked for unit testing.
type IScenarioHelper interface {
 Pause(secs int)
 UniqueId() string
}
type ScenarioHelper struct{}

// Pause waits for the specified number of seconds.
func (helper ScenarioHelper) Pause(secs int) {
 time.Sleep(time.Duration(secs) * time.Second)
}
```

```
// UniqueId returns a new UUID.
func (helper ScenarioHelper) UniqueId() string {
 return uuid.New().String()
}
```

Definieren Sie Funktionen, die vom Szenario aufgerufen werden, um Aurora-Aktionen zu verwalten.

```
import (
 "context"
 "errors"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/rds"
 "github.com/aws/aws-sdk-go-v2/service/rds/types"
)

type DbClusters struct {
 AuroraClient *rds.Client
}


// GetParameterGroup gets a DB cluster parameter group by name.
func (clusters *DbClusters) GetParameterGroup(ctx context.Context,
 parameterGroupName string) (
 *types.DBClusterParameterGroup, error) {
 output, err := clusters.AuroraClient.DescribeDBClusterParameterGroups(
  ctx, &rds.DescribeDBClusterParameterGroupsInput{
   DBClusterParameterGroupName: aws.String(parameterGroupName),
  })
 if err != nil {
  var notFoundError *types.DBParameterGroupNotFoundFault
  if errors.As(err, &notFoundError) {
   log.Printf("Parameter group %v does not exist.\n", parameterGroupName)
   err = nil
  } else {
   log.Printf("Error getting parameter group %v: %v\n", parameterGroupName, err)
  }
  return nil, err
```

```
 } else {
  return &output.DBClusterParameterGroups[0], err
 }
}


// CreateParameterGroup creates a DB cluster parameter group that is based on the
 specified
// parameter group family.
func (clusters *DbClusters) CreateParameterGroup(
 ctx context.Context, parameterGroupName string, parameterGroupFamily string,
 description string) (
 *types.DBClusterParameterGroup, error) {

 output, err := clusters.AuroraClient.CreateDBClusterParameterGroup(ctx,
  &rds.CreateDBClusterParameterGroupInput{
   DBClusterParameterGroupName: aws.String(parameterGroupName),
   DBParameterGroupFamily:      aws.String(parameterGroupFamily),
   Description:                 aws.String(description),
  })
 if err != nil {
  log.Printf("Couldn't create parameter group %v: %v\n", parameterGroupName, err)
  return nil, err
 } else {
  return output.DBClusterParameterGroup, err
 }
}



// DeleteParameterGroup deletes the named DB cluster parameter group.
func (clusters *DbClusters) DeleteParameterGroup(ctx context.Context,
 parameterGroupName string) error {
 _, err := clusters.AuroraClient.DeleteDBClusterParameterGroup(ctx,
  &rds.DeleteDBClusterParameterGroupInput{
   DBClusterParameterGroupName: aws.String(parameterGroupName),
  })
 if err != nil {
  log.Printf("Couldn't delete parameter group %v: %v\n", parameterGroupName, err)
  return err
 } else {
  return nil
 }
}
```

```
// GetParameters gets the parameters that are contained in a DB cluster parameter
 group.
func (clusters *DbClusters) GetParameters(ctx context.Context, parameterGroupName
 string, source string) (
 []types.Parameter, error) {

 var output *rds.DescribeDBClusterParametersOutput
 var params []types.Parameter
 var err error
 parameterPaginator :=
 rds.NewDescribeDBClusterParametersPaginator(clusters.AuroraClient,
  &rds.DescribeDBClusterParametersInput{
   DBClusterParameterGroupName: aws.String(parameterGroupName),
   Source:                      aws.String(source),
  })
 for parameterPaginator.HasMorePages() {
  output, err = parameterPaginator.NextPage(ctx)
  if err != nil {
   log.Printf("Couldn't get paramaeters for %v: %v\n", parameterGroupName, err)
   break
  } else {
   params = append(params, output.Parameters...)
  }
 }
 return params, err
}



// UpdateParameters updates parameters in a named DB cluster parameter group.
func (clusters *DbClusters) UpdateParameters(ctx context.Context, parameterGroupName
 string, params []types.Parameter) error {
 _, err := clusters.AuroraClient.ModifyDBClusterParameterGroup(ctx,
  &rds.ModifyDBClusterParameterGroupInput{
   DBClusterParameterGroupName: aws.String(parameterGroupName),
   Parameters:                  params,
  })
 if err != nil {
  log.Printf("Couldn't update parameters in %v: %v\n", parameterGroupName, err)
  return err
 } else {
```

```
  return nil
 }
}



// GetDbCluster gets data about an Aurora DB cluster.
func (clusters *DbClusters) GetDbCluster(ctx context.Context, clusterName string)
 (*types.DBCluster, error) {
 output, err := clusters.AuroraClient.DescribeDBClusters(ctx,
  &rds.DescribeDBClustersInput{
   DBClusterIdentifier: aws.String(clusterName),
  })
 if err != nil {
  var notFoundError *types.DBClusterNotFoundFault
  if errors.As(err, &notFoundError) {
   log.Printf("DB cluster %v does not exist.\n", clusterName)
   err = nil
  } else {
   log.Printf("Couldn't get DB cluster %v: %v\n", clusterName, err)
  }
  return nil, err
 } else {
  return &output.DBClusters[0], err
 }
}



// CreateDbCluster creates a DB cluster that is configured to use the specified
 parameter group.
// The newly created DB cluster contains a database that uses the specified engine
 and
// engine version.
func (clusters *DbClusters) CreateDbCluster(ctx context.Context, clusterName string,
 parameterGroupName string,
 dbName string, dbEngine string, dbEngineVersion string, adminName string,
 adminPassword string) (
 *types.DBCluster, error) {

 output, err := clusters.AuroraClient.CreateDBCluster(ctx,
 &rds.CreateDBClusterInput{
  DBClusterIdentifier:         aws.String(clusterName),
  Engine:                      aws.String(dbEngine),
```

```
  DBClusterParameterGroupName: aws.String(parameterGroupName),
  DatabaseName:                aws.String(dbName),
  EngineVersion:               aws.String(dbEngineVersion),
  MasterUserPassword:          aws.String(adminPassword),
  MasterUsername:              aws.String(adminName),
 })
 if err != nil {
  log.Printf("Couldn't create DB cluster %v: %v\n", clusterName, err)
  return nil, err
 } else {
  return output.DBCluster, err
 }
}



// DeleteDbCluster deletes a DB cluster without keeping a final snapshot.
func (clusters *DbClusters) DeleteDbCluster(ctx context.Context, clusterName string)
 error {
 _, err := clusters.AuroraClient.DeleteDBCluster(ctx, &rds.DeleteDBClusterInput{
  DBClusterIdentifier: aws.String(clusterName),
  SkipFinalSnapshot:   aws.Bool(true),
 })
 if err != nil {
  log.Printf("Couldn't delete DB cluster %v: %v\n", clusterName, err)
  return err
 } else {
  return nil
 }
}



// CreateClusterSnapshot creates a snapshot of a DB cluster.
func (clusters *DbClusters) CreateClusterSnapshot(ctx context.Context, clusterName
 string, snapshotName string) (
 *types.DBClusterSnapshot, error) {
 output, err := clusters.AuroraClient.CreateDBClusterSnapshot(ctx,
 &rds.CreateDBClusterSnapshotInput{
  DBClusterIdentifier:         aws.String(clusterName),
  DBClusterSnapshotIdentifier: aws.String(snapshotName),
 })
 if err != nil {
  log.Printf("Couldn't create snapshot %v: %v\n", snapshotName, err)
```

```
  return nil, err
 } else {
  return output.DBClusterSnapshot, nil
 }
}



// GetClusterSnapshot gets a DB cluster snapshot.
func (clusters *DbClusters) GetClusterSnapshot(ctx context.Context, snapshotName
 string) (*types.DBClusterSnapshot, error) {
 output, err := clusters.AuroraClient.DescribeDBClusterSnapshots(ctx,
  &rds.DescribeDBClusterSnapshotsInput{
   DBClusterSnapshotIdentifier: aws.String(snapshotName),
  })
 if err != nil {
  log.Printf("Couldn't get snapshot %v: %v\n", snapshotName, err)
  return nil, err
 } else {
  return &output.DBClusterSnapshots[0], nil
 }
}



// CreateInstanceInCluster creates a database instance in an existing DB cluster.
 The first database that is
// created defaults to a read-write DB instance.
func (clusters *DbClusters) CreateInstanceInCluster(ctx context.Context, clusterName
 string, instanceName string,
 dbEngine string, dbInstanceClass string) (*types.DBInstance, error) {
 output, err := clusters.AuroraClient.CreateDBInstance(ctx,
 &rds.CreateDBInstanceInput{
  DBInstanceIdentifier: aws.String(instanceName),
  DBClusterIdentifier:  aws.String(clusterName),
  Engine:               aws.String(dbEngine),
  DBInstanceClass:      aws.String(dbInstanceClass),
 })
 if err != nil {
  log.Printf("Couldn't create instance %v: %v\n", instanceName, err)
  return nil, err
 } else {
  return output.DBInstance, nil
 }
```

```
}



// GetInstance gets data about a DB instance.
func (clusters *DbClusters) GetInstance(ctx context.Context, instanceName string) (
 *types.DBInstance, error) {
 output, err := clusters.AuroraClient.DescribeDBInstances(ctx,
  &rds.DescribeDBInstancesInput{
   DBInstanceIdentifier: aws.String(instanceName),
  })
 if err != nil {
  var notFoundError *types.DBInstanceNotFoundFault
  if errors.As(err, &notFoundError) {
   log.Printf("DB instance %v does not exist.\n", instanceName)
   err = nil
  } else {
   log.Printf("Couldn't get instance %v: %v\n", instanceName, err)
  }
  return nil, err
 } else {
  return &output.DBInstances[0], nil
 }
}



// DeleteInstance deletes a DB instance.
func (clusters *DbClusters) DeleteInstance(ctx context.Context, instanceName string)
 error {
 _, err := clusters.AuroraClient.DeleteDBInstance(ctx, &rds.DeleteDBInstanceInput{
  DBInstanceIdentifier:   aws.String(instanceName),
  SkipFinalSnapshot:      aws.Bool(true),
  DeleteAutomatedBackups: aws.Bool(true),
 })
 if err != nil {
  log.Printf("Couldn't delete instance %v: %v\n", instanceName, err)
  return err
 } else {
  return nil
 }
}
```

```
// GetEngineVersions gets database engine versions that are available for the
 specified engine
// and parameter group family.
func (clusters *DbClusters) GetEngineVersions(ctx context.Context, engine string,
 parameterGroupFamily string) (
 []types.DBEngineVersion, error) {
 output, err := clusters.AuroraClient.DescribeDBEngineVersions(ctx,
  &rds.DescribeDBEngineVersionsInput{
   Engine:                 aws.String(engine),
   DBParameterGroupFamily: aws.String(parameterGroupFamily),
  })
 if err != nil {
  log.Printf("Couldn't get engine versions for %v: %v\n", engine, err)
  return nil, err
 } else {
  return output.DBEngineVersions, nil
 }
}



// GetOrderableInstances uses a paginator to get DB instance options that can be
 used to create DB instances that are
// compatible with a set of specifications.
func (clusters *DbClusters) GetOrderableInstances(ctx context.Context, engine
 string, engineVersion string) (
 []types.OrderableDBInstanceOption, error) {

 var output *rds.DescribeOrderableDBInstanceOptionsOutput
 var instances []types.OrderableDBInstanceOption
 var err error
 orderablePaginator :=
 rds.NewDescribeOrderableDBInstanceOptionsPaginator(clusters.AuroraClient,
  &rds.DescribeOrderableDBInstanceOptionsInput{
   Engine:        aws.String(engine),
   EngineVersion: aws.String(engineVersion),
  })
 for orderablePaginator.HasMorePages() {
  output, err = orderablePaginator.NextPage(ctx)
  if err != nil {
   log.Printf("Couldn't get orderable DB instances: %v\n", err)
   break
  } else {
```

```
    instances = append(instances, output.OrderableDBInstanceOptions...)
   }
  }
  return instances, err
}
```

- API-Details finden Sie in den folgenden Themen der AWS SDK für Go -API-Referenz.
  - CreateDBCluster
  - CreateDBClusterParameterGroup
  - DBClusterSchnappschuss erstellen
  - CreateDBInstance
  - LöschenDBCluster
  - LöschenDBClusterParameterGroup
  - LöschenDBInstance
  - Beschreiben DBCluster ParameterGroups
  - Beschreiben Sie die DBCluster Parameter
  - Beschreiben Sie DBCluster Schnappschüsse
  - Beschreiben DBClusters
  - DBEngineVersionen beschreiben
  - Beschreiben DBInstances
  - DescribeOrderableDBInstanceOptionen
  - Modifizieren SieDBClusterParameterGroup

Aktionen

`CreateDBCluster`

Das folgende Codebeispiel zeigt die Verwendung`CreateDBCluster`.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import (
 "context"
 "errors"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/rds"
 "github.com/aws/aws-sdk-go-v2/service/rds/types"
)

type DbClusters struct {
 AuroraClient *rds.Client
}



// CreateDbCluster creates a DB cluster that is configured to use the specified
 parameter group.
// The newly created DB cluster contains a database that uses the specified engine
 and
// engine version.
func (clusters *DbClusters) CreateDbCluster(ctx context.Context, clusterName string,
 parameterGroupName string,
 dbName string, dbEngine string, dbEngineVersion string, adminName string,
 adminPassword string) (
 *types.DBCluster, error) {

 output, err := clusters.AuroraClient.CreateDBCluster(ctx,
 &rds.CreateDBClusterInput{
  DBClusterIdentifier:         aws.String(clusterName),
  Engine:                      aws.String(dbEngine),
  DBClusterParameterGroupName: aws.String(parameterGroupName),
  DatabaseName:                aws.String(dbName),
```

```
  EngineVersion:               aws.String(dbEngineVersion),
  MasterUserPassword:          aws.String(adminPassword),
  MasterUsername:              aws.String(adminName),
 })
 if err != nil {
  log.Printf("Couldn't create DB cluster %v: %v\n", clusterName, err)
  return nil, err
 } else {
  return output.DBCluster, err
 }
}
```

- Einzelheiten zur API finden Sie unter DBCluster In der AWS SDK für Go API-Referenz erstellen.

`CreateDBClusterParameterGroup`

Das folgende Codebeispiel zeigt die Verwendung`CreateDBClusterParameterGroup`.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import (
 "context"
 "errors"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/rds"
 "github.com/aws/aws-sdk-go-v2/service/rds/types"
)

type DbClusters struct {
 AuroraClient *rds.Client
```

```
}



// CreateParameterGroup creates a DB cluster parameter group that is based on the
 specified
// parameter group family.
func (clusters *DbClusters) CreateParameterGroup(
 ctx context.Context, parameterGroupName string, parameterGroupFamily string,
 description string) (
 *types.DBClusterParameterGroup, error) {

 output, err := clusters.AuroraClient.CreateDBClusterParameterGroup(ctx,
  &rds.CreateDBClusterParameterGroupInput{
   DBClusterParameterGroupName: aws.String(parameterGroupName),
   DBParameterGroupFamily:      aws.String(parameterGroupFamily),
   Description:                 aws.String(description),
  })
 if err != nil {
  log.Printf("Couldn't create parameter group %v: %v\n", parameterGroupName, err)
  return nil, err
 } else {
  return output.DBClusterParameterGroup, err
 }
}
```

- Einzelheiten zur API finden Sie unter DBCluster ParameterGroup In der AWS SDK für Go API-Referenz erstellen.

`CreateDBClusterSnapshot`

Das folgende Codebeispiel zeigt die Verwendung`CreateDBClusterSnapshot`.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import (
 "context"
 "errors"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/rds"
 "github.com/aws/aws-sdk-go-v2/service/rds/types"
)

type DbClusters struct {
 AuroraClient *rds.Client
}



// CreateClusterSnapshot creates a snapshot of a DB cluster.
func (clusters *DbClusters) CreateClusterSnapshot(ctx context.Context, clusterName
 string, snapshotName string) (
 *types.DBClusterSnapshot, error) {
 output, err := clusters.AuroraClient.CreateDBClusterSnapshot(ctx,
 &rds.CreateDBClusterSnapshotInput{
  DBClusterIdentifier:         aws.String(clusterName),
  DBClusterSnapshotIdentifier: aws.String(snapshotName),
 })
 if err != nil {
  log.Printf("Couldn't create snapshot %v: %v\n", snapshotName, err)
  return nil, err
 } else {
  return output.DBClusterSnapshot, nil
 }
}
```

- Einzelheiten zur API finden Sie unter DBClusterSnapshot erstellen in der AWS SDK für Go API-Referenz.

`CreateDBInstance`

Das folgende Codebeispiel zeigt die Verwendung`CreateDBInstance`.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import (
 "context"
 "errors"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/rds"
 "github.com/aws/aws-sdk-go-v2/service/rds/types"
)

type DbClusters struct {
 AuroraClient *rds.Client
}



// CreateInstanceInCluster creates a database instance in an existing DB cluster.
 The first database that is
// created defaults to a read-write DB instance.
func (clusters *DbClusters) CreateInstanceInCluster(ctx context.Context, clusterName
 string, instanceName string,
 dbEngine string, dbInstanceClass string) (*types.DBInstance, error) {
 output, err := clusters.AuroraClient.CreateDBInstance(ctx,
 &rds.CreateDBInstanceInput{
  DBInstanceIdentifier: aws.String(instanceName),
  DBClusterIdentifier:  aws.String(clusterName),
  Engine:               aws.String(dbEngine),
  DBInstanceClass:      aws.String(dbInstanceClass),
 })
 if err != nil {
  log.Printf("Couldn't create instance %v: %v\n", instanceName, err)
  return nil, err
 } else {
```

```
  return output.DBInstance, nil
 }
}
```

- Einzelheiten zur API finden Sie unter DBInstance In der AWS SDK für Go API-Referenz erstellen.

`DeleteDBCluster`

Das folgende Codebeispiel zeigt die Verwendung`DeleteDBCluster`.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import (
 "context"
 "errors"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/rds"
 "github.com/aws/aws-sdk-go-v2/service/rds/types"
)

type DbClusters struct {
 AuroraClient *rds.Client
}



// DeleteDbCluster deletes a DB cluster without keeping a final snapshot.
func (clusters *DbClusters) DeleteDbCluster(ctx context.Context, clusterName string)
 error {
```

```
 _, err := clusters.AuroraClient.DeleteDBCluster(ctx, &rds.DeleteDBClusterInput{
  DBClusterIdentifier: aws.String(clusterName),
  SkipFinalSnapshot:   aws.Bool(true),
 })
 if err != nil {
  log.Printf("Couldn't delete DB cluster %v: %v\n", clusterName, err)
  return err
 } else {
  return nil
 }
}
```

- Einzelheiten zur API finden Sie unter Löschen DBCluster in der AWS SDK für Go API-Referenz.

`DeleteDBClusterParameterGroup`

Das folgende Codebeispiel zeigt die Verwendung`DeleteDBClusterParameterGroup`.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import (
 "context"
 "errors"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/rds"
 "github.com/aws/aws-sdk-go-v2/service/rds/types"
)

type DbClusters struct {
 AuroraClient *rds.Client
}
```

```
// DeleteParameterGroup deletes the named DB cluster parameter group.
func (clusters *DbClusters) DeleteParameterGroup(ctx context.Context,
 parameterGroupName string) error {
 _, err := clusters.AuroraClient.DeleteDBClusterParameterGroup(ctx,
  &rds.DeleteDBClusterParameterGroupInput{
   DBClusterParameterGroupName: aws.String(parameterGroupName),
  })
 if err != nil {
  log.Printf("Couldn't delete parameter group %v: %v\n", parameterGroupName, err)
  return err
 } else {
  return nil
 }
}
```

- Einzelheiten zur API finden Sie unter Löschen DBCluster ParameterGroup in der AWS SDK für Go API-Referenz.

`DeleteDBInstance`

Das folgende Codebeispiel zeigt die Verwendung`DeleteDBInstance`.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import (
 "context"
 "errors"
 "log"
```

```
 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/rds"
 "github.com/aws/aws-sdk-go-v2/service/rds/types"
)

type DbClusters struct {
 AuroraClient *rds.Client
}



// DeleteInstance deletes a DB instance.
func (clusters *DbClusters) DeleteInstance(ctx context.Context, instanceName string)
 error {
 _, err := clusters.AuroraClient.DeleteDBInstance(ctx, &rds.DeleteDBInstanceInput{
  DBInstanceIdentifier:   aws.String(instanceName),
  SkipFinalSnapshot:      aws.Bool(true),
  DeleteAutomatedBackups: aws.Bool(true),
 })
 if err != nil {
  log.Printf("Couldn't delete instance %v: %v\n", instanceName, err)
  return err
 } else {
  return nil
 }
}
```

- Einzelheiten zur API finden Sie unter Löschen DBInstance in der AWS SDK für Go API-Referenz.

`DescribeDBClusterParameterGroups`

Das folgende Codebeispiel zeigt die Verwendung`DescribeDBClusterParameterGroups`.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import (
 "context"
 "errors"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/rds"
 "github.com/aws/aws-sdk-go-v2/service/rds/types"
)

type DbClusters struct {
 AuroraClient *rds.Client
}



// GetParameterGroup gets a DB cluster parameter group by name.
func (clusters *DbClusters) GetParameterGroup(ctx context.Context,
 parameterGroupName string) (
 *types.DBClusterParameterGroup, error) {
 output, err := clusters.AuroraClient.DescribeDBClusterParameterGroups(
  ctx, &rds.DescribeDBClusterParameterGroupsInput{
   DBClusterParameterGroupName: aws.String(parameterGroupName),
  })
 if err != nil {
  var notFoundError *types.DBParameterGroupNotFoundFault
  if errors.As(err, &notFoundError) {
   log.Printf("Parameter group %v does not exist.\n", parameterGroupName)
   err = nil
  } else {
   log.Printf("Error getting parameter group %v: %v\n", parameterGroupName, err)
  }
  return nil, err
```

```
 } else {
  return &output.DBClusterParameterGroups[0], err
 }
}
```

- Einzelheiten zur API finden Sie unter Describe DBCluster ParameterGroups in der AWS SDK für Go API-Referenz.

`DescribeDBClusterParameters`

Das folgende Codebeispiel zeigt die Verwendung`DescribeDBClusterParameters`.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import (
 "context"
 "errors"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/rds"
 "github.com/aws/aws-sdk-go-v2/service/rds/types"
)

type DbClusters struct {
 AuroraClient *rds.Client
}



// GetParameters gets the parameters that are contained in a DB cluster parameter
 group.
```

```
func (clusters *DbClusters) GetParameters(ctx context.Context, parameterGroupName
 string, source string) (
 []types.Parameter, error) {

 var output *rds.DescribeDBClusterParametersOutput
 var params []types.Parameter
 var err error
 parameterPaginator :=
 rds.NewDescribeDBClusterParametersPaginator(clusters.AuroraClient,
  &rds.DescribeDBClusterParametersInput{
   DBClusterParameterGroupName: aws.String(parameterGroupName),
   Source:                      aws.String(source),
  })
 for parameterPaginator.HasMorePages() {
  output, err = parameterPaginator.NextPage(ctx)
  if err != nil {
   log.Printf("Couldn't get paramaeters for %v: %v\n", parameterGroupName, err)
   break
  } else {
   params = append(params, output.Parameters...)
  }
 }
 return params, err
}
```

- Einzelheiten zur API finden Sie unter Describe DBCluster Parameters in der AWS SDK für Go API-Referenz.

`DescribeDBClusterSnapshots`

Das folgende Codebeispiel zeigt die Verwendung`DescribeDBClusterSnapshots`.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import (
 "context"
 "errors"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/rds"
 "github.com/aws/aws-sdk-go-v2/service/rds/types"
)

type DbClusters struct {
 AuroraClient *rds.Client
}



// GetClusterSnapshot gets a DB cluster snapshot.
func (clusters *DbClusters) GetClusterSnapshot(ctx context.Context, snapshotName
 string) (*types.DBClusterSnapshot, error) {
 output, err := clusters.AuroraClient.DescribeDBClusterSnapshots(ctx,
  &rds.DescribeDBClusterSnapshotsInput{
   DBClusterSnapshotIdentifier: aws.String(snapshotName),
  })
 if err != nil {
  log.Printf("Couldn't get snapshot %v: %v\n", snapshotName, err)
  return nil, err
 } else {
  return &output.DBClusterSnapshots[0], nil
 }
}
```

- Einzelheiten zur API finden Sie unter Beschreiben von DBCluster Snapshots in der AWS SDK für Go API-Referenz.

`DescribeDBClusters`

Das folgende Codebeispiel zeigt die Verwendung`DescribeDBClusters`.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import (
 "context"
 "errors"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/rds"
 "github.com/aws/aws-sdk-go-v2/service/rds/types"
)

type DbClusters struct {
 AuroraClient *rds.Client
}



// GetDbCluster gets data about an Aurora DB cluster.
func (clusters *DbClusters) GetDbCluster(ctx context.Context, clusterName string)
 (*types.DBCluster, error) {
 output, err := clusters.AuroraClient.DescribeDBClusters(ctx,
  &rds.DescribeDBClustersInput{
   DBClusterIdentifier: aws.String(clusterName),
  })
 if err != nil {
  var notFoundError *types.DBClusterNotFoundFault
  if errors.As(err, &notFoundError) {
   log.Printf("DB cluster %v does not exist.\n", clusterName)
   err = nil
  } else {
   log.Printf("Couldn't get DB cluster %v: %v\n", clusterName, err)
  }
  return nil, err
 } else {
```

```
  return &output.DBClusters[0], err
 }
}
```

- Einzelheiten zur API finden Sie unter Describe DBClusters in der AWS SDK für Go API-Referenz.

`DescribeDBEngineVersions`

Das folgende Codebeispiel zeigt die Verwendung`DescribeDBEngineVersions`.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import (
 "context"
 "errors"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/rds"
 "github.com/aws/aws-sdk-go-v2/service/rds/types"
)

type DbClusters struct {
 AuroraClient *rds.Client
}



// GetEngineVersions gets database engine versions that are available for the
 specified engine
// and parameter group family.
```

```
func (clusters *DbClusters) GetEngineVersions(ctx context.Context, engine string,
 parameterGroupFamily string) (
 []types.DBEngineVersion, error) {
 output, err := clusters.AuroraClient.DescribeDBEngineVersions(ctx,
  &rds.DescribeDBEngineVersionsInput{
   Engine:                 aws.String(engine),
   DBParameterGroupFamily: aws.String(parameterGroupFamily),
  })
 if err != nil {
  log.Printf("Couldn't get engine versions for %v: %v\n", engine, err)
  return nil, err
 } else {
  return output.DBEngineVersions, nil
 }
}
```

- Einzelheiten zur API finden Sie unter DBEngineVersionen beschreiben in der AWS SDK für Go API-Referenz.

`DescribeDBInstances`

Das folgende Codebeispiel zeigt die Verwendung`DescribeDBInstances`.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import (
 "context"
 "errors"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/rds"
```

```
 "github.com/aws/aws-sdk-go-v2/service/rds/types"
)

type DbClusters struct {
 AuroraClient *rds.Client
}



// GetInstance gets data about a DB instance.
func (clusters *DbClusters) GetInstance(ctx context.Context, instanceName string) (
 *types.DBInstance, error) {
 output, err := clusters.AuroraClient.DescribeDBInstances(ctx,
  &rds.DescribeDBInstancesInput{
   DBInstanceIdentifier: aws.String(instanceName),
  })
 if err != nil {
  var notFoundError *types.DBInstanceNotFoundFault
  if errors.As(err, &notFoundError) {
   log.Printf("DB instance %v does not exist.\n", instanceName)
   err = nil
  } else {
   log.Printf("Couldn't get instance %v: %v\n", instanceName, err)
  }
  return nil, err
 } else {
  return &output.DBInstances[0], nil
 }
}
```

- Einzelheiten zur API finden Sie unter Describe DBInstances in der AWS SDK für Go API-Referenz.

`DescribeOrderableDBInstanceOptions`

Das folgende Codebeispiel zeigt die Verwendung`DescribeOrderableDBInstanceOptions`.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import (
 "context"
 "errors"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/rds"
 "github.com/aws/aws-sdk-go-v2/service/rds/types"
)

type DbClusters struct {
 AuroraClient *rds.Client
}



// GetOrderableInstances uses a paginator to get DB instance options that can be
 used to create DB instances that are
// compatible with a set of specifications.
func (clusters *DbClusters) GetOrderableInstances(ctx context.Context, engine
 string, engineVersion string) (
 []types.OrderableDBInstanceOption, error) {

 var output *rds.DescribeOrderableDBInstanceOptionsOutput
 var instances []types.OrderableDBInstanceOption
 var err error
 orderablePaginator :=
 rds.NewDescribeOrderableDBInstanceOptionsPaginator(clusters.AuroraClient,
  &rds.DescribeOrderableDBInstanceOptionsInput{
   Engine:        aws.String(engine),
   EngineVersion: aws.String(engineVersion),
  })
 for orderablePaginator.HasMorePages() {
```

```
  output, err = orderablePaginator.NextPage(ctx)
  if err != nil {
   log.Printf("Couldn't get orderable DB instances: %v\n", err)
   break
  } else {
   instances = append(instances, output.OrderableDBInstanceOptions...)
  }
 }
 return instances, err
}
```

- Einzelheiten zur API finden Sie unter DescribeOrderableDBInstanceOptionen in der AWS SDK für Go API-Referenz.

`ModifyDBClusterParameterGroup`

Das folgende Codebeispiel zeigt die Verwendung`ModifyDBClusterParameterGroup`.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import (
 "context"
 "errors"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/rds"
 "github.com/aws/aws-sdk-go-v2/service/rds/types"
)

type DbClusters struct {
 AuroraClient *rds.Client
```

```
}



// UpdateParameters updates parameters in a named DB cluster parameter group.
func (clusters *DbClusters) UpdateParameters(ctx context.Context, parameterGroupName
 string, params []types.Parameter) error {
 _, err := clusters.AuroraClient.ModifyDBClusterParameterGroup(ctx,
  &rds.ModifyDBClusterParameterGroupInput{
   DBClusterParameterGroupName: aws.String(parameterGroupName),
   Parameters:                  params,
  })
 if err != nil {
  log.Printf("Couldn't update parameters in %v: %v\n", parameterGroupName, err)
  return err
 } else {
  return nil
 }
}
```

- Einzelheiten zur API finden Sie unter Ändern DBCluster ParameterGroup in der AWS SDK für Go API-Referenz.

Beispiele für Amazon Bedrock mit SDK for Go V2

Die folgenden Codebeispiele zeigen Ihnen, wie Sie mithilfe von AWS SDK für Go V2 mit Amazon Bedrock Aktionen ausführen und allgemeine Szenarien implementieren.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Service-Funktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarios anzeigen.

Jedes Beispiel enthält einen Link zum vollständigen Quellcode, in dem Sie Anweisungen zur Einrichtung und Ausführung des Codes im Kontext finden.

Erste Schritte

Hallo Amazon Bedrock

Die folgenden Codebeispiele zeigen, wie Sie mit Amazon Bedrock beginnen können.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
package main

import (
 "context"
 "fmt"

 "github.com/aws/aws-sdk-go-v2/config"
 "github.com/aws/aws-sdk-go-v2/service/bedrock"
)

const region = "us-east-1"

// main uses the AWS SDK for Go (v2) to create an Amazon Bedrock client and
// list the available foundation models in your account and the chosen region.
// This example uses the default settings specified in your shared credentials
// and config files.
func main() {
 ctx := context.Background()
 sdkConfig, err := config.LoadDefaultConfig(ctx, config.WithRegion(region))
 if err != nil {
  fmt.Println("Couldn't load default configuration. Have you set up your AWS
 account?")
  fmt.Println(err)
  return
 }
 bedrockClient := bedrock.NewFromConfig(sdkConfig)
 result, err := bedrockClient.ListFoundationModels(ctx,
 &bedrock.ListFoundationModelsInput{})
 if err != nil {
  fmt.Printf("Couldn't list foundation models. Here's why: %v\n", err)
  return
 }
 if len(result.ModelSummaries) == 0 {
```

```
  fmt.Println("There are no foundation models.")
 }
 for _, modelSummary := range result.ModelSummaries {
  fmt.Println(*modelSummary.ModelId)
 }
}
```

- Einzelheiten zur API finden Sie ListFoundationModelsin der AWS SDK für Go API-Referenz.

Themen

- Aktionen

Aktionen

`ListFoundationModels`

Das folgende Codebeispiel zeigt die Verwendung`ListFoundationModels`.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Listet die verfügbaren Bedrock Foundation-Modelle auf.

```
import (
 "context"
 "log"

 "github.com/aws/aws-sdk-go-v2/service/bedrock"
 "github.com/aws/aws-sdk-go-v2/service/bedrock/types"
)

// FoundationModelWrapper encapsulates Amazon Bedrock actions used in the examples.
```

```
// It contains a Bedrock service client that is used to perform foundation model
 actions.
type FoundationModelWrapper struct {
 BedrockClient *bedrock.Client
}



// ListPolicies lists Bedrock foundation models that you can use.
func (wrapper FoundationModelWrapper) ListFoundationModels(ctx context.Context)
 ([]types.FoundationModelSummary, error) {

 var models []types.FoundationModelSummary

 result, err := wrapper.BedrockClient.ListFoundationModels(ctx,
 &bedrock.ListFoundationModelsInput{})

 if err != nil {
  log.Printf("Couldn't list foundation models. Here's why: %v\n", err)
 } else {
  models = result.ModelSummaries
 }
 return models, err
}
```

- Einzelheiten zur API finden Sie ListFoundationModelsin der AWS SDK für Go API-Referenz.

Amazon Bedrock Runtime-Beispiele mit SDK for Go V2

Die folgenden Codebeispiele zeigen Ihnen, wie Sie Aktionen ausführen und allgemeine Szenarien implementieren, indem Sie AWS SDK für Go V2 mit Amazon Bedrock Runtime verwenden.

Szenarien sind Code-Beispiele, die Ihnen zeigen, wie Sie bestimmte Aufgaben ausführen, indem Sie mehrere Funktionen innerhalb eines Services aufrufen oder mit anderen AWS-Services kombinieren.

Jedes Beispiel enthält einen Link zum vollständigen Quellcode, in dem Sie Anweisungen zur Einrichtung und Ausführung des Codes im Kontext finden.

Erste Schritte

Hallo Amazon Bedrock

Die folgenden Codebeispiele zeigen, wie Sie mit Amazon Bedrock beginnen können.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
package main

import (
 "context"
 "encoding/json"
 "flag"
 "fmt"
 "log"
 "os"
 "strings"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/config"
 "github.com/aws/aws-sdk-go-v2/service/bedrockruntime"
)

// Each model provider defines their own individual request and response formats.
// For the format, ranges, and default values for the different models, refer to:
// https://docs.aws.amazon.com/bedrock/latest/userguide/model-parameters.html

type ClaudeRequest struct {
 Prompt            string `json:"prompt"`
 MaxTokensToSample int    `json:"max_tokens_to_sample"`
 // Omitting optional request parameters
}

type ClaudeResponse struct {
 Completion string `json:"completion"`
}
```

```
// main uses the AWS SDK for Go (v2) to create an Amazon Bedrock Runtime client
// and invokes Anthropic Claude 2 inside your account and the chosen region.
// This example uses the default settings specified in your shared credentials
// and config files.
func main() {

 region := flag.String("region", "us-east-1", "The AWS region")
 flag.Parse()

 fmt.Printf("Using AWS region: %s\n", *region)

 ctx := context.Background()
 sdkConfig, err := config.LoadDefaultConfig(ctx, config.WithRegion(*region))
 if err != nil {
  fmt.Println("Couldn't load default configuration. Have you set up your AWS
 account?")
  fmt.Println(err)
  return
 }

 client := bedrockruntime.NewFromConfig(sdkConfig)

 modelId := "anthropic.claude-v2"

 prompt := "Hello, how are you today?"

 // Anthropic Claude requires you to enclose the prompt as follows:
 prefix := "Human: "
 postfix := "\n\nAssistant:"
 wrappedPrompt := prefix + prompt + postfix

 request := ClaudeRequest{
  Prompt:            wrappedPrompt,
  MaxTokensToSample: 200,
 }

 body, err := json.Marshal(request)
 if err != nil {
  log.Panicln("Couldn't marshal the request: ", err)
 }

 result, err := client.InvokeModel(ctx, &bedrockruntime.InvokeModelInput{
  ModelId:     aws.String(modelId),
  ContentType: aws.String("application/json"),
```

```
  Body:        body,
 })

 if err != nil {
  errMsg := err.Error()
  if strings.Contains(errMsg, "no such host") {
   fmt.Printf("Error: The Bedrock service is not available in the selected
 region. Please double-check the service availability for your region at https://
aws.amazon.com/about-aws/global-infrastructure/regional-product-services/.\n")
  } else if strings.Contains(errMsg, "Could not resolve the foundation model") {
   fmt.Printf("Error: Could not resolve the foundation model from model identifier:
 \"%v\". Please verify that the requested model exists and is accessible within the
 specified region.\n", modelId)
  } else {
   fmt.Printf("Error: Couldn't invoke Anthropic Claude. Here's why: %v\n", err)
  }
  os.Exit(1)
 }

 var response ClaudeResponse

 err = json.Unmarshal(result.Body, &response)

 if err != nil {
  log.Fatal("failed to unmarshal", err)
 }
 fmt.Println("Prompt:\n", prompt)
 fmt.Println("Response from Anthropic Claude:\n", response.Completion)
}
```

- Einzelheiten zur API finden Sie InvokeModelin der AWS SDK für Go API-Referenz.

Themen

- Szenarien
- AI21 Labs Jurassic-2
- Amazon Titan Image Generator
- Amazon Titan Text
- Anthropic Claude

Szenarien

Rufen Sie mehrere Foundation-Modelle auf Amazon Bedrock auf

Das folgende Codebeispiel zeigt, wie Sie eine Aufforderung vorbereiten und an eine Vielzahl von großsprachigen Modellen (LLMs) auf Amazon Bedrock senden

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Rufen Sie mehrere Foundation-Modelle auf Amazon Bedrock auf.

```
import (
 "context"
 "encoding/base64"
 "fmt"
 "log"
 "math/rand"
 "os"
 "path/filepath"
 "strings"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/bedrockruntime"
 "github.com/awsdocs/aws-doc-sdk-examples/gov2/bedrock-runtime/actions"
 "github.com/awsdocs/aws-doc-sdk-examples/gov2/demotools"
)

// InvokeModelsScenario demonstrates how to use the Amazon Bedrock Runtime client
// to invoke various foundation models for text and image generation
//
// 1. Generate text with Anthropic Claude 2
// 2. Generate text with AI21 Labs Jurassic-2
// 3. Generate text with Meta Llama 2 Chat
// 4. Generate text and asynchronously process the response stream with Anthropic
 Claude 2
```

```
// 5. Generate an image with the Amazon Titan image generation model
// 6. Generate text with Amazon Titan Text G1 Express model
type InvokeModelsScenario struct {
 sdkConfig             aws.Config
 invokeModelWrapper    actions.InvokeModelWrapper
 responseStreamWrapper actions.InvokeModelWithResponseStreamWrapper
 questioner            demotools.IQuestioner
}

// NewInvokeModelsScenario constructs an InvokeModelsScenario instance from a
 configuration.
// It uses the specified config to get a Bedrock Runtime client and create wrappers
 for the
// actions used in the scenario.
func NewInvokeModelsScenario(sdkConfig aws.Config, questioner demotools.IQuestioner)
 InvokeModelsScenario {
 client := bedrockruntime.NewFromConfig(sdkConfig)
 return InvokeModelsScenario{
  sdkConfig:             sdkConfig,
  invokeModelWrapper:    actions.InvokeModelWrapper{BedrockRuntimeClient: client},
  responseStreamWrapper:
 actions.InvokeModelWithResponseStreamWrapper{BedrockRuntimeClient: client},
  questioner:            questioner,
 }
}

// Runs the interactive scenario.
func (scenario InvokeModelsScenario) Run(ctx context.Context) {
 defer func() {
  if r := recover(); r != nil {
   log.Printf("Something went wrong with the demo: %v\n", r)
  }
 }()

 log.Println(strings.Repeat("=", 77))
 log.Println("Welcome to the Amazon Bedrock Runtime model invocation demo.")
 log.Println(strings.Repeat("=", 77))

 log.Printf("First, let's invoke a few large-language models using the synchronous
 client:\n\n")

 text2textPrompt := "In one paragraph, who are you?"

 log.Println(strings.Repeat("-", 77))
```

```
 log.Printf("Invoking Claude with prompt: %v\n", text2textPrompt)
 scenario.InvokeClaude(ctx, text2textPrompt)

 log.Println(strings.Repeat("-", 77))
 log.Printf("Invoking Jurassic-2 with prompt: %v\n", text2textPrompt)
 scenario.InvokeJurassic2(ctx, text2textPrompt)

 log.Println(strings.Repeat("=", 77))
 log.Printf("Now, let's invoke Claude with the asynchronous client and process the
 response stream:\n\n")

 log.Println(strings.Repeat("-", 77))
 log.Printf("Invoking Claude with prompt: %v\n", text2textPrompt)
 scenario.InvokeWithResponseStream(ctx, text2textPrompt)

 log.Println(strings.Repeat("=", 77))
 log.Printf("Now, let's create an image with the Amazon Titan image generation
 model:\n\n")

 text2ImagePrompt := "stylized picture of a cute old steampunk robot"
 seed := rand.Int63n(2147483648)

 log.Println(strings.Repeat("-", 77))
 log.Printf("Invoking Amazon Titan with prompt: %v\n", text2ImagePrompt)
 scenario.InvokeTitanImage(ctx, text2ImagePrompt, seed)

 log.Println(strings.Repeat("-", 77))
 log.Printf("Invoking Titan Text Express with prompt: %v\n", text2textPrompt)
 scenario.InvokeTitanText(ctx, text2textPrompt)

 log.Println(strings.Repeat("=", 77))
 log.Println("Thanks for watching!")
 log.Println(strings.Repeat("=", 77))
}

func (scenario InvokeModelsScenario) InvokeClaude(ctx context.Context, prompt
 string) {
 completion, err := scenario.invokeModelWrapper.InvokeClaude(ctx, prompt)
 if err != nil {
  panic(err)
 }
 log.Printf("\nClaude     : %v\n", strings.TrimSpace(completion))
}
```

```
func (scenario InvokeModelsScenario) InvokeJurassic2(ctx context.Context, prompt
 string) {
 completion, err := scenario.invokeModelWrapper.InvokeJurassic2(ctx, prompt)
 if err != nil {
  panic(err)
 }
 log.Printf("\nJurassic-2 : %v\n", strings.TrimSpace(completion))
}

func (scenario InvokeModelsScenario) InvokeWithResponseStream(ctx context.Context,
 prompt string) {
 log.Println("\nClaude with response stream:")
 _, err := scenario.responseStreamWrapper.InvokeModelWithResponseStream(ctx, prompt)
 if err != nil {
  panic(err)
 }
 log.Println()
}

func (scenario InvokeModelsScenario) InvokeTitanImage(ctx context.Context, prompt
 string, seed int64) {
 base64ImageData, err := scenario.invokeModelWrapper.InvokeTitanImage(ctx, prompt,
 seed)
 if err != nil {
  panic(err)
 }
 imagePath := saveImage(base64ImageData, "amazon.titan-image-generator-v1")
 fmt.Printf("The generated image has been saved to %s\n", imagePath)
}

func (scenario InvokeModelsScenario) InvokeTitanText(ctx context.Context, prompt
 string) {
 completion, err := scenario.invokeModelWrapper.InvokeTitanText(ctx, prompt)
 if err != nil {
  panic(err)
 }
 log.Printf("\nTitan Text Express    : %v\n\n", strings.TrimSpace(completion))
}
```

- API-Details finden Sie in den folgenden Themen der AWS SDK für Go -API-Referenz.
  - InvokeModel

- InvokeModelWithResponseStream

AI21 Labs Jurassic-2

InvokeModel

Das folgende Codebeispiel zeigt, wie mithilfe der Invoke Model API eine Textnachricht an AI21 Labs Jurassic-2 gesendet wird.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Verwenden Sie die Invoke Model API, um eine Textnachricht zu senden.

```
import (
 "context"
 "encoding/json"
 "log"
 "strings"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/bedrockruntime"
)

// InvokeModelWrapper encapsulates Amazon Bedrock actions used in the examples.
// It contains a Bedrock Runtime client that is used to invoke foundation models.
type InvokeModelWrapper struct {
 BedrockRuntimeClient *bedrockruntime.Client
}



// Each model provider has their own individual request and response formats.
// For the format, ranges, and default values for AI21 Labs Jurassic-2, refer to:
```

```
// https://docs.aws.amazon.com/bedrock/latest/userguide/model-parameters-
jurassic2.html

type Jurassic2Request struct {
 Prompt      string  `json:"prompt"`
 MaxTokens   int     `json:"maxTokens,omitempty"`
 Temperature float64 `json:"temperature,omitempty"`
}

type Jurassic2Response struct {
 Completions []Completion `json:"completions"`
}
type Completion struct {
 Data Data `json:"data"`
}
type Data struct {
 Text string `json:"text"`
}

// Invokes AI21 Labs Jurassic-2 on Amazon Bedrock to run an inference using the
 input
// provided in the request body.
func (wrapper InvokeModelWrapper) InvokeJurassic2(ctx context.Context, prompt
 string) (string, error) {
 modelId := "ai21.j2-mid-v1"

 body, err := json.Marshal(Jurassic2Request{
  Prompt:      prompt,
  MaxTokens:   200,
  Temperature: 0.5,
 })

 if err != nil {
  log.Fatal("failed to marshal", err)
 }

 output, err := wrapper.BedrockRuntimeClient.InvokeModel(ctx,
 &bedrockruntime.InvokeModelInput{
  ModelId:     aws.String(modelId),
  ContentType: aws.String("application/json"),
  Body:        body,
 })

 if err != nil {
```

```
  ProcessError(err, modelId)
 }

 var response Jurassic2Response
 if err := json.Unmarshal(output.Body, &response); err != nil {
  log.Fatal("failed to unmarshal", err)
 }

 return response.Completions[0].Data.Text, nil
}
```

- Einzelheiten zur API finden Sie InvokeModelunter AWS SDK für Go API-Referenz.

Amazon Titan Image Generator

InvokeModel

Das folgende Codebeispiel zeigt, wie Amazon Titan Image auf Amazon Bedrock aufgerufen wird, um ein Bild zu generieren.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Erstellen Sie ein Bild mit dem Amazon Titan Image Generator.

```
import (
 "context"
 "encoding/json"
 "log"
 "strings"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/bedrockruntime"
)
```

```
// InvokeModelWrapper encapsulates Amazon Bedrock actions used in the examples.
// It contains a Bedrock Runtime client that is used to invoke foundation models.
type InvokeModelWrapper struct {
 BedrockRuntimeClient *bedrockruntime.Client
}



type TitanImageRequest struct {
 TaskType              string                `json:"taskType"`
 TextToImageParams     TextToImageParams     `json:"textToImageParams"`
 ImageGenerationConfig ImageGenerationConfig `json:"imageGenerationConfig"`
}
type TextToImageParams struct {
 Text string `json:"text"`
}
type ImageGenerationConfig struct {
 NumberOfImages int     `json:"numberOfImages"`
 Quality        string  `json:"quality"`
 CfgScale       float64 `json:"cfgScale"`
 Height         int     `json:"height"`
 Width          int     `json:"width"`
 Seed           int64   `json:"seed"`
}

type TitanImageResponse struct {
 Images []string `json:"images"`
}

// Invokes the Titan Image model to create an image using the input provided
// in the request body.
func (wrapper InvokeModelWrapper) InvokeTitanImage(ctx context.Context, prompt
 string, seed int64) (string, error) {
 modelId := "amazon.titan-image-generator-v1"

 body, err := json.Marshal(TitanImageRequest{
  TaskType: "TEXT_IMAGE",
  TextToImageParams: TextToImageParams{
   Text: prompt,
  },
  ImageGenerationConfig: ImageGenerationConfig{
   NumberOfImages: 1,
   Quality:        "standard",
```

```
    CfgScale:       8.0,
    Height:         512,
    Width:          512,
    Seed:           seed,
   },
  })

  if err != nil {
   log.Fatal("failed to marshal", err)
  }

  output, err := wrapper.BedrockRuntimeClient.InvokeModel(ctx,
  &bedrockruntime.InvokeModelInput{
   ModelId:     aws.String(modelId),
   ContentType: aws.String("application/json"),
   Body:        body,
  })

  if err != nil {
   ProcessError(err, modelId)
  }

  var response TitanImageResponse
  if err := json.Unmarshal(output.Body, &response); err != nil {
   log.Fatal("failed to unmarshal", err)
  }

  base64ImageData := response.Images[0]

  return base64ImageData, nil

}
```

- Einzelheiten zur API finden Sie InvokeModelunter AWS SDK für Go API-Referenz.

Amazon Titan Text

InvokeModel

Das folgende Codebeispiel zeigt, wie Sie mithilfe der Invoke Model API eine Textnachricht an Amazon Titan Text senden.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Verwenden Sie die Invoke Model API, um eine Textnachricht zu senden.

```
import (
 "context"
 "encoding/json"
 "log"
 "strings"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/bedrockruntime"
)

// InvokeModelWrapper encapsulates Amazon Bedrock actions used in the examples.
// It contains a Bedrock Runtime client that is used to invoke foundation models.
type InvokeModelWrapper struct {
 BedrockRuntimeClient *bedrockruntime.Client
}



// Each model provider has their own individual request and response formats.
// For the format, ranges, and default values for Amazon Titan Text, refer to:
// https://docs.aws.amazon.com/bedrock/latest/userguide/model-parameters-titan-
text.html
type TitanTextRequest struct {
 InputText            string               `json:"inputText"`
```

```
 TextGenerationConfig TextGenerationConfig `json:"textGenerationConfig"`
}

type TextGenerationConfig struct {
 Temperature   float64  `json:"temperature"`
 TopP          float64  `json:"topP"`
 MaxTokenCount int      `json:"maxTokenCount"`
 StopSequences []string `json:"stopSequences,omitempty"`
}

type TitanTextResponse struct {
 InputTextTokenCount int      `json:"inputTextTokenCount"`
 Results             []Result `json:"results"`
}

type Result struct {
 TokenCount       int    `json:"tokenCount"`
 OutputText       string `json:"outputText"`
 CompletionReason string `json:"completionReason"`
}

func (wrapper InvokeModelWrapper) InvokeTitanText(ctx context.Context, prompt
 string) (string, error) {
 modelId := "amazon.titan-text-express-v1"

 body, err := json.Marshal(TitanTextRequest{
  InputText: prompt,
  TextGenerationConfig: TextGenerationConfig{
   Temperature:   0,
   TopP:          1,
   MaxTokenCount: 4096,
  },
 })

 if err != nil {
  log.Fatal("failed to marshal", err)
 }

 output, err := wrapper.BedrockRuntimeClient.InvokeModel(ctx,
 &bedrockruntime.InvokeModelInput{
  ModelId:     aws.String(modelId),
  ContentType: aws.String("application/json"),
  Body:        body,
 })
```

```
 if err != nil {
  ProcessError(err, modelId)
 }

 var response TitanTextResponse
 if err := json.Unmarshal(output.Body, &response); err != nil {
  log.Fatal("failed to unmarshal", err)
 }

 return response.Results[0].OutputText, nil
}
```

- Einzelheiten zur API finden Sie InvokeModelunter AWS SDK für Go API-Referenz.

Anthropic Claude

Converse

Das folgende Codebeispiel zeigt, wie Sie mithilfe der Converse-API von Bedrock eine Textnachricht an Anthropic Claude senden.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Senden Sie mithilfe der Converse-API von Bedrock eine Textnachricht an Anthropic Claude.

```
import (
 "context"
 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/bedrockruntime"
 "github.com/aws/aws-sdk-go-v2/service/bedrockruntime/types"
)
```

```
// ConverseWrapper encapsulates Amazon Bedrock actions used in the examples.
// It contains a Bedrock Runtime client that is used to invoke Bedrock.
type ConverseWrapper struct {
 BedrockRuntimeClient *bedrockruntime.Client
}



func (wrapper ConverseWrapper) ConverseClaude(ctx context.Context, prompt string)
 (string, error) {
 var content = types.ContentBlockMemberText{
  Value: prompt,
 }
 var message = types.Message{
  Content: []types.ContentBlock{&content},
  Role:    "user",
 }
 modelId := "anthropic.claude-3-haiku-20240307-v1:0"
 var converseInput = bedrockruntime.ConverseInput{
  ModelId:  aws.String(modelId),
  Messages: []types.Message{message},
 }
 response, err := wrapper.BedrockRuntimeClient.Converse(ctx, &converseInput)
 if err != nil {
  ProcessError(err, modelId)
 }

 responseText, _ := response.Output.(*types.ConverseOutputMemberMessage)
 responseContentBlock := responseText.Value.Content[0]
 text, _ := responseContentBlock.(*types.ContentBlockMemberText)
 return text.Value, nil

}
```

- Einzelheiten zur API finden Sie unter Converse in der API-Referenz.AWS SDK für Go

InvokeModel

Das folgende Codebeispiel zeigt, wie Sie mithilfe der Invoke Model API eine Textnachricht an Anthropic Claude senden.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Rufen Sie das Anthropic Claude 2 Foundation-Modell auf, um Text zu generieren.

```
import (
 "context"
 "encoding/json"
 "log"
 "strings"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/bedrockruntime"
)

// InvokeModelWrapper encapsulates Amazon Bedrock actions used in the examples.
// It contains a Bedrock Runtime client that is used to invoke foundation models.
type InvokeModelWrapper struct {
 BedrockRuntimeClient *bedrockruntime.Client
}



// Each model provider has their own individual request and response formats.
// For the format, ranges, and default values for Anthropic Claude, refer to:
// https://docs.aws.amazon.com/bedrock/latest/userguide/model-parameters-claude.html

type ClaudeRequest struct {
 Prompt            string   `json:"prompt"`
 MaxTokensToSample int      `json:"max_tokens_to_sample"`
 Temperature       float64  `json:"temperature,omitempty"`
 StopSequences     []string `json:"stop_sequences,omitempty"`
}

type ClaudeResponse struct {
 Completion string `json:"completion"`
}
```

```
// Invokes Anthropic Claude on Amazon Bedrock to run an inference using the input
// provided in the request body.
func (wrapper InvokeModelWrapper) InvokeClaude(ctx context.Context, prompt string)
 (string, error) {
 modelId := "anthropic.claude-v2"

 // Anthropic Claude requires enclosing the prompt as follows:
 enclosedPrompt := "Human: " + prompt + "\n\nAssistant:"

 body, err := json.Marshal(ClaudeRequest{
  Prompt:            enclosedPrompt,
  MaxTokensToSample: 200,
  Temperature:       0.5,
  StopSequences:     []string{"\n\nHuman:"},
 })

 if err != nil {
  log.Fatal("failed to marshal", err)
 }

 output, err := wrapper.BedrockRuntimeClient.InvokeModel(ctx,
 &bedrockruntime.InvokeModelInput{
  ModelId:     aws.String(modelId),
  ContentType: aws.String("application/json"),
  Body:        body,
 })

 if err != nil {
  ProcessError(err, modelId)
 }

 var response ClaudeResponse
 if err := json.Unmarshal(output.Body, &response); err != nil {
  log.Fatal("failed to unmarshal", err)
 }

 return response.Completion, nil
}
```

- Einzelheiten zur API finden Sie InvokeModelin der AWS SDK für Go API-Referenz.

InvokeModelWithResponseStream

Das folgende Codebeispiel zeigt, wie Sie mithilfe der Invoke Model API eine Textnachricht an Modelle von Anthropic Claude senden und den Antwortstream drucken.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Verwenden Sie die Invoke Model API, um eine Textnachricht zu senden und den Antwortstream in Echtzeit zu verarbeiten.

```
import (
 "bytes"
 "context"
 "encoding/json"
 "fmt"
 "log"
 "strings"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/bedrockruntime"
 "github.com/aws/aws-sdk-go-v2/service/bedrockruntime/types"
)

// InvokeModelWithResponseStreamWrapper encapsulates Amazon Bedrock actions used in
 the examples.
// It contains a Bedrock Runtime client that is used to invoke foundation models.
type InvokeModelWithResponseStreamWrapper struct {
 BedrockRuntimeClient *bedrockruntime.Client
}



// Each model provider defines their own individual request and response formats.
// For the format, ranges, and default values for the different models, refer to:
// https://docs.aws.amazon.com/bedrock/latest/userguide/model-parameters.html
```

```
type Request struct {
 Prompt            string  `json:"prompt"`
 MaxTokensToSample int     `json:"max_tokens_to_sample"`
 Temperature       float64 `json:"temperature,omitempty"`
}

type Response struct {
 Completion string `json:"completion"`
}

// Invokes Anthropic Claude on Amazon Bedrock to run an inference and asynchronously
// process the response stream.

func (wrapper InvokeModelWithResponseStreamWrapper)
 InvokeModelWithResponseStream(ctx context.Context, prompt string) (string, error) {

 modelId := "anthropic.claude-v2"

 // Anthropic Claude requires you to enclose the prompt as follows:
 prefix := "Human: "
 postfix := "\n\nAssistant:"
 prompt = prefix + prompt + postfix

 request := ClaudeRequest{
  Prompt:            prompt,
  MaxTokensToSample: 200,
  Temperature:       0.5,
  StopSequences:     []string{"\n\nHuman:"},
 }

 body, err := json.Marshal(request)
 if err != nil {
  log.Panicln("Couldn't marshal the request: ", err)
 }

 output, err := wrapper.BedrockRuntimeClient.InvokeModelWithResponseStream(ctx,
 &bedrockruntime.InvokeModelWithResponseStreamInput{
  Body:        body,
  ModelId:     aws.String(modelId),
  ContentType: aws.String("application/json"),
 })

 if err != nil {
```

```
  errMsg := err.Error()
  if strings.Contains(errMsg, "no such host") {
   log.Printf("The Bedrock service is not available in the selected region. Please
 double-check the service availability for your region at https://aws.amazon.com/
about-aws/global-infrastructure/regional-product-services/.\n")
  } else if strings.Contains(errMsg, "Could not resolve the foundation model") {
   log.Printf("Could not resolve the foundation model from model identifier: \"%v
\". Please verify that the requested model exists and is accessible within the
 specified region.\n", modelId)
  } else {
   log.Printf("Couldn't invoke Anthropic Claude. Here's why: %v\n", err)
  }
 }

 resp, err := processStreamingOutput(ctx, output, func(ctx context.Context, part
 []byte) error {
  fmt.Print(string(part))
  return nil
 })

 if err != nil {
  log.Fatal("streaming output processing error: ", err)
 }

 return resp.Completion, nil

}

type StreamingOutputHandler func(ctx context.Context, part []byte) error

func processStreamingOutput(ctx context.Context, output
 *bedrockruntime.InvokeModelWithResponseStreamOutput, handler
 StreamingOutputHandler) (Response, error) {

 var combinedResult string
 resp := Response{}

 for event := range output.GetStream().Events() {
  switch v := event.(type) {
  case *types.ResponseStreamMemberChunk:

   //fmt.Println("payload", string(v.Value.Bytes))

   var resp Response
```

```
    err := json.NewDecoder(bytes.NewReader(v.Value.Bytes)).Decode(&resp)
    if err != nil {
     return resp, err
    }

    err = handler(ctx, []byte(resp.Completion))
    if err != nil {
     return resp, err
    }

    combinedResult += resp.Completion

   case *types.UnknownUnionMember:
    fmt.Println("unknown tag:", v.Tag)

   default:
    fmt.Println("union is nil or unknown type")
   }
  }

  resp.Completion = combinedResult

  return resp, nil
}
```

- Einzelheiten zur API finden Sie InvokeModelWithResponseStreamunter AWS SDK für Go API-Referenz.

AWS CloudFormation Beispiele mit SDK for Go V2

Die folgenden Codebeispiele zeigen Ihnen, wie Sie Aktionen ausführen und allgemeine Szenarien implementieren, indem Sie AWS SDK für Go V2 mit verwenden AWS CloudFormation.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Service-Funktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarios anzeigen.

Jedes Beispiel enthält einen Link zum vollständigen Quellcode, in dem Sie Anweisungen zum Einrichten und Ausführen des Codes im Kontext finden.

Themen

- Aktionen

Aktionen

`DescribeStacks`

Das folgende Codebeispiel zeigt die Verwendung`DescribeStacks`.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import (
 "context"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/cloudformation"
)

// StackOutputs defines a map of outputs from a specific stack.
type StackOutputs map[string]string

type CloudFormationActions struct {
 CfnClient *cloudformation.Client
}

// GetOutputs gets the outputs from a CloudFormation stack and puts them into a
 structured format.
func (actor CloudFormationActions) GetOutputs(ctx context.Context, stackName string)
 StackOutputs {
 output, err := actor.CfnClient.DescribeStacks(ctx,
 &cloudformation.DescribeStacksInput{
  StackName: aws.String(stackName),
 })
```

```
 if err != nil || len(output.Stacks) == 0 {
  log.Panicf("Couldn't find a CloudFormation stack named %v. Here's why: %v\n",
 stackName, err)
 }
 stackOutputs := StackOutputs{}
 for _, out := range output.Stacks[0].Outputs {
  stackOutputs[*out.OutputKey] = *out.OutputValue
 }
 return stackOutputs
}
```

- Einzelheiten zur API finden Sie DescribeStacksin der AWS SDK für Go API-Referenz.

CloudWatch Log-Beispiele mit SDK for Go V2

Die folgenden Codebeispiele zeigen Ihnen, wie Sie mithilfe von AWS SDK für Go V2 mit CloudWatch Logs Aktionen ausführen und allgemeine Szenarien implementieren.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Service-Funktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarios anzeigen.

Jedes Beispiel enthält einen Link zum vollständigen Quellcode, in dem Sie Anweisungen zum Einrichten und Ausführen des Codes im Kontext finden.

Themen

- Aktionen

Aktionen

`StartLiveTail`

Das folgende Codebeispiel zeigt die Verwendung`StartLiveTail`.

SDK für Go V2

Binden Sie die erforderlichen Dateien ein.

```
import (
```

```
 "context"
 "log"
 "time"

 "github.com/aws/aws-sdk-go-v2/config"
 "github.com/aws/aws-sdk-go-v2/service/cloudwatchlogs"
 "github.com/aws/aws-sdk-go-v2/service/cloudwatchlogs/types"
)
```

Behandeln Sie die Ereignisse aus der Live Tail-Sitzung.

```
func handleEventStreamAsync(stream *cloudwatchlogs.StartLiveTailEventStream) {
 eventsChan := stream.Events()
 for {
  event := <-eventsChan
  switch e := event.(type) {
  case *types.StartLiveTailResponseStreamMemberSessionStart:
   log.Println("Received SessionStart event")
  case *types.StartLiveTailResponseStreamMemberSessionUpdate:
   for _, logEvent := range e.Value.SessionResults {
    log.Println(*logEvent.Message)
   }
  default:
   // Handle on-stream exceptions
   if err := stream.Err(); err != nil {
    log.Fatalf("Error occured during streaming: %v", err)
   } else if event == nil {
    log.Println("Stream is Closed")
    return
   } else {
    log.Fatalf("Unknown event type: %T", e)
   }
  }
 }
}
```

Starten Sie die Live-Tail-Sitzung.

```
 cfg, err := config.LoadDefaultConfig(context.TODO())
 if err != nil {
  panic("configuration error, " + err.Error())
```

```
}
client := cloudwatchlogs.NewFromConfig(cfg)

request := &cloudwatchlogs.StartLiveTailInput{
 LogGroupIdentifiers:   logGroupIdentifiers,
 LogStreamNames:        logStreamNames,
 LogEventFilterPattern: logEventFilterPattern,
}

response, err := client.StartLiveTail(context.TODO(), request)
// Handle pre-stream Exceptions
if err != nil {
 log.Fatalf("Failed to start streaming: %v", err)
}

// Start a Goroutine to handle events over stream
stream := response.GetStream()
go handleEventStreamAsync(stream)
```

Beenden Sie die Live-Tail-Sitzung nach Ablauf einer gewissen Zeit.

```
// Close the stream (which ends the session) after a timeout
time.Sleep(10 * time.Second)
stream.Close()
log.Println("Event stream closed")
```

- Einzelheiten zur API finden Sie StartLiveTailin der AWS SDK für Go API-Referenz.

Beispiele für Amazon Cognito Identity Provider mit SDK for Go V2

Die folgenden Codebeispiele zeigen Ihnen, wie Sie Aktionen ausführen und allgemeine Szenarien implementieren, indem Sie AWS SDK für Go V2 mit Amazon Cognito Identity Provider verwenden.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Service-Funktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarios anzeigen.

Szenarien sind Code-Beispiele, die Ihnen zeigen, wie Sie bestimmte Aufgaben ausführen, indem Sie mehrere Funktionen innerhalb eines Services aufrufen oder mit anderen AWS-Services kombinieren.

Jedes Beispiel enthält einen Link zum vollständigen Quellcode, in dem Sie Anweisungen zur Einrichtung und Ausführung des Codes im Kontext finden.

Erste Schritte

Hello Amazon Cognito

Die folgenden Codebeispiele veranschaulichen die ersten Schritte mit Amazon Cognito.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
package main

import (
 "context"
 "fmt"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/config"
 "github.com/aws/aws-sdk-go-v2/service/cognitoidentityprovider"
 "github.com/aws/aws-sdk-go-v2/service/cognitoidentityprovider/types"
)

// main uses the AWS SDK for Go V2 to create an Amazon Simple Notification Service
// (Amazon SNS) client and list the topics in your account.
// This example uses the default settings specified in your shared credentials
// and config files.
func main() {
 ctx := context.Background()
 sdkConfig, err := config.LoadDefaultConfig(ctx)
 if err != nil {
  fmt.Println("Couldn't load default configuration. Have you set up your AWS
 account?")
  fmt.Println(err)
```

```
  return
 }
 cognitoClient := cognitoidentityprovider.NewFromConfig(sdkConfig)
 fmt.Println("Let's list the user pools for your account.")
 var pools []types.UserPoolDescriptionType
 paginator := cognitoidentityprovider.NewListUserPoolsPaginator(
  cognitoClient, &cognitoidentityprovider.ListUserPoolsInput{MaxResults:
 aws.Int32(10)})
 for paginator.HasMorePages() {
  output, err := paginator.NextPage(ctx)
  if err != nil {
   log.Printf("Couldn't get user pools. Here's why: %v\n", err)
  } else {
   pools = append(pools, output.UserPools...)
  }
 }
 if len(pools) == 0 {
  fmt.Println("You don't have any user pools!")
 } else {
  for _, pool := range pools {
   fmt.Printf("\t%v: %v\n", *pool.Name, *pool.Id)
  }
 }
}
```

- Einzelheiten zur API finden Sie ListUserPoolsin der AWS SDK für Go API-Referenz.

Themen

- Aktionen
- Szenarien

Aktionen

`AdminCreateUser`

Das folgende Codebeispiel zeigt die Verwendung`AdminCreateUser`.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import (
 "context"
 "errors"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/cognitoidentityprovider"
 "github.com/aws/aws-sdk-go-v2/service/cognitoidentityprovider/types"
)

type CognitoActions struct {
 CognitoClient *cognitoidentityprovider.Client
}



// AdminCreateUser uses administrator credentials to add a user to a user pool. This
 method leaves the user
// in a state that requires they enter a new password next time they sign in.
func (actor CognitoActions) AdminCreateUser(ctx context.Context, userPoolId string,
 userName string, userEmail string) error {
 _, err := actor.CognitoClient.AdminCreateUser(ctx,
 &cognitoidentityprovider.AdminCreateUserInput{
  UserPoolId:     aws.String(userPoolId),
  Username:       aws.String(userName),
  MessageAction:  types.MessageActionTypeSuppress,
  UserAttributes: []types.AttributeType{{Name: aws.String("email"), Value:
 aws.String(userEmail)}},
 })
 if err != nil {
  var userExists *types.UsernameExistsException
  if errors.As(err, &userExists) {
   log.Printf("User %v already exists in the user pool.", userName)
```

```
    err = nil
   } else {
    log.Printf("Couldn't create user %v. Here's why: %v\n", userName, err)
   }
  }
  return err
}
```

- Einzelheiten zur API finden Sie AdminCreateUserin der AWS SDK für Go API-Referenz.

`AdminSetUserPassword`

Das folgende Codebeispiel zeigt die Verwendung`AdminSetUserPassword`.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import (
 "context"
 "errors"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/cognitoidentityprovider"
 "github.com/aws/aws-sdk-go-v2/service/cognitoidentityprovider/types"
)

type CognitoActions struct {
 CognitoClient *cognitoidentityprovider.Client
}
```

```
// AdminSetUserPassword uses administrator credentials to set a password for a user
 without requiring a
// temporary password.
func (actor CognitoActions) AdminSetUserPassword(ctx context.Context, userPoolId
 string, userName string, password string) error {
 _, err := actor.CognitoClient.AdminSetUserPassword(ctx,
 &cognitoidentityprovider.AdminSetUserPasswordInput{
  Password:   aws.String(password),
  UserPoolId: aws.String(userPoolId),
  Username:   aws.String(userName),
  Permanent:  true,
 })
 if err != nil {
  var invalidPassword *types.InvalidPasswordException
  if errors.As(err, &invalidPassword) {
   log.Println(*invalidPassword.Message)
  } else {
   log.Printf("Couldn't set password for user %v. Here's why: %v\n", userName, err)
  }
 }
 return err
}
```

- Einzelheiten zur API finden Sie AdminSetUserPasswordin der AWS SDK für Go API-Referenz.

`ConfirmForgotPassword`

Das folgende Codebeispiel zeigt die Verwendung`ConfirmForgotPassword`.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import (
 "context"
```

```
 "errors"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/cognitoidentityprovider"
 "github.com/aws/aws-sdk-go-v2/service/cognitoidentityprovider/types"
)

type CognitoActions struct {
 CognitoClient *cognitoidentityprovider.Client
}



// ConfirmForgotPassword confirms a user with a confirmation code and a new
 password.
func (actor CognitoActions) ConfirmForgotPassword(ctx context.Context, clientId
 string, code string, userName string, password string) error {
 _, err := actor.CognitoClient.ConfirmForgotPassword(ctx,
 &cognitoidentityprovider.ConfirmForgotPasswordInput{
  ClientId:         aws.String(clientId),
  ConfirmationCode: aws.String(code),
  Password:         aws.String(password),
  Username:         aws.String(userName),
 })
 if err != nil {
  var invalidPassword *types.InvalidPasswordException
  if errors.As(err, &invalidPassword) {
   log.Println(*invalidPassword.Message)
  } else {
   log.Printf("Couldn't confirm user %v. Here's why: %v", userName, err)
  }
 }
 return err
}
```

- Einzelheiten zur API finden Sie ConfirmForgotPasswordin der AWS SDK für Go API-Referenz.

`DeleteUser`

Das folgende Codebeispiel zeigt die Verwendung`DeleteUser`.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import (
 "context"
 "errors"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/cognitoidentityprovider"
 "github.com/aws/aws-sdk-go-v2/service/cognitoidentityprovider/types"
)

type CognitoActions struct {
 CognitoClient *cognitoidentityprovider.Client
}



// DeleteUser removes a user from the user pool.
func (actor CognitoActions) DeleteUser(ctx context.Context, userAccessToken string)
 error {
 _, err := actor.CognitoClient.DeleteUser(ctx,
 &cognitoidentityprovider.DeleteUserInput{
  AccessToken: aws.String(userAccessToken),
 })
 if err != nil {
  log.Printf("Couldn't delete user. Here's why: %v\n", err)
 }
 return err
}
```

- Einzelheiten zur API finden Sie DeleteUserin der AWS SDK für Go API-Referenz.

`ForgotPassword`

Das folgende Codebeispiel zeigt die Verwendung`ForgotPassword`.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import (
 "context"
 "errors"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/cognitoidentityprovider"
 "github.com/aws/aws-sdk-go-v2/service/cognitoidentityprovider/types"
)

type CognitoActions struct {
 CognitoClient *cognitoidentityprovider.Client
}



// ForgotPassword starts a password recovery flow for a user. This flow typically
 sends a confirmation code
// to the user's configured notification destination, such as email.
func (actor CognitoActions) ForgotPassword(ctx context.Context, clientId string,
 userName string) (*types.CodeDeliveryDetailsType, error) {
 output, err := actor.CognitoClient.ForgotPassword(ctx,
 &cognitoidentityprovider.ForgotPasswordInput{
  ClientId: aws.String(clientId),
  Username: aws.String(userName),
 })
 if err != nil {
  log.Printf("Couldn't start password reset for user '%v'. Here;s why: %v\n",
 userName, err)
 }
```

```
 return output.CodeDeliveryDetails, err
}
```

- Einzelheiten zur API finden Sie ForgotPasswordin der AWS SDK für Go API-Referenz.

`InitiateAuth`

Das folgende Codebeispiel zeigt die Verwendung`InitiateAuth`.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import (
 "context"
 "errors"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/cognitoidentityprovider"
 "github.com/aws/aws-sdk-go-v2/service/cognitoidentityprovider/types"
)

type CognitoActions struct {
 CognitoClient *cognitoidentityprovider.Client
}



// SignIn signs in a user to Amazon Cognito using a username and password
 authentication flow.
func (actor CognitoActions) SignIn(ctx context.Context, clientId string, userName
 string, password string) (*types.AuthenticationResultType, error) {
 var authResult *types.AuthenticationResultType
```

```
 output, err := actor.CognitoClient.InitiateAuth(ctx,
 &cognitoidentityprovider.InitiateAuthInput{
  AuthFlow:       "USER_PASSWORD_AUTH",
  ClientId:       aws.String(clientId),
  AuthParameters: map[string]string{"USERNAME": userName, "PASSWORD": password},
 })
 if err != nil {
  var resetRequired *types.PasswordResetRequiredException
  if errors.As(err, &resetRequired) {
   log.Println(*resetRequired.Message)
  } else {
   log.Printf("Couldn't sign in user %v. Here's why: %v\n", userName, err)
  }
 } else {
  authResult = output.AuthenticationResult
 }
 return authResult, err
}
```

- Einzelheiten zur API finden Sie InitiateAuthin der AWS SDK für Go API-Referenz.

`ListUserPools`

Das folgende Codebeispiel zeigt die Verwendung`ListUserPools`.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
package main

import (
 "context"
 "fmt"
 "log"
```

```
 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/config"
 "github.com/aws/aws-sdk-go-v2/service/cognitoidentityprovider"
 "github.com/aws/aws-sdk-go-v2/service/cognitoidentityprovider/types"
)

// main uses the AWS SDK for Go V2 to create an Amazon Simple Notification Service
// (Amazon SNS) client and list the topics in your account.
// This example uses the default settings specified in your shared credentials
// and config files.
func main() {
 ctx := context.Background()
 sdkConfig, err := config.LoadDefaultConfig(ctx)
 if err != nil {
  fmt.Println("Couldn't load default configuration. Have you set up your AWS
 account?")
  fmt.Println(err)
  return
 }
 cognitoClient := cognitoidentityprovider.NewFromConfig(sdkConfig)
 fmt.Println("Let's list the user pools for your account.")
 var pools []types.UserPoolDescriptionType
 paginator := cognitoidentityprovider.NewListUserPoolsPaginator(
  cognitoClient, &cognitoidentityprovider.ListUserPoolsInput{MaxResults:
 aws.Int32(10)})
 for paginator.HasMorePages() {
  output, err := paginator.NextPage(ctx)
  if err != nil {
   log.Printf("Couldn't get user pools. Here's why: %v\n", err)
  } else {
   pools = append(pools, output.UserPools...)
  }
 }
 if len(pools) == 0 {
  fmt.Println("You don't have any user pools!")
 } else {
  for _, pool := range pools {
   fmt.Printf("\t%v: %v\n", *pool.Name, *pool.Id)
  }
 }
}
```

- Einzelheiten zur API finden Sie ListUserPoolsin der AWS SDK für Go API-Referenz.

`SignUp`

Das folgende Codebeispiel zeigt die Verwendung`SignUp`.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import (
 "context"
 "errors"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/cognitoidentityprovider"
 "github.com/aws/aws-sdk-go-v2/service/cognitoidentityprovider/types"
)

type CognitoActions struct {
 CognitoClient *cognitoidentityprovider.Client
}



// SignUp signs up a user with Amazon Cognito.
func (actor CognitoActions) SignUp(ctx context.Context, clientId string, userName
 string, password string, userEmail string) (bool, error) {
 confirmed := false
 output, err := actor.CognitoClient.SignUp(ctx,
 &cognitoidentityprovider.SignUpInput{
  ClientId: aws.String(clientId),
  Password: aws.String(password),
  Username: aws.String(userName),
```

```
  UserAttributes: []types.AttributeType{
   {Name: aws.String("email"), Value: aws.String(userEmail)},
  },
 })
 if err != nil {
  var invalidPassword *types.InvalidPasswordException
  if errors.As(err, &invalidPassword) {
   log.Println(*invalidPassword.Message)
  } else {
   log.Printf("Couldn't sign up user %v. Here's why: %v\n", userName, err)
  }
 } else {
  confirmed = output.UserConfirmed
 }
 return confirmed, err
}
```

- Einzelheiten zur API finden Sie SignUpin der AWS SDK für Go API-Referenz.

`UpdateUserPool`

Das folgende Codebeispiel zeigt die VerwendungUpdateUserPool.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import (
 "context"
 "errors"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/cognitoidentityprovider"
 "github.com/aws/aws-sdk-go-v2/service/cognitoidentityprovider/types"
```

```
)

type CognitoActions struct {
 CognitoClient *cognitoidentityprovider.Client
}



// Trigger and TriggerInfo define typed data for updating an Amazon Cognito trigger.
type Trigger int

const (
 PreSignUp Trigger = iota
 UserMigration
 PostAuthentication
)

type TriggerInfo struct {
 Trigger    Trigger
 HandlerArn *string
}

// UpdateTriggers adds or removes Lambda triggers for a user pool. When a trigger is
 specified with a `nil` value,
// it is removed from the user pool.
func (actor CognitoActions) UpdateTriggers(ctx context.Context, userPoolId string,
 triggers ...TriggerInfo) error {
 output, err := actor.CognitoClient.DescribeUserPool(ctx,
 &cognitoidentityprovider.DescribeUserPoolInput{
  UserPoolId: aws.String(userPoolId),
 })
 if err != nil {
  log.Printf("Couldn't get info about user pool %v. Here's why: %v\n", userPoolId,
 err)
  return err
 }
 lambdaConfig := output.UserPool.LambdaConfig
 for _, trigger := range triggers {
  switch trigger.Trigger {
  case PreSignUp:
   lambdaConfig.PreSignUp = trigger.HandlerArn
  case UserMigration:
   lambdaConfig.UserMigration = trigger.HandlerArn
  case PostAuthentication:
```

```
   lambdaConfig.PostAuthentication = trigger.HandlerArn
  }
 }
 _, err = actor.CognitoClient.UpdateUserPool(ctx,
 &cognitoidentityprovider.UpdateUserPoolInput{
  UserPoolId:   aws.String(userPoolId),
  LambdaConfig: lambdaConfig,
 })
 if err != nil {
  log.Printf("Couldn't update user pool %v. Here's why: %v\n", userPoolId, err)
 }
 return err
}
```

- Einzelheiten zur API finden Sie UpdateUserPoolin der AWS SDK für Go API-Referenz.

Szenarien

Bestätigen Sie automatisch bekannte Benutzer mit einer Lambda-Funktion

Das folgende Codebeispiel zeigt, wie bekannte Amazon Cognito Cognito-Benutzer automatisch mit einer Lambda-Funktion bestätigt werden.

- Konfigurieren Sie einen Benutzerpool, um eine Lambda-Funktion für den `PreSignUp`-Trigger aufzurufen.
- Registrieren eines Benutzers bei Amazon Cognito.
- Die Lambda-Funktion scannt eine DynamoDB-Tabelle und bestätigt automatisch bekannte Benutzer.
- Melden Sie sich als neuer Benutzer an und bereinigen Sie dann die Ressourcen.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Führen Sie ein interaktives Szenario an einem Prompt aus.

```
import (
 "context"
 "errors"
 "log"
 "strings"
 "user_pools_and_lambda_triggers/actions"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/cognitoidentityprovider"
 "github.com/aws/aws-sdk-go-v2/service/cognitoidentityprovider/types"
 "github.com/awsdocs/aws-doc-sdk-examples/gov2/demotools"
)

// AutoConfirm separates the steps of this scenario into individual functions so
 that
// they are simpler to read and understand.
type AutoConfirm struct {
 helper       IScenarioHelper
 questioner   demotools.IQuestioner
 resources    Resources
 cognitoActor *actions.CognitoActions
}

// NewAutoConfirm constructs a new auto confirm runner.
func NewAutoConfirm(sdkConfig aws.Config, questioner demotools.IQuestioner, helper
 IScenarioHelper) AutoConfirm {
 scenario := AutoConfirm{
  helper:       helper,
  questioner:   questioner,
  resources:    Resources{},
  cognitoActor: &actions.CognitoActions{CognitoClient:
 cognitoidentityprovider.NewFromConfig(sdkConfig)},
 }
 scenario.resources.init(scenario.cognitoActor, questioner)
 return scenario
}

// AddPreSignUpTrigger adds a Lambda handler as an invocation target for the
 PreSignUp trigger.
func (runner *AutoConfirm) AddPreSignUpTrigger(ctx context.Context, userPoolId
 string, functionArn string) {
```

```
 log.Printf("Let's add a Lambda function to handle the PreSignUp trigger from
 Cognito.\n" +
  "This trigger happens when a user signs up, and lets your function take action
 before the main Cognito\n" +
  "sign up processing occurs.\n")
 err := runner.cognitoActor.UpdateTriggers(
  ctx, userPoolId,
  actions.TriggerInfo{Trigger: actions.PreSignUp, HandlerArn:
 aws.String(functionArn)})
 if err != nil {
  panic(err)
 }
 log.Printf("Lambda function %v added to user pool %v to handle the PreSignUp
 trigger.\n",
  functionArn, userPoolId)
}

// SignUpUser signs up a user from the known user table with a password you specify.
func (runner *AutoConfirm) SignUpUser(ctx context.Context, clientId string,
 usersTable string) (string, string) {
 log.Println("Let's sign up a user to your Cognito user pool. When the user's email
 matches an email in the\n" +
  "DynamoDB known users table, it is automatically verified and the user is
 confirmed.")

 knownUsers, err := runner.helper.GetKnownUsers(ctx, usersTable)
 if err != nil {
  panic(err)
 }
 userChoice := runner.questioner.AskChoice("Which user do you want to use?\n",
 knownUsers.UserNameList())
 user := knownUsers.Users[userChoice]

 var signedUp bool
 var userConfirmed bool
 password := runner.questioner.AskPassword("Enter a password that has at least eight
 characters, uppercase, lowercase, numbers and symbols.\n"+
  "(the password will not display as you type):", 8)
 for !signedUp {
  log.Printf("Signing up user '%v' with email '%v' to Cognito.\n", user.UserName,
 user.UserEmail)
  userConfirmed, err = runner.cognitoActor.SignUp(ctx, clientId, user.UserName,
 password, user.UserEmail)
  if err != nil {
```

```
    var invalidPassword *types.InvalidPasswordException
    if errors.As(err, &invalidPassword) {
     password = runner.questioner.AskPassword("Enter another password:", 8)
    } else {
     panic(err)
    }
   } else {
    signedUp = true
   }
  }
  log.Printf("User %v signed up, confirmed = %v.\n", user.UserName, userConfirmed)

  log.Println(strings.Repeat("-", 88))

  return user.UserName, password
}

// SignInUser signs in a user.
func (runner *AutoConfirm) SignInUser(ctx context.Context, clientId string, userName
 string, password string) string {
 runner.questioner.Ask("Press Enter when you're ready to continue.")
 log.Printf("Let's sign in as %v...\n", userName)
 authResult, err := runner.cognitoActor.SignIn(ctx, clientId, userName, password)
 if err != nil {
  panic(err)
 }
 log.Printf("Successfully signed in. Your access token starts with: %v...\n",
 (*authResult.AccessToken)[:10])
 log.Println(strings.Repeat("-", 88))
 return *authResult.AccessToken
}

// Run runs the scenario.
func (runner *AutoConfirm) Run(ctx context.Context, stackName string) {
 defer func() {
  if r := recover(); r != nil {
   log.Println("Something went wrong with the demo.")
   runner.resources.Cleanup(ctx)
  }
 }()

 log.Println(strings.Repeat("-", 88))
 log.Printf("Welcome\n")
```

```
 log.Println(strings.Repeat("-", 88))

 stackOutputs, err := runner.helper.GetStackOutputs(ctx, stackName)
 if err != nil {
  panic(err)
 }
 runner.resources.userPoolId = stackOutputs["UserPoolId"]
 runner.helper.PopulateUserTable(ctx, stackOutputs["TableName"])

 runner.AddPreSignUpTrigger(ctx, stackOutputs["UserPoolId"],
 stackOutputs["AutoConfirmFunctionArn"])
 runner.resources.triggers = append(runner.resources.triggers, actions.PreSignUp)
 userName, password := runner.SignUpUser(ctx, stackOutputs["UserPoolClientId"],
 stackOutputs["TableName"])
 runner.helper.ListRecentLogEvents(ctx, stackOutputs["AutoConfirmFunction"])
 runner.resources.userAccessTokens = append(runner.resources.userAccessTokens,
  runner.SignInUser(ctx, stackOutputs["UserPoolClientId"], userName, password))

 runner.resources.Cleanup(ctx)

 log.Println(strings.Repeat("-", 88))
 log.Println("Thanks for watching!")
 log.Println(strings.Repeat("-", 88))
}
```

Behandeln Sie den `PreSignUp`-Trigger mit einer Lambda-Funktion.

```
import (
 "context"
 "log"
 "os"

 "github.com/aws/aws-lambda-go/events"
 "github.com/aws/aws-lambda-go/lambda"
 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/config"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 dynamodbtypes "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)
```

```
const TABLE_NAME = "TABLE_NAME"

// UserInfo defines structured user data that can be marshalled to a DynamoDB
 format.
type UserInfo struct {
 UserName  string `dynamodbav:"UserName"`
 UserEmail string `dynamodbav:"UserEmail"`
}

// GetKey marshals the user email value to a DynamoDB key format.
func (user UserInfo) GetKey() map[string]dynamodbtypes.AttributeValue {
 userEmail, err := attributevalue.Marshal(user.UserEmail)
 if err != nil {
  panic(err)
 }
 return map[string]dynamodbtypes.AttributeValue{"UserEmail": userEmail}
}

type handler struct {
 dynamoClient *dynamodb.Client
}

// HandleRequest handles the PreSignUp event by looking up a user in an Amazon
 DynamoDB table and
// specifying whether they should be confirmed and verified.
func (h *handler) HandleRequest(ctx context.Context, event
 events.CognitoEventUserPoolsPreSignup) (events.CognitoEventUserPoolsPreSignup,
 error) {
 log.Printf("Received presignup from %v for user '%v'", event.TriggerSource,
 event.UserName)
 if event.TriggerSource != "PreSignUp_SignUp" {
  // Other trigger sources, such as PreSignUp_AdminInitiateAuth, ignore the response
 from this handler.
  return event, nil
 }
 tableName := os.Getenv(TABLE_NAME)
 user := UserInfo{
  UserEmail: event.Request.UserAttributes["email"],
 }
 log.Printf("Looking up email %v in table %v.\n", user.UserEmail, tableName)
 output, err := h.dynamoClient.GetItem(ctx, &dynamodb.GetItemInput{
  Key:       user.GetKey(),
  TableName: aws.String(tableName),
```

```
 })
 if err != nil {
  log.Printf("Error looking up email %v.\n", user.UserEmail)
  return event, err
 }
 if output.Item == nil {
  log.Printf("Email %v not found. Email verification is required.\n",
 user.UserEmail)
  return event, err
 }

 err = attributevalue.UnmarshalMap(output.Item, &user)
 if err != nil {
  log.Printf("Couldn't unmarshal DynamoDB item. Here's why: %v\n", err)
  return event, err
 }

 if user.UserName != event.UserName {
  log.Printf("UserEmail %v found, but stored UserName '%v' does not match supplied
 UserName '%v'. Verification is required.\n",
   user.UserEmail, user.UserName, event.UserName)
 } else {
  log.Printf("UserEmail %v found with matching UserName %v. User is confirmed.\n",
 user.UserEmail, user.UserName)
  event.Response.AutoConfirmUser = true
  event.Response.AutoVerifyEmail = true
 }

 return event, err
}

func main() {
 ctx := context.Background()
 sdkConfig, err := config.LoadDefaultConfig(ctx)
 if err != nil {
  log.Panicln(err)
 }
 h := handler{
  dynamoClient: dynamodb.NewFromConfig(sdkConfig),
 }
 lambda.Start(h.HandleRequest)
}
```

Erstellen Sie eine Struktur, die allgemeine Aufgaben ausführt.

```
import (
 "context"
 "log"
 "strings"
 "time"
 "user_pools_and_lambda_triggers/actions"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/cloudformation"
 "github.com/aws/aws-sdk-go-v2/service/cloudwatchlogs"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/awsdocs/aws-doc-sdk-examples/gov2/demotools"
)

// IScenarioHelper defines common functions used by the workflows in this example.
type IScenarioHelper interface {
 Pause(secs int)
 GetStackOutputs(ctx context.Context, stackName string) (actions.StackOutputs,
 error)
 PopulateUserTable(ctx context.Context, tableName string)
 GetKnownUsers(ctx context.Context, tableName string) (actions.UserList, error)
 AddKnownUser(ctx context.Context, tableName string, user actions.User)
 ListRecentLogEvents(ctx context.Context, functionName string)
}

// ScenarioHelper contains AWS wrapper structs used by the workflows in this
 example.
type ScenarioHelper struct {
 questioner  demotools.IQuestioner
 dynamoActor *actions.DynamoActions
 cfnActor    *actions.CloudFormationActions
 cwlActor    *actions.CloudWatchLogsActions
 isTestRun   bool
}

// NewScenarioHelper constructs a new scenario helper.
func NewScenarioHelper(sdkConfig aws.Config, questioner demotools.IQuestioner)
 ScenarioHelper {
```

```
 scenario := ScenarioHelper{
  questioner:  questioner,
  dynamoActor: &actions.DynamoActions{DynamoClient:
 dynamodb.NewFromConfig(sdkConfig)},
  cfnActor:    &actions.CloudFormationActions{CfnClient:
 cloudformation.NewFromConfig(sdkConfig)},
  cwlActor:    &actions.CloudWatchLogsActions{CwlClient:
 cloudwatchlogs.NewFromConfig(sdkConfig)},
 }
 return scenario
}

// Pause waits for the specified number of seconds.
func (helper ScenarioHelper) Pause(secs int) {
 if !helper.isTestRun {
  time.Sleep(time.Duration(secs) * time.Second)
 }
}

// GetStackOutputs gets the outputs from the specified CloudFormation stack in a
 structured format.
func (helper ScenarioHelper) GetStackOutputs(ctx context.Context, stackName string)
 (actions.StackOutputs, error) {
 return helper.cfnActor.GetOutputs(ctx, stackName), nil
}

// PopulateUserTable fills the known user table with example data.
func (helper ScenarioHelper) PopulateUserTable(ctx context.Context, tableName
 string) {
 log.Printf("First, let's add some users to the DynamoDB %v table we'll use for this
 example.\n", tableName)
 err := helper.dynamoActor.PopulateTable(ctx, tableName)
 if err != nil {
  panic(err)
 }
}

// GetKnownUsers gets the users from the known users table in a structured format.
func (helper ScenarioHelper) GetKnownUsers(ctx context.Context, tableName string)
 (actions.UserList, error) {
 knownUsers, err := helper.dynamoActor.Scan(ctx, tableName)
 if err != nil {
  log.Printf("Couldn't get known users from table %v. Here's why: %v\n", tableName,
 err)
```

```
 }
 return knownUsers, err
}

// AddKnownUser adds a user to the known users table.
func (helper ScenarioHelper) AddKnownUser(ctx context.Context, tableName string,
 user actions.User) {
 log.Printf("Adding user '%v' with email '%v' to the DynamoDB known users table...
\n",
  user.UserName, user.UserEmail)
 err := helper.dynamoActor.AddUser(ctx, tableName, user)
 if err != nil {
  panic(err)
 }
}

// ListRecentLogEvents gets the most recent log stream and events for the specified
 Lambda function and displays them.
func (helper ScenarioHelper) ListRecentLogEvents(ctx context.Context, functionName
 string) {
 log.Println("Waiting a few seconds to let Lambda write to CloudWatch Logs...")
 helper.Pause(10)
 log.Println("Okay, let's check the logs to find what's happened recently with your
 Lambda function.")
 logStream, err := helper.cwlActor.GetLatestLogStream(ctx, functionName)
 if err != nil {
  panic(err)
 }
 log.Printf("Getting some recent events from log stream %v\n",
 *logStream.LogStreamName)
 events, err := helper.cwlActor.GetLogEvents(ctx, functionName,
 *logStream.LogStreamName, 10)
 if err != nil {
  panic(err)
 }
 for _, event := range events {
  log.Printf("\t%v", *event.Message)
 }
 log.Println(strings.Repeat("-", 88))
}
```

Erstellen Sie eine Struktur, die Amazon Cognito-Aktionen umschließt.

```
import (
 "context"
 "errors"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/cognitoidentityprovider"
 "github.com/aws/aws-sdk-go-v2/service/cognitoidentityprovider/types"
)

type CognitoActions struct {
 CognitoClient *cognitoidentityprovider.Client
}



// Trigger and TriggerInfo define typed data for updating an Amazon Cognito trigger.
type Trigger int

const (
 PreSignUp Trigger = iota
 UserMigration
 PostAuthentication
)

type TriggerInfo struct {
 Trigger    Trigger
 HandlerArn *string
}

// UpdateTriggers adds or removes Lambda triggers for a user pool. When a trigger is
 specified with a `nil` value,
// it is removed from the user pool.
func (actor CognitoActions) UpdateTriggers(ctx context.Context, userPoolId string,
 triggers ...TriggerInfo) error {
 output, err := actor.CognitoClient.DescribeUserPool(ctx,
 &cognitoidentityprovider.DescribeUserPoolInput{
  UserPoolId: aws.String(userPoolId),
 })
 if err != nil {
```

```
  log.Printf("Couldn't get info about user pool %v. Here's why: %v\n", userPoolId,
 err)
  return err
 }
 lambdaConfig := output.UserPool.LambdaConfig
 for _, trigger := range triggers {
  switch trigger.Trigger {
  case PreSignUp:
   lambdaConfig.PreSignUp = trigger.HandlerArn
  case UserMigration:
   lambdaConfig.UserMigration = trigger.HandlerArn
  case PostAuthentication:
   lambdaConfig.PostAuthentication = trigger.HandlerArn
  }
 }
 _, err = actor.CognitoClient.UpdateUserPool(ctx,
 &cognitoidentityprovider.UpdateUserPoolInput{
  UserPoolId:   aws.String(userPoolId),
  LambdaConfig: lambdaConfig,
 })
 if err != nil {
  log.Printf("Couldn't update user pool %v. Here's why: %v\n", userPoolId, err)
 }
 return err
}



// SignUp signs up a user with Amazon Cognito.
func (actor CognitoActions) SignUp(ctx context.Context, clientId string, userName
 string, password string, userEmail string) (bool, error) {
 confirmed := false
 output, err := actor.CognitoClient.SignUp(ctx,
 &cognitoidentityprovider.SignUpInput{
  ClientId: aws.String(clientId),
  Password: aws.String(password),
  Username: aws.String(userName),
  UserAttributes: []types.AttributeType{
   {Name: aws.String("email"), Value: aws.String(userEmail)},
  },
 })
 if err != nil {
  var invalidPassword *types.InvalidPasswordException
  if errors.As(err, &invalidPassword) {
```

```
   log.Println(*invalidPassword.Message)
  } else {
   log.Printf("Couldn't sign up user %v. Here's why: %v\n", userName, err)
  }
 } else {
  confirmed = output.UserConfirmed
 }
 return confirmed, err
}



// SignIn signs in a user to Amazon Cognito using a username and password
 authentication flow.
func (actor CognitoActions) SignIn(ctx context.Context, clientId string, userName
 string, password string) (*types.AuthenticationResultType, error) {
 var authResult *types.AuthenticationResultType
 output, err := actor.CognitoClient.InitiateAuth(ctx,
 &cognitoidentityprovider.InitiateAuthInput{
  AuthFlow:       "USER_PASSWORD_AUTH",
  ClientId:       aws.String(clientId),
  AuthParameters: map[string]string{"USERNAME": userName, "PASSWORD": password},
 })
 if err != nil {
  var resetRequired *types.PasswordResetRequiredException
  if errors.As(err, &resetRequired) {
   log.Println(*resetRequired.Message)
  } else {
   log.Printf("Couldn't sign in user %v. Here's why: %v\n", userName, err)
  }
 } else {
  authResult = output.AuthenticationResult
 }
 return authResult, err
}



// ForgotPassword starts a password recovery flow for a user. This flow typically
 sends a confirmation code
// to the user's configured notification destination, such as email.
func (actor CognitoActions) ForgotPassword(ctx context.Context, clientId string,
 userName string) (*types.CodeDeliveryDetailsType, error) {
```

```
 output, err := actor.CognitoClient.ForgotPassword(ctx,
 &cognitoidentityprovider.ForgotPasswordInput{
  ClientId: aws.String(clientId),
  Username: aws.String(userName),
 })
 if err != nil {
  log.Printf("Couldn't start password reset for user '%v'. Here;s why: %v\n",
 userName, err)
 }
 return output.CodeDeliveryDetails, err
}



// ConfirmForgotPassword confirms a user with a confirmation code and a new
 password.
func (actor CognitoActions) ConfirmForgotPassword(ctx context.Context, clientId
 string, code string, userName string, password string) error {
 _, err := actor.CognitoClient.ConfirmForgotPassword(ctx,
 &cognitoidentityprovider.ConfirmForgotPasswordInput{
  ClientId:         aws.String(clientId),
  ConfirmationCode: aws.String(code),
  Password:         aws.String(password),
  Username:         aws.String(userName),
 })
 if err != nil {
  var invalidPassword *types.InvalidPasswordException
  if errors.As(err, &invalidPassword) {
   log.Println(*invalidPassword.Message)
  } else {
   log.Printf("Couldn't confirm user %v. Here's why: %v", userName, err)
  }
 }
 return err
}



// DeleteUser removes a user from the user pool.
func (actor CognitoActions) DeleteUser(ctx context.Context, userAccessToken string)
 error {
 _, err := actor.CognitoClient.DeleteUser(ctx,
 &cognitoidentityprovider.DeleteUserInput{
  AccessToken: aws.String(userAccessToken),
```

```
 })
 if err != nil {
  log.Printf("Couldn't delete user. Here's why: %v\n", err)
 }
 return err
}



// AdminCreateUser uses administrator credentials to add a user to a user pool. This
 method leaves the user
// in a state that requires they enter a new password next time they sign in.
func (actor CognitoActions) AdminCreateUser(ctx context.Context, userPoolId string,
 userName string, userEmail string) error {
 _, err := actor.CognitoClient.AdminCreateUser(ctx,
 &cognitoidentityprovider.AdminCreateUserInput{
  UserPoolId:     aws.String(userPoolId),
  Username:       aws.String(userName),
  MessageAction:  types.MessageActionTypeSuppress,
  UserAttributes: []types.AttributeType{{Name: aws.String("email"), Value:
 aws.String(userEmail)}},
 })
 if err != nil {
  var userExists *types.UsernameExistsException
  if errors.As(err, &userExists) {
   log.Printf("User %v already exists in the user pool.", userName)
   err = nil
  } else {
   log.Printf("Couldn't create user %v. Here's why: %v\n", userName, err)
  }
 }
 return err
}



// AdminSetUserPassword uses administrator credentials to set a password for a user
 without requiring a
// temporary password.
func (actor CognitoActions) AdminSetUserPassword(ctx context.Context, userPoolId
 string, userName string, password string) error {
 _, err := actor.CognitoClient.AdminSetUserPassword(ctx,
 &cognitoidentityprovider.AdminSetUserPasswordInput{
  Password:   aws.String(password),
```

```
  UserPoolId: aws.String(userPoolId),
  Username:   aws.String(userName),
  Permanent:  true,
 })
 if err != nil {
  var invalidPassword *types.InvalidPasswordException
  if errors.As(err, &invalidPassword) {
   log.Println(*invalidPassword.Message)
  } else {
   log.Printf("Couldn't set password for user %v. Here's why: %v\n", userName, err)
  }
 }
 return err
}
```

Erstellen Sie eine Struktur, die DynamoDB-Aktionen umhüllt.

```
import (
 "context"
 "fmt"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// DynamoActions encapsulates the Amazon Simple Notification Service (Amazon SNS)
 actions
// used in the examples.
type DynamoActions struct {
 DynamoClient *dynamodb.Client
}

// User defines structured user data.
type User struct {
 UserName  string
 UserEmail string
 LastLogin *LoginInfo `dynamodbav:",omitempty"`
```

```
}

// LoginInfo defines structured custom login data.
type LoginInfo struct {
 UserPoolId string
 ClientId   string
 Time       string
}

// UserList defines a list of users.
type UserList struct {
 Users []User
}

// UserNameList returns the usernames contained in a UserList as a list of strings.
func (users *UserList) UserNameList() []string {
 names := make([]string, len(users.Users))
 for i := 0; i < len(users.Users); i++ {
  names[i] = users.Users[i].UserName
 }
 return names
}

// PopulateTable adds a set of test users to the table.
func (actor DynamoActions) PopulateTable(ctx context.Context, tableName string)
 error {
 var err error
 var item map[string]types.AttributeValue
 var writeReqs []types.WriteRequest
 for i := 1; i < 4; i++ {
  item, err = attributevalue.MarshalMap(User{UserName: fmt.Sprintf("test_user_%v",
 i), UserEmail: fmt.Sprintf("test_email_%v@example.com", i)})
  if err != nil {
   log.Printf("Couldn't marshall user into DynamoDB format. Here's why: %v\n", err)
   return err
  }
  writeReqs = append(writeReqs, types.WriteRequest{PutRequest:
 &types.PutRequest{Item: item}})
 }
 _, err = actor.DynamoClient.BatchWriteItem(ctx, &dynamodb.BatchWriteItemInput{
  RequestItems: map[string][]types.WriteRequest{tableName: writeReqs},
 })
 if err != nil {
```

```
  log.Printf("Couldn't populate table %v with users. Here's why: %v\n", tableName,
 err)
 }
 return err
}

// Scan scans the table for all items.
func (actor DynamoActions) Scan(ctx context.Context, tableName string) (UserList,
 error) {
 var userList UserList
 output, err := actor.DynamoClient.Scan(ctx, &dynamodb.ScanInput{
  TableName: aws.String(tableName),
 })
 if err != nil {
  log.Printf("Couldn't scan table %v for items. Here's why: %v\n", tableName, err)
 } else {
  err = attributevalue.UnmarshalListOfMaps(output.Items, &userList.Users)
  if err != nil {
   log.Printf("Couldn't unmarshal items into users. Here's why: %v\n", err)
  }
 }
 return userList, err
}

// AddUser adds a user item to a table.
func (actor DynamoActions) AddUser(ctx context.Context, tableName string, user User)
 error {
 userItem, err := attributevalue.MarshalMap(user)
 if err != nil {
  log.Printf("Couldn't marshall user to item. Here's why: %v\n", err)
 }
 _, err = actor.DynamoClient.PutItem(ctx, &dynamodb.PutItemInput{
  Item:      userItem,
  TableName: aws.String(tableName),
 })
 if err != nil {
  log.Printf("Couldn't put item in table %v. Here's why: %v", tableName, err)
 }
 return err
}
```

Erstellen Sie eine Struktur, die CloudWatch Logs-Aktionen umschließt.

```
import (
 "context"
 "fmt"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/cloudwatchlogs"
 "github.com/aws/aws-sdk-go-v2/service/cloudwatchlogs/types"
)

type CloudWatchLogsActions struct {
 CwlClient *cloudwatchlogs.Client
}

// GetLatestLogStream gets the most recent log stream for a Lambda function.
func (actor CloudWatchLogsActions) GetLatestLogStream(ctx context.Context,
 functionName string) (types.LogStream, error) {
 var logStream types.LogStream
 logGroupName := fmt.Sprintf("/aws/lambda/%s", functionName)
 output, err := actor.CwlClient.DescribeLogStreams(ctx,
 &cloudwatchlogs.DescribeLogStreamsInput{
  Descending:   aws.Bool(true),
  Limit:        aws.Int32(1),
  LogGroupName: aws.String(logGroupName),
  OrderBy:      types.OrderByLastEventTime,
 })
 if err != nil {
  log.Printf("Couldn't get log streams for log group %v. Here's why: %v\n",
 logGroupName, err)
 } else {
  logStream = output.LogStreams[0]
 }
 return logStream, err
}

// GetLogEvents gets the most recent eventCount events from the specified log
 stream.
func (actor CloudWatchLogsActions) GetLogEvents(ctx context.Context, functionName
 string, logStreamName string, eventCount int32) (
 []types.OutputLogEvent, error) {
 var events []types.OutputLogEvent
```

```
 logGroupName := fmt.Sprintf("/aws/lambda/%s", functionName)
 output, err := actor.CwlClient.GetLogEvents(ctx, &cloudwatchlogs.GetLogEventsInput{
  LogStreamName: aws.String(logStreamName),
  Limit:         aws.Int32(eventCount),
  LogGroupName:  aws.String(logGroupName),
 })
 if err != nil {
  log.Printf("Couldn't get log event for log stream %v. Here's why: %v\n",
 logStreamName, err)
 } else {
  events = output.Events
 }
 return events, err
}
```

Erstellen Sie eine Struktur, die Aktionen umschließt. AWS CloudFormation

```
import (
 "context"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/cloudformation"
)

// StackOutputs defines a map of outputs from a specific stack.
type StackOutputs map[string]string

type CloudFormationActions struct {
 CfnClient *cloudformation.Client
}

// GetOutputs gets the outputs from a CloudFormation stack and puts them into a
 structured format.
func (actor CloudFormationActions) GetOutputs(ctx context.Context, stackName string)
 StackOutputs {
 output, err := actor.CfnClient.DescribeStacks(ctx,
 &cloudformation.DescribeStacksInput{
  StackName: aws.String(stackName),
 })
```

```
 if err != nil || len(output.Stacks) == 0 {
  log.Panicf("Couldn't find a CloudFormation stack named %v. Here's why: %v\n",
 stackName, err)
 }
 stackOutputs := StackOutputs{}
 for _, out := range output.Stacks[0].Outputs {
  stackOutputs[*out.OutputKey] = *out.OutputValue
 }
 return stackOutputs
}
```

Bereinigen von Ressourcen.

```
import (
 "context"
 "log"
 "user_pools_and_lambda_triggers/actions"

 "github.com/awsdocs/aws-doc-sdk-examples/gov2/demotools"
)

// Resources keeps track of AWS resources created during an example and handles
// cleanup when the example finishes.
type Resources struct {
 userPoolId       string
 userAccessTokens []string
 triggers         []actions.Trigger

 cognitoActor *actions.CognitoActions
 questioner   demotools.IQuestioner
}

func (resources *Resources) init(cognitoActor *actions.CognitoActions, questioner
 demotools.IQuestioner) {
 resources.userAccessTokens = []string{}
 resources.triggers = []actions.Trigger{}
 resources.cognitoActor = cognitoActor
 resources.questioner = questioner
}
```

```
// Cleanup deletes all AWS resources created during an example.
func (resources *Resources) Cleanup(ctx context.Context) {
 defer func() {
  if r := recover(); r != nil {
   log.Printf("Something went wrong during cleanup.\n%v\n", r)
   log.Println("Use the AWS Management Console to remove any remaining resources \n"
 +
    "that were created for this scenario.")
  }
 }()

 wantDelete := resources.questioner.AskBool("Do you want to remove all of the AWS
 resources that were created "+
  "during this demo (y/n)?", "y")
 if wantDelete {
  for _, accessToken := range resources.userAccessTokens {
   err := resources.cognitoActor.DeleteUser(ctx, accessToken)
   if err != nil {
    log.Println("Couldn't delete user during cleanup.")
    panic(err)
   }
   log.Println("Deleted user.")
  }
  triggerList := make([]actions.TriggerInfo, len(resources.triggers))
  for i := 0; i < len(resources.triggers); i++ {
   triggerList[i] = actions.TriggerInfo{Trigger: resources.triggers[i], HandlerArn:
 nil}
  }
  err := resources.cognitoActor.UpdateTriggers(ctx, resources.userPoolId,
 triggerList...)
  if err != nil {
   log.Println("Couldn't update Cognito triggers during cleanup.")
   panic(err)
  }
  log.Println("Removed Cognito triggers from user pool.")
 } else {
  log.Println("Be sure to remove resources when you're done with them to avoid
 unexpected charges!")
 }
}
```

- API-Details finden Sie in den folgenden Themen der AWS SDK für Go -API-Referenz.

- DeleteUser
- InitiateAuth
- SignUp
- UpdateUserPool

Migrieren Sie automatisch bekannte Benutzer mit einer Lambda-Funktion

Das folgende Codebeispiel zeigt, wie bekannte Amazon Cognito-Benutzer mit einer Lambda-Funktion automatisch migriert werden können.

- Konfigurieren Sie einen Benutzerpool, um eine Lambda-Funktion für den `MigrateUser`-Trigger aufzurufen.
- Melden Sie sich bei Amazon Cognito mit einem Benutzernamen und einer E-Mail-Adresse an, die sich nicht im Benutzerpool befinden.
- Die Lambda-Funktion durchsucht eine DynamoDB-Tabelle und migriert bekannte Benutzer automatisch in den Benutzerpool.
- Führen Sie den Ablauf „Passwort vergessen“ aus, um das Passwort für den migrierten Benutzer zurückzusetzen.
- Melden Sie sich als neuer Benutzer an und bereinigen Sie dann die Ressourcen.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Führen Sie ein interaktives Szenario an einem Prompt aus.

```
import (
 "context"
 "errors"
 "fmt"
 "log"
 "strings"
```

```
 "user_pools_and_lambda_triggers/actions"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/cognitoidentityprovider"
 "github.com/aws/aws-sdk-go-v2/service/cognitoidentityprovider/types"
 "github.com/awsdocs/aws-doc-sdk-examples/gov2/demotools"
)

// MigrateUser separates the steps of this scenario into individual functions so
 that
// they are simpler to read and understand.
type MigrateUser struct {
 helper       IScenarioHelper
 questioner   demotools.IQuestioner
 resources    Resources
 cognitoActor *actions.CognitoActions
}

// NewMigrateUser constructs a new migrate user runner.
func NewMigrateUser(sdkConfig aws.Config, questioner demotools.IQuestioner, helper
 IScenarioHelper) MigrateUser {
 scenario := MigrateUser{
  helper:       helper,
  questioner:   questioner,
  resources:    Resources{},
  cognitoActor: &actions.CognitoActions{CognitoClient:
 cognitoidentityprovider.NewFromConfig(sdkConfig)},
 }
 scenario.resources.init(scenario.cognitoActor, questioner)
 return scenario
}

// AddMigrateUserTrigger adds a Lambda handler as an invocation target for the
 MigrateUser trigger.
func (runner *MigrateUser) AddMigrateUserTrigger(ctx context.Context, userPoolId
 string, functionArn string) {
 log.Printf("Let's add a Lambda function to handle the MigrateUser trigger from
 Cognito.\n" +
  "This trigger happens when an unknown user signs in, and lets your function take
 action before Cognito\n" +
  "rejects the user.\n\n")
 err := runner.cognitoActor.UpdateTriggers(
  ctx, userPoolId,
```

```
  actions.TriggerInfo{Trigger: actions.UserMigration, HandlerArn:
 aws.String(functionArn)})
 if err != nil {
  panic(err)
 }
 log.Printf("Lambda function %v added to user pool %v to handle the MigrateUser
 trigger.\n",
  functionArn, userPoolId)

 log.Println(strings.Repeat("-", 88))
}

// SignInUser adds a new user to the known users table and signs that user in to
 Amazon Cognito.
func (runner *MigrateUser) SignInUser(ctx context.Context, usersTable string,
 clientId string) (bool, actions.User) {
 log.Println("Let's sign in a user to your Cognito user pool. When the username and
 email matches an entry in the\n" +
  "DynamoDB known users table, the email is automatically verified and the user is
 migrated to the Cognito user pool.")

 user := actions.User{}
 user.UserName = runner.questioner.Ask("\nEnter a username:")
 user.UserEmail = runner.questioner.Ask("\nEnter an email that you own. This email
 will be used to confirm user migration\n" +
  "during this example:")

 runner.helper.AddKnownUser(ctx, usersTable, user)

 var err error
 var resetRequired *types.PasswordResetRequiredException
 var authResult *types.AuthenticationResultType
 signedIn := false
 for !signedIn && resetRequired == nil {
  log.Printf("Signing in to Cognito as user '%v'. The expected result is a
 PasswordResetRequiredException.\n\n", user.UserName)
  authResult, err = runner.cognitoActor.SignIn(ctx, clientId, user.UserName, "_")
  if err != nil {
   if errors.As(err, &resetRequired) {
    log.Printf("\nUser '%v' is not in the Cognito user pool but was found in the
 DynamoDB known users table.\n"+
     "User migration is started and a password reset is required.", user.UserName)
   } else {
    panic(err)
```

```
   }
  } else {
   log.Printf("User '%v' successfully signed in. This is unexpected and probably
 means you have not\n"+
    "cleaned up a previous run of this scenario, so the user exist in the Cognito
 user pool.\n"+
    "You can continue this example and select to clean up resources, or manually
 remove\n"+
    "the user from your user pool and try again.", user.UserName)
   runner.resources.userAccessTokens = append(runner.resources.userAccessTokens,
 *authResult.AccessToken)
   signedIn = true
  }
 }

 log.Println(strings.Repeat("-", 88))
 return resetRequired != nil, user
}

// ResetPassword starts a password recovery flow.
func (runner *MigrateUser) ResetPassword(ctx context.Context, clientId string, user
 actions.User) {
 wantCode := runner.questioner.AskBool(fmt.Sprintf("In order to migrate the user to
 Cognito, you must be able to receive a confirmation\n"+
  "code by email at %v. Do you want to send a code (y/n)?", user.UserEmail), "y")
 if !wantCode {
  log.Println("To complete this example and successfully migrate a user to Cognito,
 you must enter an email\n" +
   "you own that can receive a confirmation code.")
  return
 }
 codeDelivery, err := runner.cognitoActor.ForgotPassword(ctx, clientId,
 user.UserName)
 if err != nil {
  panic(err)
 }
 log.Printf("\nA confirmation code has been sent to %v.", *codeDelivery.Destination)
 code := runner.questioner.Ask("Check your email and enter it here:")

 confirmed := false
 password := runner.questioner.AskPassword("\nEnter a password that has at least
 eight characters, uppercase, lowercase, numbers and symbols.\n"+
  "(the password will not display as you type):", 8)
 for !confirmed {
```

```
  log.Printf("\nConfirming password reset for user '%v'.\n", user.UserName)
  err = runner.cognitoActor.ConfirmForgotPassword(ctx, clientId, code,
 user.UserName, password)
  if err != nil {
   var invalidPassword *types.InvalidPasswordException
   if errors.As(err, &invalidPassword) {
    password = runner.questioner.AskPassword("\nEnter another password:", 8)
   } else {
    panic(err)
   }
  } else {
   confirmed = true
  }
 }
 log.Printf("User '%v' successfully confirmed and migrated.\n", user.UserName)
 log.Println("Signing in with your username and password...")
 authResult, err := runner.cognitoActor.SignIn(ctx, clientId, user.UserName,
 password)
 if err != nil {
  panic(err)
 }
 log.Printf("Successfully signed in. Your access token starts with: %v...\n",
 (*authResult.AccessToken)[:10])
 runner.resources.userAccessTokens = append(runner.resources.userAccessTokens,
 *authResult.AccessToken)

 log.Println(strings.Repeat("-", 88))
}

// Run runs the scenario.
func (runner *MigrateUser) Run(ctx context.Context, stackName string) {
 defer func() {
  if r := recover(); r != nil {
   log.Println("Something went wrong with the demo.")
   runner.resources.Cleanup(ctx)
  }
 }()

 log.Println(strings.Repeat("-", 88))
 log.Printf("Welcome\n")

 log.Println(strings.Repeat("-", 88))

 stackOutputs, err := runner.helper.GetStackOutputs(ctx, stackName)
```

```
 if err != nil {
  panic(err)
 }
 runner.resources.userPoolId = stackOutputs["UserPoolId"]

 runner.AddMigrateUserTrigger(ctx, stackOutputs["UserPoolId"],
 stackOutputs["MigrateUserFunctionArn"])
 runner.resources.triggers = append(runner.resources.triggers,
 actions.UserMigration)
 resetNeeded, user := runner.SignInUser(ctx, stackOutputs["TableName"],
 stackOutputs["UserPoolClientId"])
 if resetNeeded {
  runner.helper.ListRecentLogEvents(ctx, stackOutputs["MigrateUserFunction"])
  runner.ResetPassword(ctx, stackOutputs["UserPoolClientId"], user)
 }

 runner.resources.Cleanup(ctx)

 log.Println(strings.Repeat("-", 88))
 log.Println("Thanks for watching!")
 log.Println(strings.Repeat("-", 88))
}
```

Behandeln Sie den `MigrateUser`-Trigger mit einer Lambda-Funktion.

```
import (
 "context"
 "log"
 "os"

 "github.com/aws/aws-lambda-go/events"
 "github.com/aws/aws-lambda-go/lambda"
 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/config"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/expression"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
)

const TABLE_NAME = "TABLE_NAME"
```

```
// UserInfo defines structured user data that can be marshalled to a DynamoDB
 format.
type UserInfo struct {
 UserName  string `dynamodbav:"UserName"`
 UserEmail string `dynamodbav:"UserEmail"`
}

type handler struct {
 dynamoClient *dynamodb.Client
}

// HandleRequest handles the MigrateUser event by looking up a user in an Amazon
 DynamoDB table and
// specifying whether they should be migrated to the user pool.
func (h *handler) HandleRequest(ctx context.Context, event
 events.CognitoEventUserPoolsMigrateUser) (events.CognitoEventUserPoolsMigrateUser,
 error) {
 log.Printf("Received migrate trigger from %v for user '%v'", event.TriggerSource,
 event.UserName)
 if event.TriggerSource != "UserMigration_Authentication" {
  return event, nil
 }
 tableName := os.Getenv(TABLE_NAME)
 user := UserInfo{
  UserName: event.UserName,
 }
 log.Printf("Looking up user '%v' in table %v.\n", user.UserName, tableName)
 filterEx := expression.Name("UserName").Equal(expression.Value(user.UserName))
 expr, err := expression.NewBuilder().WithFilter(filterEx).Build()
 if err != nil {
  log.Printf("Error building expression to query for user '%v'.\n", user.UserName)
  return event, err
 }
 output, err := h.dynamoClient.Scan(ctx, &dynamodb.ScanInput{
  TableName:                 aws.String(tableName),
  FilterExpression:          expr.Filter(),
  ExpressionAttributeNames:  expr.Names(),
  ExpressionAttributeValues: expr.Values(),
 })
 if err != nil {
  log.Printf("Error looking up user '%v'.\n", user.UserName)
  return event, err
 }
```

```
 if len(output.Items) == 0 {
  log.Printf("User '%v' not found, not migrating user.\n", user.UserName)
  return event, err
 }

 var users []UserInfo
 err = attributevalue.UnmarshalListOfMaps(output.Items, &users)
 if err != nil {
  log.Printf("Couldn't unmarshal DynamoDB items. Here's why: %v\n", err)
  return event, err
 }

 user = users[0]
 log.Printf("UserName '%v' found with email %v. User is migrated and must reset
 password.\n", user.UserName, user.UserEmail)
 event.CognitoEventUserPoolsMigrateUserResponse.UserAttributes = map[string]string{
  "email":          user.UserEmail,
  "email_verified": "true", // email_verified is required for the forgot password
 flow.
 }
 event.CognitoEventUserPoolsMigrateUserResponse.FinalUserStatus = "RESET_REQUIRED"
 event.CognitoEventUserPoolsMigrateUserResponse.MessageAction = "SUPPRESS"

 return event, err
}

func main() {
 ctx := context.Background()
 sdkConfig, err := config.LoadDefaultConfig(ctx)
 if err != nil {
  log.Panicln(err)
 }
 h := handler{
  dynamoClient: dynamodb.NewFromConfig(sdkConfig),
 }
 lambda.Start(h.HandleRequest)
}
```

Erstellen Sie eine Struktur, die allgemeine Aufgaben ausführt.

```
import (
 "context"
 "log"
 "strings"
 "time"
 "user_pools_and_lambda_triggers/actions"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/cloudformation"
 "github.com/aws/aws-sdk-go-v2/service/cloudwatchlogs"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/awsdocs/aws-doc-sdk-examples/gov2/demotools"
)

// IScenarioHelper defines common functions used by the workflows in this example.
type IScenarioHelper interface {
 Pause(secs int)
 GetStackOutputs(ctx context.Context, stackName string) (actions.StackOutputs,
 error)
 PopulateUserTable(ctx context.Context, tableName string)
 GetKnownUsers(ctx context.Context, tableName string) (actions.UserList, error)
 AddKnownUser(ctx context.Context, tableName string, user actions.User)
 ListRecentLogEvents(ctx context.Context, functionName string)
}

// ScenarioHelper contains AWS wrapper structs used by the workflows in this
 example.
type ScenarioHelper struct {
 questioner  demotools.IQuestioner
 dynamoActor *actions.DynamoActions
 cfnActor    *actions.CloudFormationActions
 cwlActor    *actions.CloudWatchLogsActions
 isTestRun   bool
}

// NewScenarioHelper constructs a new scenario helper.
func NewScenarioHelper(sdkConfig aws.Config, questioner demotools.IQuestioner)
 ScenarioHelper {
 scenario := ScenarioHelper{
  questioner:  questioner,
  dynamoActor: &actions.DynamoActions{DynamoClient:
 dynamodb.NewFromConfig(sdkConfig)},
  cfnActor:    &actions.CloudFormationActions{CfnClient:
 cloudformation.NewFromConfig(sdkConfig)},
```

```
  cwlActor:    &actions.CloudWatchLogsActions{CwlClient:
 cloudwatchlogs.NewFromConfig(sdkConfig)},
 }
 return scenario
}

// Pause waits for the specified number of seconds.
func (helper ScenarioHelper) Pause(secs int) {
 if !helper.isTestRun {
  time.Sleep(time.Duration(secs) * time.Second)
 }
}

// GetStackOutputs gets the outputs from the specified CloudFormation stack in a
 structured format.
func (helper ScenarioHelper) GetStackOutputs(ctx context.Context, stackName string)
 (actions.StackOutputs, error) {
 return helper.cfnActor.GetOutputs(ctx, stackName), nil
}

// PopulateUserTable fills the known user table with example data.
func (helper ScenarioHelper) PopulateUserTable(ctx context.Context, tableName
 string) {
 log.Printf("First, let's add some users to the DynamoDB %v table we'll use for this
 example.\n", tableName)
 err := helper.dynamoActor.PopulateTable(ctx, tableName)
 if err != nil {
  panic(err)
 }
}

// GetKnownUsers gets the users from the known users table in a structured format.
func (helper ScenarioHelper) GetKnownUsers(ctx context.Context, tableName string)
 (actions.UserList, error) {
 knownUsers, err := helper.dynamoActor.Scan(ctx, tableName)
 if err != nil {
  log.Printf("Couldn't get known users from table %v. Here's why: %v\n", tableName,
 err)
 }
 return knownUsers, err
}

// AddKnownUser adds a user to the known users table.
```

```
func (helper ScenarioHelper) AddKnownUser(ctx context.Context, tableName string,
 user actions.User) {
 log.Printf("Adding user '%v' with email '%v' to the DynamoDB known users table...
\n",
  user.UserName, user.UserEmail)
 err := helper.dynamoActor.AddUser(ctx, tableName, user)
 if err != nil {
  panic(err)
 }
}

// ListRecentLogEvents gets the most recent log stream and events for the specified
 Lambda function and displays them.
func (helper ScenarioHelper) ListRecentLogEvents(ctx context.Context, functionName
 string) {
 log.Println("Waiting a few seconds to let Lambda write to CloudWatch Logs...")
 helper.Pause(10)
 log.Println("Okay, let's check the logs to find what's happened recently with your
 Lambda function.")
 logStream, err := helper.cwlActor.GetLatestLogStream(ctx, functionName)
 if err != nil {
  panic(err)
 }
 log.Printf("Getting some recent events from log stream %v\n",
 *logStream.LogStreamName)
 events, err := helper.cwlActor.GetLogEvents(ctx, functionName,
 *logStream.LogStreamName, 10)
 if err != nil {
  panic(err)
 }
 for _, event := range events {
  log.Printf("\t%v", *event.Message)
 }
 log.Println(strings.Repeat("-", 88))
}
```

Erstellen Sie eine Struktur, die Amazon Cognito-Aktionen umschließt.

```
import (
 "context"
```

```
 "errors"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/cognitoidentityprovider"
 "github.com/aws/aws-sdk-go-v2/service/cognitoidentityprovider/types"
)

type CognitoActions struct {
 CognitoClient *cognitoidentityprovider.Client
}



// Trigger and TriggerInfo define typed data for updating an Amazon Cognito trigger.
type Trigger int

const (
 PreSignUp Trigger = iota
 UserMigration
 PostAuthentication
)

type TriggerInfo struct {
 Trigger    Trigger
 HandlerArn *string
}

// UpdateTriggers adds or removes Lambda triggers for a user pool. When a trigger is
 specified with a `nil` value,
// it is removed from the user pool.
func (actor CognitoActions) UpdateTriggers(ctx context.Context, userPoolId string,
 triggers ...TriggerInfo) error {
 output, err := actor.CognitoClient.DescribeUserPool(ctx,
 &cognitoidentityprovider.DescribeUserPoolInput{
  UserPoolId: aws.String(userPoolId),
 })
 if err != nil {
  log.Printf("Couldn't get info about user pool %v. Here's why: %v\n", userPoolId,
 err)
  return err
 }
 lambdaConfig := output.UserPool.LambdaConfig
 for _, trigger := range triggers {
```

```
  switch trigger.Trigger {
  case PreSignUp:
   lambdaConfig.PreSignUp = trigger.HandlerArn
  case UserMigration:
   lambdaConfig.UserMigration = trigger.HandlerArn
  case PostAuthentication:
   lambdaConfig.PostAuthentication = trigger.HandlerArn
  }
 }
 _, err = actor.CognitoClient.UpdateUserPool(ctx,
 &cognitoidentityprovider.UpdateUserPoolInput{
  UserPoolId:   aws.String(userPoolId),
  LambdaConfig: lambdaConfig,
 })
 if err != nil {
  log.Printf("Couldn't update user pool %v. Here's why: %v\n", userPoolId, err)
 }
 return err
}



// SignUp signs up a user with Amazon Cognito.
func (actor CognitoActions) SignUp(ctx context.Context, clientId string, userName
 string, password string, userEmail string) (bool, error) {
 confirmed := false
 output, err := actor.CognitoClient.SignUp(ctx,
 &cognitoidentityprovider.SignUpInput{
  ClientId: aws.String(clientId),
  Password: aws.String(password),
  Username: aws.String(userName),
  UserAttributes: []types.AttributeType{
   {Name: aws.String("email"), Value: aws.String(userEmail)},
  },
 })
 if err != nil {
  var invalidPassword *types.InvalidPasswordException
  if errors.As(err, &invalidPassword) {
   log.Println(*invalidPassword.Message)
  } else {
   log.Printf("Couldn't sign up user %v. Here's why: %v\n", userName, err)
  }
 } else {
  confirmed = output.UserConfirmed
```

```
 }
 return confirmed, err
}



// SignIn signs in a user to Amazon Cognito using a username and password
 authentication flow.
func (actor CognitoActions) SignIn(ctx context.Context, clientId string, userName
 string, password string) (*types.AuthenticationResultType, error) {
 var authResult *types.AuthenticationResultType
 output, err := actor.CognitoClient.InitiateAuth(ctx,
 &cognitoidentityprovider.InitiateAuthInput{
  AuthFlow:       "USER_PASSWORD_AUTH",
  ClientId:       aws.String(clientId),
  AuthParameters: map[string]string{"USERNAME": userName, "PASSWORD": password},
 })
 if err != nil {
  var resetRequired *types.PasswordResetRequiredException
  if errors.As(err, &resetRequired) {
   log.Println(*resetRequired.Message)
  } else {
   log.Printf("Couldn't sign in user %v. Here's why: %v\n", userName, err)
  }
 } else {
  authResult = output.AuthenticationResult
 }
 return authResult, err
}



// ForgotPassword starts a password recovery flow for a user. This flow typically
 sends a confirmation code
// to the user's configured notification destination, such as email.
func (actor CognitoActions) ForgotPassword(ctx context.Context, clientId string,
 userName string) (*types.CodeDeliveryDetailsType, error) {
 output, err := actor.CognitoClient.ForgotPassword(ctx,
 &cognitoidentityprovider.ForgotPasswordInput{
  ClientId: aws.String(clientId),
  Username: aws.String(userName),
 })
 if err != nil {
```

```
  log.Printf("Couldn't start password reset for user '%v'. Here;s why: %v\n",
 userName, err)
 }
 return output.CodeDeliveryDetails, err
}



// ConfirmForgotPassword confirms a user with a confirmation code and a new
 password.
func (actor CognitoActions) ConfirmForgotPassword(ctx context.Context, clientId
 string, code string, userName string, password string) error {
 _, err := actor.CognitoClient.ConfirmForgotPassword(ctx,
 &cognitoidentityprovider.ConfirmForgotPasswordInput{
  ClientId:         aws.String(clientId),
  ConfirmationCode: aws.String(code),
  Password:         aws.String(password),
  Username:         aws.String(userName),
 })
 if err != nil {
  var invalidPassword *types.InvalidPasswordException
  if errors.As(err, &invalidPassword) {
   log.Println(*invalidPassword.Message)
  } else {
   log.Printf("Couldn't confirm user %v. Here's why: %v", userName, err)
  }
 }
 return err
}



// DeleteUser removes a user from the user pool.
func (actor CognitoActions) DeleteUser(ctx context.Context, userAccessToken string)
 error {
 _, err := actor.CognitoClient.DeleteUser(ctx,
 &cognitoidentityprovider.DeleteUserInput{
  AccessToken: aws.String(userAccessToken),
 })
 if err != nil {
  log.Printf("Couldn't delete user. Here's why: %v\n", err)
 }
 return err
}
```

```
// AdminCreateUser uses administrator credentials to add a user to a user pool. This
 method leaves the user
// in a state that requires they enter a new password next time they sign in.
func (actor CognitoActions) AdminCreateUser(ctx context.Context, userPoolId string,
 userName string, userEmail string) error {
 _, err := actor.CognitoClient.AdminCreateUser(ctx,
 &cognitoidentityprovider.AdminCreateUserInput{
  UserPoolId:     aws.String(userPoolId),
  Username:       aws.String(userName),
  MessageAction:  types.MessageActionTypeSuppress,
  UserAttributes: []types.AttributeType{{Name: aws.String("email"), Value:
 aws.String(userEmail)}},
 })
 if err != nil {
  var userExists *types.UsernameExistsException
  if errors.As(err, &userExists) {
   log.Printf("User %v already exists in the user pool.", userName)
   err = nil
  } else {
   log.Printf("Couldn't create user %v. Here's why: %v\n", userName, err)
  }
 }
 return err
}



// AdminSetUserPassword uses administrator credentials to set a password for a user
 without requiring a
// temporary password.
func (actor CognitoActions) AdminSetUserPassword(ctx context.Context, userPoolId
 string, userName string, password string) error {
 _, err := actor.CognitoClient.AdminSetUserPassword(ctx,
 &cognitoidentityprovider.AdminSetUserPasswordInput{
  Password:   aws.String(password),
  UserPoolId: aws.String(userPoolId),
  Username:   aws.String(userName),
  Permanent:  true,
 })
 if err != nil {
  var invalidPassword *types.InvalidPasswordException
```

```
  if errors.As(err, &invalidPassword) {
   log.Println(*invalidPassword.Message)
  } else {
   log.Printf("Couldn't set password for user %v. Here's why: %v\n", userName, err)
  }
 }
 return err
}
```

Erstellen Sie eine Struktur, die DynamoDB-Aktionen umhüllt.

```
import (
 "context"
 "fmt"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// DynamoActions encapsulates the Amazon Simple Notification Service (Amazon SNS)
 actions
// used in the examples.
type DynamoActions struct {
 DynamoClient *dynamodb.Client
}

// User defines structured user data.
type User struct {
 UserName  string
 UserEmail string
 LastLogin *LoginInfo `dynamodbav:",omitempty"`
}

// LoginInfo defines structured custom login data.
type LoginInfo struct {
 UserPoolId string
 ClientId   string
```

```
 Time       string
}

// UserList defines a list of users.
type UserList struct {
 Users []User
}

// UserNameList returns the usernames contained in a UserList as a list of strings.
func (users *UserList) UserNameList() []string {
 names := make([]string, len(users.Users))
 for i := 0; i < len(users.Users); i++ {
  names[i] = users.Users[i].UserName
 }
 return names
}

// PopulateTable adds a set of test users to the table.
func (actor DynamoActions) PopulateTable(ctx context.Context, tableName string)
 error {
 var err error
 var item map[string]types.AttributeValue
 var writeReqs []types.WriteRequest
 for i := 1; i < 4; i++ {
  item, err = attributevalue.MarshalMap(User{UserName: fmt.Sprintf("test_user_%v",
 i), UserEmail: fmt.Sprintf("test_email_%v@example.com", i)})
  if err != nil {
   log.Printf("Couldn't marshall user into DynamoDB format. Here's why: %v\n", err)
   return err
  }
  writeReqs = append(writeReqs, types.WriteRequest{PutRequest:
 &types.PutRequest{Item: item}})
 }
 _, err = actor.DynamoClient.BatchWriteItem(ctx, &dynamodb.BatchWriteItemInput{
  RequestItems: map[string][]types.WriteRequest{tableName: writeReqs},
 })
 if err != nil {
  log.Printf("Couldn't populate table %v with users. Here's why: %v\n", tableName,
 err)
 }
 return err
}

// Scan scans the table for all items.
```

```
func (actor DynamoActions) Scan(ctx context.Context, tableName string) (UserList,
 error) {
 var userList UserList
 output, err := actor.DynamoClient.Scan(ctx, &dynamodb.ScanInput{
  TableName: aws.String(tableName),
 })
 if err != nil {
  log.Printf("Couldn't scan table %v for items. Here's why: %v\n", tableName, err)
 } else {
  err = attributevalue.UnmarshalListOfMaps(output.Items, &userList.Users)
  if err != nil {
   log.Printf("Couldn't unmarshal items into users. Here's why: %v\n", err)
  }
 }
 return userList, err
}

// AddUser adds a user item to a table.
func (actor DynamoActions) AddUser(ctx context.Context, tableName string, user User)
 error {
 userItem, err := attributevalue.MarshalMap(user)
 if err != nil {
  log.Printf("Couldn't marshall user to item. Here's why: %v\n", err)
 }
 _, err = actor.DynamoClient.PutItem(ctx, &dynamodb.PutItemInput{
  Item:      userItem,
  TableName: aws.String(tableName),
 })
 if err != nil {
  log.Printf("Couldn't put item in table %v. Here's why: %v", tableName, err)
 }
 return err
}
```

Erstellen Sie eine Struktur, die CloudWatch Logs-Aktionen umschließt.

```
import (
 "context"
 "fmt"
 "log"
```

```
 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/cloudwatchlogs"
 "github.com/aws/aws-sdk-go-v2/service/cloudwatchlogs/types"
)

type CloudWatchLogsActions struct {
 CwlClient *cloudwatchlogs.Client
}

// GetLatestLogStream gets the most recent log stream for a Lambda function.
func (actor CloudWatchLogsActions) GetLatestLogStream(ctx context.Context,
 functionName string) (types.LogStream, error) {
 var logStream types.LogStream
 logGroupName := fmt.Sprintf("/aws/lambda/%s", functionName)
 output, err := actor.CwlClient.DescribeLogStreams(ctx,
 &cloudwatchlogs.DescribeLogStreamsInput{
  Descending:   aws.Bool(true),
  Limit:        aws.Int32(1),
  LogGroupName: aws.String(logGroupName),
  OrderBy:      types.OrderByLastEventTime,
 })
 if err != nil {
  log.Printf("Couldn't get log streams for log group %v. Here's why: %v\n",
 logGroupName, err)
 } else {
  logStream = output.LogStreams[0]
 }
 return logStream, err
}

// GetLogEvents gets the most recent eventCount events from the specified log
 stream.
func (actor CloudWatchLogsActions) GetLogEvents(ctx context.Context, functionName
 string, logStreamName string, eventCount int32) (
 []types.OutputLogEvent, error) {
 var events []types.OutputLogEvent
 logGroupName := fmt.Sprintf("/aws/lambda/%s", functionName)
 output, err := actor.CwlClient.GetLogEvents(ctx, &cloudwatchlogs.GetLogEventsInput{
  LogStreamName: aws.String(logStreamName),
  Limit:         aws.Int32(eventCount),
  LogGroupName:  aws.String(logGroupName),
 })
 if err != nil {
```

```
  log.Printf("Couldn't get log event for log stream %v. Here's why: %v\n",
 logStreamName, err)
 } else {
  events = output.Events
 }
 return events, err
}
```

Erstellen Sie eine Struktur, die Aktionen umschließt. AWS CloudFormation

```
import (
 "context"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/cloudformation"
)

// StackOutputs defines a map of outputs from a specific stack.
type StackOutputs map[string]string

type CloudFormationActions struct {
 CfnClient *cloudformation.Client
}

// GetOutputs gets the outputs from a CloudFormation stack and puts them into a
 structured format.
func (actor CloudFormationActions) GetOutputs(ctx context.Context, stackName string)
 StackOutputs {
 output, err := actor.CfnClient.DescribeStacks(ctx,
 &cloudformation.DescribeStacksInput{
  StackName: aws.String(stackName),
 })
 if err != nil || len(output.Stacks) == 0 {
  log.Panicf("Couldn't find a CloudFormation stack named %v. Here's why: %v\n",
 stackName, err)
 }
 stackOutputs := StackOutputs{}
 for _, out := range output.Stacks[0].Outputs {
  stackOutputs[*out.OutputKey] = *out.OutputValue
```

```
 }
 return stackOutputs
}
```

Bereinigen von Ressourcen.

```
import (
 "context"
 "log"
 "user_pools_and_lambda_triggers/actions"

 "github.com/awsdocs/aws-doc-sdk-examples/gov2/demotools"
)

// Resources keeps track of AWS resources created during an example and handles
// cleanup when the example finishes.
type Resources struct {
 userPoolId       string
 userAccessTokens []string
 triggers         []actions.Trigger

 cognitoActor *actions.CognitoActions
 questioner   demotools.IQuestioner
}

func (resources *Resources) init(cognitoActor *actions.CognitoActions, questioner
 demotools.IQuestioner) {
 resources.userAccessTokens = []string{}
 resources.triggers = []actions.Trigger{}
 resources.cognitoActor = cognitoActor
 resources.questioner = questioner
}

// Cleanup deletes all AWS resources created during an example.
func (resources *Resources) Cleanup(ctx context.Context) {
 defer func() {
  if r := recover(); r != nil {
   log.Printf("Something went wrong during cleanup.\n%v\n", r)
   log.Println("Use the AWS Management Console to remove any remaining resources \n"
 +
```

```
    "that were created for this scenario.")
  }
 }()

 wantDelete := resources.questioner.AskBool("Do you want to remove all of the AWS
 resources that were created "+
  "during this demo (y/n)?", "y")
 if wantDelete {
  for _, accessToken := range resources.userAccessTokens {
   err := resources.cognitoActor.DeleteUser(ctx, accessToken)
   if err != nil {
    log.Println("Couldn't delete user during cleanup.")
    panic(err)
   }
   log.Println("Deleted user.")
  }
  triggerList := make([]actions.TriggerInfo, len(resources.triggers))
  for i := 0; i < len(resources.triggers); i++ {
   triggerList[i] = actions.TriggerInfo{Trigger: resources.triggers[i], HandlerArn:
 nil}
  }
  err := resources.cognitoActor.UpdateTriggers(ctx, resources.userPoolId,
 triggerList...)
  if err != nil {
   log.Println("Couldn't update Cognito triggers during cleanup.")
   panic(err)
  }
  log.Println("Removed Cognito triggers from user pool.")
 } else {
  log.Println("Be sure to remove resources when you're done with them to avoid
 unexpected charges!")
 }
}
```

- API-Details finden Sie in den folgenden Themen der AWS SDK für Go -API-Referenz.
  - ConfirmForgotPassword
  - DeleteUser
  - ForgotPassword
  - InitiateAuth

- SignUp
- UpdateUserPool

Schreiben von benutzerdefinierten Aktivitätsdaten mit einer Lambda-Funktion nach der Amazon Cognito-Benutzerauthentifizierung

Das folgende Codebeispiel veranschaulicht, wie benutzerdefinierte Aktivitätsdaten mit einer Lambda-Funktion nach der Amazon-Cognito-Benutzerauthentifizierung geschrieben werden.

- Verwenden Sie Administratorfunktionen, um einen Benutzer zu einem Benutzerpool hinzuzufügen.
- Konfigurieren Sie einen Benutzerpool, um eine Lambda-Funktion für den `PostAuthentication`-Trigger aufzurufen.
- Melden Sie den neuen Benutzer bei Amazon Cognito an.
- Die Lambda-Funktion schreibt benutzerdefinierte Informationen in CloudWatch Logs und in eine DynamoDB-Tabelle.
- Ermitteln Sie benutzerdefinierte Daten aus der DynamoDB-Tabelle, zeigen Sie diese an und bereinigen Sie dann die Ressourcen.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Führen Sie ein interaktives Szenario an einem Prompt aus.

```
import (
 "context"
 "errors"
 "log"
 "strings"
 "user_pools_and_lambda_triggers/actions"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/cognitoidentityprovider"
```

```
 "github.com/aws/aws-sdk-go-v2/service/cognitoidentityprovider/types"
 "github.com/awsdocs/aws-doc-sdk-examples/gov2/demotools"
)

// ActivityLog separates the steps of this scenario into individual functions so
 that
// they are simpler to read and understand.
type ActivityLog struct {
 helper       IScenarioHelper
 questioner   demotools.IQuestioner
 resources    Resources
 cognitoActor *actions.CognitoActions
}

// NewActivityLog constructs a new activity log runner.
func NewActivityLog(sdkConfig aws.Config, questioner demotools.IQuestioner, helper
 IScenarioHelper) ActivityLog {
 scenario := ActivityLog{
  helper:       helper,
  questioner:   questioner,
  resources:    Resources{},
  cognitoActor: &actions.CognitoActions{CognitoClient:
 cognitoidentityprovider.NewFromConfig(sdkConfig)},
 }
 scenario.resources.init(scenario.cognitoActor, questioner)
 return scenario
}

// AddUserToPool selects a user from the known users table and uses administrator
 credentials to add the user to the user pool.
func (runner *ActivityLog) AddUserToPool(ctx context.Context, userPoolId string,
 tableName string) (string, string) {
 log.Println("To facilitate this example, let's add a user to the user pool using
 administrator privileges.")
 users, err := runner.helper.GetKnownUsers(ctx, tableName)
 if err != nil {
  panic(err)
 }
 user := users.Users[0]
 log.Printf("Adding known user %v to the user pool.\n", user.UserName)
 err = runner.cognitoActor.AdminCreateUser(ctx, userPoolId, user.UserName,
 user.UserEmail)
 if err != nil {
  panic(err)
```

```
 }
 pwSet := false
 password := runner.questioner.AskPassword("\nEnter a password that has at least
 eight characters, uppercase, lowercase, numbers and symbols.\n"+
  "(the password will not display as you type):", 8)
 for !pwSet {
  log.Printf("\nSetting password for user '%v'.\n", user.UserName)
  err = runner.cognitoActor.AdminSetUserPassword(ctx, userPoolId, user.UserName,
 password)
  if err != nil {
   var invalidPassword *types.InvalidPasswordException
   if errors.As(err, &invalidPassword) {
    password = runner.questioner.AskPassword("\nEnter another password:", 8)
   } else {
    panic(err)
   }
  } else {
   pwSet = true
  }
 }

 log.Println(strings.Repeat("-", 88))

 return user.UserName, password
}

// AddActivityLogTrigger adds a Lambda handler as an invocation target for the
 PostAuthentication trigger.
func (runner *ActivityLog) AddActivityLogTrigger(ctx context.Context, userPoolId
 string, activityLogArn string) {
 log.Println("Let's add a Lambda function to handle the PostAuthentication trigger
 from Cognito.\n" +
  "This trigger happens after a user is authenticated, and lets your function take
 action, such as logging\n" +
  "the outcome.")
 err := runner.cognitoActor.UpdateTriggers(
  ctx, userPoolId,
  actions.TriggerInfo{Trigger: actions.PostAuthentication, HandlerArn:
 aws.String(activityLogArn)})
 if err != nil {
  panic(err)
 }
 runner.resources.triggers = append(runner.resources.triggers,
 actions.PostAuthentication)
```

```
 log.Printf("Lambda function %v added to user pool %v to handle PostAuthentication
 Cognito trigger.\n",
  activityLogArn, userPoolId)

 log.Println(strings.Repeat("-", 88))
}

// SignInUser signs in as the specified user.
func (runner *ActivityLog) SignInUser(ctx context.Context, clientId string, userName
 string, password string) {
 log.Printf("Now we'll sign in user %v and check the results in the logs and the
 DynamoDB table.", userName)
 runner.questioner.Ask("Press Enter when you're ready.")
 authResult, err := runner.cognitoActor.SignIn(ctx, clientId, userName, password)
 if err != nil {
  panic(err)
 }
 log.Println("Sign in successful.",
  "The PostAuthentication Lambda handler writes custom information to CloudWatch
 Logs.")

 runner.resources.userAccessTokens = append(runner.resources.userAccessTokens,
 *authResult.AccessToken)
}

// GetKnownUserLastLogin gets the login info for a user from the Amazon DynamoDB
 table and displays it.
func (runner *ActivityLog) GetKnownUserLastLogin(ctx context.Context, tableName
 string, userName string) {
 log.Println("The PostAuthentication handler also writes login data to the DynamoDB
 table.")
 runner.questioner.Ask("Press Enter when you're ready to continue.")
 users, err := runner.helper.GetKnownUsers(ctx, tableName)
 if err != nil {
  panic(err)
 }
 for _, user := range users.Users {
  if user.UserName == userName {
   log.Println("The last login info for the user in the known users table is:")
   log.Printf("\t%+v", *user.LastLogin)
  }
 }
 log.Println(strings.Repeat("-", 88))
}
```

```
// Run runs the scenario.
func (runner *ActivityLog) Run(ctx context.Context, stackName string) {
 defer func() {
  if r := recover(); r != nil {
   log.Println("Something went wrong with the demo.")
   runner.resources.Cleanup(ctx)
  }
 }()

 log.Println(strings.Repeat("-", 88))
 log.Printf("Welcome\n")

 log.Println(strings.Repeat("-", 88))

 stackOutputs, err := runner.helper.GetStackOutputs(ctx, stackName)
 if err != nil {
  panic(err)
 }
 runner.resources.userPoolId = stackOutputs["UserPoolId"]
 runner.helper.PopulateUserTable(ctx, stackOutputs["TableName"])
 userName, password := runner.AddUserToPool(ctx, stackOutputs["UserPoolId"],
 stackOutputs["TableName"])

 runner.AddActivityLogTrigger(ctx, stackOutputs["UserPoolId"],
 stackOutputs["ActivityLogFunctionArn"])
 runner.SignInUser(ctx, stackOutputs["UserPoolClientId"], userName, password)
 runner.helper.ListRecentLogEvents(ctx, stackOutputs["ActivityLogFunction"])
 runner.GetKnownUserLastLogin(ctx, stackOutputs["TableName"], userName)

 runner.resources.Cleanup(ctx)

 log.Println(strings.Repeat("-", 88))
 log.Println("Thanks for watching!")
 log.Println(strings.Repeat("-", 88))
}
```

Behandeln Sie den `PostAuthentication`-Trigger mit einer Lambda-Funktion.

```
import (
```

```
 "context"
 "fmt"
 "log"
 "os"
 "time"

 "github.com/aws/aws-lambda-go/events"
 "github.com/aws/aws-lambda-go/lambda"
 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/config"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 dynamodbtypes "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

const TABLE_NAME = "TABLE_NAME"

// LoginInfo defines structured login data that can be marshalled to a DynamoDB
 format.
type LoginInfo struct {
 UserPoolId string `dynamodbav:"UserPoolId"`
 ClientId   string `dynamodbav:"ClientId"`
 Time       string `dynamodbav:"Time"`
}

// UserInfo defines structured user data that can be marshalled to a DynamoDB
 format.
type UserInfo struct {
 UserName  string    `dynamodbav:"UserName"`
 UserEmail string    `dynamodbav:"UserEmail"`
 LastLogin LoginInfo `dynamodbav:"LastLogin"`
}

// GetKey marshals the user email value to a DynamoDB key format.
func (user UserInfo) GetKey() map[string]dynamodbtypes.AttributeValue {
 userEmail, err := attributevalue.Marshal(user.UserEmail)
 if err != nil {
  panic(err)
 }
 return map[string]dynamodbtypes.AttributeValue{"UserEmail": userEmail}
}

type handler struct {
 dynamoClient *dynamodb.Client
```

```
}

// HandleRequest handles the PostAuthentication event by writing custom data to the
 logs and
// to an Amazon DynamoDB table.
func (h *handler) HandleRequest(ctx context.Context,
 event events.CognitoEventUserPoolsPostAuthentication)
 (events.CognitoEventUserPoolsPostAuthentication, error) {
 log.Printf("Received post authentication trigger from %v for user '%v'",
 event.TriggerSource, event.UserName)
 tableName := os.Getenv(TABLE_NAME)
 user := UserInfo{
  UserName:  event.UserName,
  UserEmail: event.Request.UserAttributes["email"],
  LastLogin: LoginInfo{
   UserPoolId: event.UserPoolID,
   ClientId:   event.CallerContext.ClientID,
   Time:       time.Now().Format(time.UnixDate),
  },
 }
 // Write to CloudWatch Logs.
 fmt.Printf("%#v", user)

 // Also write to an external system. This examples uses DynamoDB to demonstrate.
 userMap, err := attributevalue.MarshalMap(user)
 if err != nil {
  log.Printf("Couldn't marshal to DynamoDB map. Here's why: %v\n", err)
 } else if len(userMap) == 0 {
  log.Printf("User info marshaled to an empty map.")
 } else {
  _, err := h.dynamoClient.PutItem(ctx, &dynamodb.PutItemInput{
   Item:      userMap,
   TableName: aws.String(tableName),
  })
  if err != nil {
   log.Printf("Couldn't write to DynamoDB. Here's why: %v\n", err)
  } else {
   log.Printf("Wrote user info to DynamoDB table %v.\n", tableName)
  }
 }

 return event, nil
}
```

```
func main() {
 ctx := context.Background()
 sdkConfig, err := config.LoadDefaultConfig(ctx)
 if err != nil {
  log.Panicln(err)
 }
 h := handler{
  dynamoClient: dynamodb.NewFromConfig(sdkConfig),
 }
 lambda.Start(h.HandleRequest)
}
```

Erstellen Sie eine Struktur, die allgemeine Aufgaben ausführt.

```
import (
 "context"
 "log"
 "strings"
 "time"
 "user_pools_and_lambda_triggers/actions"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/cloudformation"
 "github.com/aws/aws-sdk-go-v2/service/cloudwatchlogs"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/awsdocs/aws-doc-sdk-examples/gov2/demotools"
)

// IScenarioHelper defines common functions used by the workflows in this example.
type IScenarioHelper interface {
 Pause(secs int)
 GetStackOutputs(ctx context.Context, stackName string) (actions.StackOutputs,
 error)
 PopulateUserTable(ctx context.Context, tableName string)
 GetKnownUsers(ctx context.Context, tableName string) (actions.UserList, error)
 AddKnownUser(ctx context.Context, tableName string, user actions.User)
 ListRecentLogEvents(ctx context.Context, functionName string)
}
```

```
// ScenarioHelper contains AWS wrapper structs used by the workflows in this
 example.
type ScenarioHelper struct {
 questioner  demotools.IQuestioner
 dynamoActor *actions.DynamoActions
 cfnActor    *actions.CloudFormationActions
 cwlActor    *actions.CloudWatchLogsActions
 isTestRun   bool
}

// NewScenarioHelper constructs a new scenario helper.
func NewScenarioHelper(sdkConfig aws.Config, questioner demotools.IQuestioner)
 ScenarioHelper {
 scenario := ScenarioHelper{
  questioner:  questioner,
  dynamoActor: &actions.DynamoActions{DynamoClient:
 dynamodb.NewFromConfig(sdkConfig)},
  cfnActor:    &actions.CloudFormationActions{CfnClient:
 cloudformation.NewFromConfig(sdkConfig)},
  cwlActor:    &actions.CloudWatchLogsActions{CwlClient:
 cloudwatchlogs.NewFromConfig(sdkConfig)},
 }
 return scenario
}

// Pause waits for the specified number of seconds.
func (helper ScenarioHelper) Pause(secs int) {
 if !helper.isTestRun {
  time.Sleep(time.Duration(secs) * time.Second)
 }
}

// GetStackOutputs gets the outputs from the specified CloudFormation stack in a
 structured format.
func (helper ScenarioHelper) GetStackOutputs(ctx context.Context, stackName string)
 (actions.StackOutputs, error) {
 return helper.cfnActor.GetOutputs(ctx, stackName), nil
}

// PopulateUserTable fills the known user table with example data.
func (helper ScenarioHelper) PopulateUserTable(ctx context.Context, tableName
 string) {
 log.Printf("First, let's add some users to the DynamoDB %v table we'll use for this
 example.\n", tableName)
```

```
 err := helper.dynamoActor.PopulateTable(ctx, tableName)
 if err != nil {
  panic(err)
 }
}

// GetKnownUsers gets the users from the known users table in a structured format.
func (helper ScenarioHelper) GetKnownUsers(ctx context.Context, tableName string)
 (actions.UserList, error) {
 knownUsers, err := helper.dynamoActor.Scan(ctx, tableName)
 if err != nil {
  log.Printf("Couldn't get known users from table %v. Here's why: %v\n", tableName,
 err)
 }
 return knownUsers, err
}

// AddKnownUser adds a user to the known users table.
func (helper ScenarioHelper) AddKnownUser(ctx context.Context, tableName string,
 user actions.User) {
 log.Printf("Adding user '%v' with email '%v' to the DynamoDB known users table...
\n",
  user.UserName, user.UserEmail)
 err := helper.dynamoActor.AddUser(ctx, tableName, user)
 if err != nil {
  panic(err)
 }
}

// ListRecentLogEvents gets the most recent log stream and events for the specified
 Lambda function and displays them.
func (helper ScenarioHelper) ListRecentLogEvents(ctx context.Context, functionName
 string) {
 log.Println("Waiting a few seconds to let Lambda write to CloudWatch Logs...")
 helper.Pause(10)
 log.Println("Okay, let's check the logs to find what's happened recently with your
 Lambda function.")
 logStream, err := helper.cwlActor.GetLatestLogStream(ctx, functionName)
 if err != nil {
  panic(err)
 }
 log.Printf("Getting some recent events from log stream %v\n",
 *logStream.LogStreamName)
```

```
 events, err := helper.cwlActor.GetLogEvents(ctx, functionName,
 *logStream.LogStreamName, 10)
 if err != nil {
  panic(err)
 }
 for _, event := range events {
  log.Printf("\t%v", *event.Message)
 }
 log.Println(strings.Repeat("-", 88))
}
```

Erstellen Sie eine Struktur, die Amazon Cognito-Aktionen umschließt.

```
import (
 "context"
 "errors"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/cognitoidentityprovider"
 "github.com/aws/aws-sdk-go-v2/service/cognitoidentityprovider/types"
)

type CognitoActions struct {
 CognitoClient *cognitoidentityprovider.Client
}



// Trigger and TriggerInfo define typed data for updating an Amazon Cognito trigger.
type Trigger int

const (
 PreSignUp Trigger = iota
 UserMigration
 PostAuthentication
)

type TriggerInfo struct {
 Trigger    Trigger
```

```
 HandlerArn *string
}

// UpdateTriggers adds or removes Lambda triggers for a user pool. When a trigger is
 specified with a `nil` value,
// it is removed from the user pool.
func (actor CognitoActions) UpdateTriggers(ctx context.Context, userPoolId string,
 triggers ...TriggerInfo) error {
 output, err := actor.CognitoClient.DescribeUserPool(ctx,
 &cognitoidentityprovider.DescribeUserPoolInput{
  UserPoolId: aws.String(userPoolId),
 })
 if err != nil {
  log.Printf("Couldn't get info about user pool %v. Here's why: %v\n", userPoolId,
 err)
  return err
 }
 lambdaConfig := output.UserPool.LambdaConfig
 for _, trigger := range triggers {
  switch trigger.Trigger {
  case PreSignUp:
   lambdaConfig.PreSignUp = trigger.HandlerArn
  case UserMigration:
   lambdaConfig.UserMigration = trigger.HandlerArn
  case PostAuthentication:
   lambdaConfig.PostAuthentication = trigger.HandlerArn
  }
 }
 _, err = actor.CognitoClient.UpdateUserPool(ctx,
 &cognitoidentityprovider.UpdateUserPoolInput{
  UserPoolId:   aws.String(userPoolId),
  LambdaConfig: lambdaConfig,
 })
 if err != nil {
  log.Printf("Couldn't update user pool %v. Here's why: %v\n", userPoolId, err)
 }
 return err
}



// SignUp signs up a user with Amazon Cognito.
func (actor CognitoActions) SignUp(ctx context.Context, clientId string, userName
 string, password string, userEmail string) (bool, error) {
```

```
 confirmed := false
 output, err := actor.CognitoClient.SignUp(ctx,
 &cognitoidentityprovider.SignUpInput{
  ClientId: aws.String(clientId),
  Password: aws.String(password),
  Username: aws.String(userName),
  UserAttributes: []types.AttributeType{
   {Name: aws.String("email"), Value: aws.String(userEmail)},
  },
 })
 if err != nil {
  var invalidPassword *types.InvalidPasswordException
  if errors.As(err, &invalidPassword) {
   log.Println(*invalidPassword.Message)
  } else {
   log.Printf("Couldn't sign up user %v. Here's why: %v\n", userName, err)
  }
 } else {
  confirmed = output.UserConfirmed
 }
 return confirmed, err
}



// SignIn signs in a user to Amazon Cognito using a username and password
 authentication flow.
func (actor CognitoActions) SignIn(ctx context.Context, clientId string, userName
 string, password string) (*types.AuthenticationResultType, error) {
 var authResult *types.AuthenticationResultType
 output, err := actor.CognitoClient.InitiateAuth(ctx,
 &cognitoidentityprovider.InitiateAuthInput{
  AuthFlow:       "USER_PASSWORD_AUTH",
  ClientId:       aws.String(clientId),
  AuthParameters: map[string]string{"USERNAME": userName, "PASSWORD": password},
 })
 if err != nil {
  var resetRequired *types.PasswordResetRequiredException
  if errors.As(err, &resetRequired) {
   log.Println(*resetRequired.Message)
  } else {
   log.Printf("Couldn't sign in user %v. Here's why: %v\n", userName, err)
  }
 } else {
```

```
  authResult = output.AuthenticationResult
 }
 return authResult, err
}



// ForgotPassword starts a password recovery flow for a user. This flow typically
 sends a confirmation code
// to the user's configured notification destination, such as email.
func (actor CognitoActions) ForgotPassword(ctx context.Context, clientId string,
 userName string) (*types.CodeDeliveryDetailsType, error) {
 output, err := actor.CognitoClient.ForgotPassword(ctx,
 &cognitoidentityprovider.ForgotPasswordInput{
  ClientId: aws.String(clientId),
  Username: aws.String(userName),
 })
 if err != nil {
  log.Printf("Couldn't start password reset for user '%v'. Here;s why: %v\n",
 userName, err)
 }
 return output.CodeDeliveryDetails, err
}



// ConfirmForgotPassword confirms a user with a confirmation code and a new
 password.
func (actor CognitoActions) ConfirmForgotPassword(ctx context.Context, clientId
 string, code string, userName string, password string) error {
 _, err := actor.CognitoClient.ConfirmForgotPassword(ctx,
 &cognitoidentityprovider.ConfirmForgotPasswordInput{
  ClientId:         aws.String(clientId),
  ConfirmationCode: aws.String(code),
  Password:         aws.String(password),
  Username:         aws.String(userName),
 })
 if err != nil {
  var invalidPassword *types.InvalidPasswordException
  if errors.As(err, &invalidPassword) {
   log.Println(*invalidPassword.Message)
  } else {
   log.Printf("Couldn't confirm user %v. Here's why: %v", userName, err)
  }
```

```
 }
 return err
}



// DeleteUser removes a user from the user pool.
func (actor CognitoActions) DeleteUser(ctx context.Context, userAccessToken string)
 error {
 _, err := actor.CognitoClient.DeleteUser(ctx,
 &cognitoidentityprovider.DeleteUserInput{
  AccessToken: aws.String(userAccessToken),
 })
 if err != nil {
  log.Printf("Couldn't delete user. Here's why: %v\n", err)
 }
 return err
}



// AdminCreateUser uses administrator credentials to add a user to a user pool. This
 method leaves the user
// in a state that requires they enter a new password next time they sign in.
func (actor CognitoActions) AdminCreateUser(ctx context.Context, userPoolId string,
 userName string, userEmail string) error {
 _, err := actor.CognitoClient.AdminCreateUser(ctx,
 &cognitoidentityprovider.AdminCreateUserInput{
  UserPoolId:     aws.String(userPoolId),
  Username:       aws.String(userName),
  MessageAction:  types.MessageActionTypeSuppress,
  UserAttributes: []types.AttributeType{{Name: aws.String("email"), Value:
 aws.String(userEmail)}},
 })
 if err != nil {
  var userExists *types.UsernameExistsException
  if errors.As(err, &userExists) {
   log.Printf("User %v already exists in the user pool.", userName)
   err = nil
  } else {
   log.Printf("Couldn't create user %v. Here's why: %v\n", userName, err)
  }
 }
 return err
```

```
}



// AdminSetUserPassword uses administrator credentials to set a password for a user
 without requiring a
// temporary password.
func (actor CognitoActions) AdminSetUserPassword(ctx context.Context, userPoolId
 string, userName string, password string) error {
 _, err := actor.CognitoClient.AdminSetUserPassword(ctx,
 &cognitoidentityprovider.AdminSetUserPasswordInput{
  Password:   aws.String(password),
  UserPoolId: aws.String(userPoolId),
  Username:   aws.String(userName),
  Permanent:  true,
 })
 if err != nil {
  var invalidPassword *types.InvalidPasswordException
  if errors.As(err, &invalidPassword) {
   log.Println(*invalidPassword.Message)
  } else {
   log.Printf("Couldn't set password for user %v. Here's why: %v\n", userName, err)
  }
 }
 return err
}
```

Erstellen Sie eine Struktur, die DynamoDB-Aktionen umhüllt.

```
import (
 "context"
 "fmt"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)
```

```
// DynamoActions encapsulates the Amazon Simple Notification Service (Amazon SNS)
 actions
// used in the examples.
type DynamoActions struct {
 DynamoClient *dynamodb.Client
}

// User defines structured user data.
type User struct {
 UserName  string
 UserEmail string
 LastLogin *LoginInfo `dynamodbav:",omitempty"`
}

// LoginInfo defines structured custom login data.
type LoginInfo struct {
 UserPoolId string
 ClientId   string
 Time       string
}

// UserList defines a list of users.
type UserList struct {
 Users []User
}

// UserNameList returns the usernames contained in a UserList as a list of strings.
func (users *UserList) UserNameList() []string {
 names := make([]string, len(users.Users))
 for i := 0; i < len(users.Users); i++ {
  names[i] = users.Users[i].UserName
 }
 return names
}

// PopulateTable adds a set of test users to the table.
func (actor DynamoActions) PopulateTable(ctx context.Context, tableName string)
 error {
 var err error
 var item map[string]types.AttributeValue
 var writeReqs []types.WriteRequest
 for i := 1; i < 4; i++ {
  item, err = attributevalue.MarshalMap(User{UserName: fmt.Sprintf("test_user_%v",
 i), UserEmail: fmt.Sprintf("test_email_%v@example.com", i)})
```

```
  if err != nil {
   log.Printf("Couldn't marshall user into DynamoDB format. Here's why: %v\n", err)
   return err
  }
  writeReqs = append(writeReqs, types.WriteRequest{PutRequest:
 &types.PutRequest{Item: item}})
 }
 _, err = actor.DynamoClient.BatchWriteItem(ctx, &dynamodb.BatchWriteItemInput{
  RequestItems: map[string][]types.WriteRequest{tableName: writeReqs},
 })
 if err != nil {
  log.Printf("Couldn't populate table %v with users. Here's why: %v\n", tableName,
 err)
 }
 return err
}

// Scan scans the table for all items.
func (actor DynamoActions) Scan(ctx context.Context, tableName string) (UserList,
 error) {
 var userList UserList
 output, err := actor.DynamoClient.Scan(ctx, &dynamodb.ScanInput{
  TableName: aws.String(tableName),
 })
 if err != nil {
  log.Printf("Couldn't scan table %v for items. Here's why: %v\n", tableName, err)
 } else {
  err = attributevalue.UnmarshalListOfMaps(output.Items, &userList.Users)
  if err != nil {
   log.Printf("Couldn't unmarshal items into users. Here's why: %v\n", err)
  }
 }
 return userList, err
}

// AddUser adds a user item to a table.
func (actor DynamoActions) AddUser(ctx context.Context, tableName string, user User)
 error {
 userItem, err := attributevalue.MarshalMap(user)
 if err != nil {
  log.Printf("Couldn't marshall user to item. Here's why: %v\n", err)
 }
 _, err = actor.DynamoClient.PutItem(ctx, &dynamodb.PutItemInput{
  Item:      userItem,
```

```
  TableName: aws.String(tableName),
 })
 if err != nil {
  log.Printf("Couldn't put item in table %v. Here's why: %v", tableName, err)
 }
 return err
}
```

Erstellen Sie eine Struktur, die CloudWatch Logs-Aktionen umschließt.

```
import (
 "context"
 "fmt"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/cloudwatchlogs"
 "github.com/aws/aws-sdk-go-v2/service/cloudwatchlogs/types"
)

type CloudWatchLogsActions struct {
 CwlClient *cloudwatchlogs.Client
}

// GetLatestLogStream gets the most recent log stream for a Lambda function.
func (actor CloudWatchLogsActions) GetLatestLogStream(ctx context.Context,
 functionName string) (types.LogStream, error) {
 var logStream types.LogStream
 logGroupName := fmt.Sprintf("/aws/lambda/%s", functionName)
 output, err := actor.CwlClient.DescribeLogStreams(ctx,
 &cloudwatchlogs.DescribeLogStreamsInput{
  Descending:   aws.Bool(true),
  Limit:        aws.Int32(1),
  LogGroupName: aws.String(logGroupName),
  OrderBy:      types.OrderByLastEventTime,
 })
 if err != nil {
  log.Printf("Couldn't get log streams for log group %v. Here's why: %v\n",
 logGroupName, err)
 } else {
```

```
  logStream = output.LogStreams[0]
 }
 return logStream, err
}

// GetLogEvents gets the most recent eventCount events from the specified log
 stream.
func (actor CloudWatchLogsActions) GetLogEvents(ctx context.Context, functionName
 string, logStreamName string, eventCount int32) (
 []types.OutputLogEvent, error) {
 var events []types.OutputLogEvent
 logGroupName := fmt.Sprintf("/aws/lambda/%s", functionName)
 output, err := actor.CwlClient.GetLogEvents(ctx, &cloudwatchlogs.GetLogEventsInput{
  LogStreamName: aws.String(logStreamName),
  Limit:         aws.Int32(eventCount),
  LogGroupName:  aws.String(logGroupName),
 })
 if err != nil {
  log.Printf("Couldn't get log event for log stream %v. Here's why: %v\n",
 logStreamName, err)
 } else {
  events = output.Events
 }
 return events, err
}
```

Erstellen Sie eine Struktur, die Aktionen umschließt. AWS CloudFormation

```
import (
 "context"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/cloudformation"
)

// StackOutputs defines a map of outputs from a specific stack.
type StackOutputs map[string]string

type CloudFormationActions struct {
```

```
 CfnClient *cloudformation.Client
}

// GetOutputs gets the outputs from a CloudFormation stack and puts them into a
 structured format.
func (actor CloudFormationActions) GetOutputs(ctx context.Context, stackName string)
 StackOutputs {
 output, err := actor.CfnClient.DescribeStacks(ctx,
 &cloudformation.DescribeStacksInput{
  StackName: aws.String(stackName),
 })
 if err != nil || len(output.Stacks) == 0 {
  log.Panicf("Couldn't find a CloudFormation stack named %v. Here's why: %v\n",
 stackName, err)
 }
 stackOutputs := StackOutputs{}
 for _, out := range output.Stacks[0].Outputs {
  stackOutputs[*out.OutputKey] = *out.OutputValue
 }
 return stackOutputs
}
```

Bereinigen von Ressourcen.

```
import (
 "context"
 "log"
 "user_pools_and_lambda_triggers/actions"

 "github.com/awsdocs/aws-doc-sdk-examples/gov2/demotools"
)

// Resources keeps track of AWS resources created during an example and handles
// cleanup when the example finishes.
type Resources struct {
 userPoolId       string
 userAccessTokens []string
 triggers         []actions.Trigger

 cognitoActor *actions.CognitoActions
```

```
 questioner   demotools.IQuestioner
}

func (resources *Resources) init(cognitoActor *actions.CognitoActions, questioner
 demotools.IQuestioner) {
 resources.userAccessTokens = []string{}
 resources.triggers = []actions.Trigger{}
 resources.cognitoActor = cognitoActor
 resources.questioner = questioner
}

// Cleanup deletes all AWS resources created during an example.
func (resources *Resources) Cleanup(ctx context.Context) {
 defer func() {
  if r := recover(); r != nil {
   log.Printf("Something went wrong during cleanup.\n%v\n", r)
   log.Println("Use the AWS Management Console to remove any remaining resources \n"
 +
    "that were created for this scenario.")
  }
 }()

 wantDelete := resources.questioner.AskBool("Do you want to remove all of the AWS
 resources that were created "+
  "during this demo (y/n)?", "y")
 if wantDelete {
  for _, accessToken := range resources.userAccessTokens {
   err := resources.cognitoActor.DeleteUser(ctx, accessToken)
   if err != nil {
    log.Println("Couldn't delete user during cleanup.")
    panic(err)
   }
   log.Println("Deleted user.")
  }
  triggerList := make([]actions.TriggerInfo, len(resources.triggers))
  for i := 0; i < len(resources.triggers); i++ {
   triggerList[i] = actions.TriggerInfo{Trigger: resources.triggers[i], HandlerArn:
 nil}
  }
  err := resources.cognitoActor.UpdateTriggers(ctx, resources.userPoolId,
 triggerList...)
  if err != nil {
   log.Println("Couldn't update Cognito triggers during cleanup.")
   panic(err)
```

```
  }
  log.Println("Removed Cognito triggers from user pool.")
 } else {
  log.Println("Be sure to remove resources when you're done with them to avoid
 unexpected charges!")
 }
}
```

- API-Details finden Sie in den folgenden Themen der AWS SDK für Go -API-Referenz.
  - AdminCreateUser
  - AdminSetUserPassword
  - DeleteUser
  - InitiateAuth
  - UpdateUserPool

Amazon DocumentDB DocumentDB-Beispiele mit SDK for Go V2

Die folgenden Codebeispiele zeigen Ihnen, wie Sie mithilfe von AWS SDK für Go V2 mit Amazon DocumentDB Aktionen ausführen und allgemeine Szenarien implementieren.

Jedes Beispiel enthält einen Link zum vollständigen Quellcode, in dem Sie Anweisungen zur Einrichtung und Ausführung des Codes im Kontext finden.

Themen

- Serverless-Beispiele

Serverless-Beispiele

Aufrufen einer Lambda-Funktion über einen Amazon DocumentDB-Auslöser

Das folgende Codebeispiel zeigt, wie eine Lambda-Funktion implementiert wird, die ein Ereignis empfängt, das durch den Empfang von Datensätzen aus einem DocumentDB-Änderungsstream ausgelöst wird. Die Funktion ruft die DocumentDB-Nutzdaten ab und protokolliert den Inhalt des Datensatzes.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu. GitHub Das vollständige Beispiel sowie eine Anleitung zum Einrichten und Ausführen finden Sie im Repository mit Serverless-Beispielen.

Nutzen eines Amazon DocumentDB-Ereignisses mit Lambda unter Verwendung von Go.

```
package main

import (
 "context"
 "encoding/json"
 "fmt"

 "github.com/aws/aws-lambda-go/lambda"
)

type Event struct {
 Events []Record `json:"events"`
}

type Record struct {
 Event struct {
  OperationType string `json:"operationType"`
  NS            struct {
   DB   string `json:"db"`
   Coll string `json:"coll"`
  } `json:"ns"`
  FullDocument interface{} `json:"fullDocument"`
 } `json:"event"`
}

func main() {
 lambda.Start(handler)
}

func handler(ctx context.Context, event Event) (string, error) {
 fmt.Println("Loading function")
 for _, record := range event.Events {
```

```
  logDocumentDBEvent(record)
 }

 return "OK", nil
}

func logDocumentDBEvent(record Record) {
 fmt.Printf("Operation type: %s\n", record.Event.OperationType)
 fmt.Printf("db: %s\n", record.Event.NS.DB)
 fmt.Printf("collection: %s\n", record.Event.NS.Coll)
 docBytes, _ := json.MarshalIndent(record.Event.FullDocument, "", "  ")
 fmt.Printf("Full document: %s\n", string(docBytes))
}
```

DynamoDB-Beispiele mit SDK for Go V2

Die folgenden Codebeispiele zeigen Ihnen, wie Sie mithilfe von AWS SDK für Go V2 mit DynamoDB Aktionen ausführen und allgemeine Szenarien implementieren.

Bei Grundlagen handelt es sich um Code-Beispiele, die Ihnen zeigen, wie Sie die wesentlichen Vorgänge innerhalb eines Services ausführen.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Service-Funktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarios anzeigen.

Szenarien sind Code-Beispiele, die Ihnen zeigen, wie Sie bestimmte Aufgaben ausführen, indem Sie mehrere Funktionen innerhalb eines Services aufrufen oder mit anderen AWS-Services kombinieren.

AWS Community-Beiträge sind Beispiele, die von mehreren Teams erstellt wurden und verwaltet werden. AWS Um Feedback zu geben, verwende den Mechanismus, der in den verlinkten Repositorien zur Verfügung steht.

Jedes Beispiel enthält einen Link zum vollständigen Quellcode, in dem Sie Anweisungen zum Einrichten und Ausführen des Codes im Kontext finden.

Themen

- Grundlagen
- Aktionen

- Szenarien
- Serverless-Beispiele
- AWS Beiträge der Gemeinschaft

Grundlagen

Erlernen der Grundlagen

Wie das aussehen kann, sehen Sie am nachfolgenden Beispielcode:

- Erstellen einer Tabelle, die Filmdaten enthalten kann.
- Einfügen, Abrufen und Aktualisieren eines einzelnen Films in der Tabelle.
- Schreiben von Filmdaten in die Tabelle anhand einer JSON-Beispieldatei.
- Abfragen nach Filmen, die in einem bestimmten Jahr veröffentlicht wurden.
- Scan nach Filmen, die in mehreren Jahren veröffentlicht wurden.
- Löschen eines Films aus der Tabelle und anschließendes Löschen der Tabelle.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Führen Sie ein interaktives Szenario aus, um die Tabelle zu erstellen und Aktionen darauf auszuführen.

```
import (
 "context"
 "fmt"
 "log"
 "strings"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/awsdocs/aws-doc-sdk-examples/gov2/demotools"
```

```
 "github.com/awsdocs/aws-doc-sdk-examples/gov2/dynamodb/actions"
)

// RunMovieScenario is an interactive example that shows you how to use the AWS SDK
 for Go
// to create and use an Amazon DynamoDB table that stores data about movies.
//
//  1. Create a table that can hold movie data.
//  2. Put, get, and update a single movie in the table.
//  3. Write movie data to the table from a sample JSON file.
//  4. Query for movies that were released in a given year.
//  5. Scan for movies that were released in a range of years.
//  6. Delete a movie from the table.
//  7. Delete the table.
//
// This example creates a DynamoDB service client from the specified sdkConfig so
 that
// you can replace it with a mocked or stubbed config for unit testing.
//
// It uses a questioner from the `demotools` package to get input during the
 example.
// This package can be found in the ..\..\demotools folder of this repo.
//
// The specified movie sampler is used to get sample data from a URL that is loaded
// into the named table.
func RunMovieScenario(
 ctx context.Context, sdkConfig aws.Config, questioner demotools.IQuestioner,
 tableName string,
 movieSampler actions.IMovieSampler) {
 defer func() {
  if r := recover(); r != nil {
   fmt.Printf("Something went wrong with the demo.")
  }
 }()

 log.Println(strings.Repeat("-", 88))
 log.Println("Welcome to the Amazon DynamoDB getting started demo.")
 log.Println(strings.Repeat("-", 88))

 tableBasics := actions.TableBasics{TableName: tableName,
  DynamoDbClient: dynamodb.NewFromConfig(sdkConfig)}

 exists, err := tableBasics.TableExists(ctx)
 if err != nil {
```

```
 panic(err)
}
if !exists {
 log.Printf("Creating table %v...\n", tableName)
 _, err = tableBasics.CreateMovieTable(ctx)
 if err != nil {
  panic(err)
 } else {
  log.Printf("Created table %v.\n", tableName)
 }
} else {
 log.Printf("Table %v already exists.\n", tableName)
}

var customMovie actions.Movie
customMovie.Title = questioner.Ask("Enter a movie title to add to the table:",
 demotools.NotEmpty{})
customMovie.Year = questioner.AskInt("What year was it released?",
 demotools.NotEmpty{}, demotools.InIntRange{Lower: 1900, Upper: 2030})
customMovie.Info = map[string]interface{}{}
customMovie.Info["rating"] = questioner.AskFloat64(
 "Enter a rating between 1 and 10:",
 demotools.NotEmpty{}, demotools.InFloatRange{Lower: 1, Upper: 10})
customMovie.Info["plot"] = questioner.Ask("What's the plot? ",
 demotools.NotEmpty{})
err = tableBasics.AddMovie(ctx, customMovie)
if err == nil {
 log.Printf("Added %v to the movie table.\n", customMovie.Title)
}
log.Println(strings.Repeat("-", 88))

log.Printf("Let's update your movie. You previously rated it %v.\n",
customMovie.Info["rating"])
customMovie.Info["rating"] = questioner.AskFloat64(
 "What new rating would you give it?",
 demotools.NotEmpty{}, demotools.InFloatRange{Lower: 1, Upper: 10})
log.Printf("You summarized the plot as '%v'.\n", customMovie.Info["plot"])
customMovie.Info["plot"] = questioner.Ask("What would you say now?",
 demotools.NotEmpty{})
attributes, err := tableBasics.UpdateMovie(ctx, customMovie)
if err == nil {
 log.Printf("Updated %v with new values.\n", customMovie.Title)
 for _, attVal := range attributes {
  for valKey, val := range attVal {
```

```
    log.Printf("\t%v: %v\n", valKey, val)
   }
  }
 }
log.Println(strings.Repeat("-", 88))

log.Printf("Getting movie data from %v and adding 250 movies to the table...\n",
 movieSampler.GetURL())
movies := movieSampler.GetSampleMovies()
written, err := tableBasics.AddMovieBatch(ctx, movies, 250)
if err != nil {
 panic(err)
} else {
 log.Printf("Added %v movies to the table.\n", written)
}

show := 10
if show > written {
 show = written
}
log.Printf("The first %v movies in the table are:", show)
for index, movie := range movies[:show] {
 log.Printf("\t%v. %v\n", index+1, movie.Title)
}
movieIndex := questioner.AskInt(
 "Enter the number of a movie to get info about it: ",
 demotools.InIntRange{Lower: 1, Upper: show},
)
movie, err := tableBasics.GetMovie(ctx, movies[movieIndex-1].Title,
movies[movieIndex-1].Year)
if err == nil {
 log.Println(movie)
}
log.Println(strings.Repeat("-", 88))

log.Println("Let's get a list of movies released in a given year.")
releaseYear := questioner.AskInt("Enter a year between 1972 and 2018: ",
 demotools.InIntRange{Lower: 1972, Upper: 2018},
)
releases, err := tableBasics.Query(ctx, releaseYear)
if err == nil {
 if len(releases) == 0 {
  log.Printf("I couldn't find any movies released in %v!\n", releaseYear)
 } else {
```

```
  for _, movie = range releases {
   log.Println(movie)
  }
 }
}
log.Println(strings.Repeat("-", 88))

log.Println("Now let's scan for movies released in a range of years.")
startYear := questioner.AskInt("Enter a year: ",
 demotools.InIntRange{Lower: 1972, Upper: 2018})
endYear := questioner.AskInt("Enter another year: ",
 demotools.InIntRange{Lower: 1972, Upper: 2018})
releases, err = tableBasics.Scan(ctx, startYear, endYear)
if err == nil {
 if len(releases) == 0 {
  log.Printf("I couldn't find any movies released between %v and %v!\n", startYear,
endYear)
 } else {
  log.Printf("Found %v movies. In this list, the plot is <nil> because "+
   "we used a projection expression when scanning for items to return only "+
   "the title, year, and rating.\n", len(releases))
  for _, movie = range releases {
   log.Println(movie)
  }
 }
}
log.Println(strings.Repeat("-", 88))

var tables []string
if questioner.AskBool("Do you want to list all of your tables? (y/n) ", "y") {
 tables, err = tableBasics.ListTables(ctx)
 if err == nil {
  log.Printf("Found %v tables:", len(tables))
  for _, table := range tables {
   log.Printf("\t%v", table)
  }
 }
}
log.Println(strings.Repeat("-", 88))

log.Printf("Let's remove your movie '%v'.\n", customMovie.Title)
if questioner.AskBool("Do you want to delete it from the table? (y/n) ", "y") {
 err = tableBasics.DeleteMovie(ctx, customMovie)
}
```

```
 if err == nil {
  log.Printf("Deleted %v.\n", customMovie.Title)
 }

 if questioner.AskBool("Delete the table, too? (y/n)", "y") {
  err = tableBasics.DeleteTable(ctx)
 } else {
  log.Println("Don't forget to delete the table when you're done or you might " +
   "incur charges on your account.")
 }
 if err == nil {
  log.Printf("Deleted table %v.\n", tableBasics.TableName)
 }

 log.Println(strings.Repeat("-", 88))
 log.Println("Thanks for watching!")
 log.Println(strings.Repeat("-", 88))
}
```

Definieren Sie eine Movie-Struktur, die in diesem Beispiel verwendet wird.

```
import (
 "archive/zip"
 "bytes"
 "encoding/json"
 "fmt"
 "io"
 "log"
 "net/http"

 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// Movie encapsulates data about a movie. Title and Year are the composite primary
 key
// of the movie in Amazon DynamoDB. Title is the sort key, Year is the partition
 key,
// and Info is additional data.
type Movie struct {
```

```
 Title string                 `dynamodbav:"title"`
 Year  int                    `dynamodbav:"year"`
 Info  map[string]interface{} `dynamodbav:"info"`
}

// GetKey returns the composite primary key of the movie in a format that can be
// sent to DynamoDB.
func (movie Movie) GetKey() map[string]types.AttributeValue {
 title, err := attributevalue.Marshal(movie.Title)
 if err != nil {
  panic(err)
 }
 year, err := attributevalue.Marshal(movie.Year)
 if err != nil {
  panic(err)
 }
 return map[string]types.AttributeValue{"title": title, "year": year}
}

// String returns the title, year, rating, and plot of a movie, formatted for the
 example.
func (movie Movie) String() string {
 return fmt.Sprintf("%v\n\tReleased: %v\n\tRating: %v\n\tPlot: %v\n",
  movie.Title, movie.Year, movie.Info["rating"], movie.Info["plot"])
}
```

Erstellen Sie eine Struktur und Methoden, die DynamoDB-Aktionen aufrufen.

```
import (
 "context"
 "errors"
 "log"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/expression"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)
```

```
// TableBasics encapsulates the Amazon DynamoDB service actions used in the
 examples.
// It contains a DynamoDB service client that is used to act on the specified table.
type TableBasics struct {
 DynamoDbClient *dynamodb.Client
 TableName      string
}



// TableExists determines whether a DynamoDB table exists.
func (basics TableBasics) TableExists(ctx context.Context) (bool, error) {
 exists := true
 _, err := basics.DynamoDbClient.DescribeTable(
  ctx, &dynamodb.DescribeTableInput{TableName: aws.String(basics.TableName)},
 )
 if err != nil {
  var notFoundEx *types.ResourceNotFoundException
  if errors.As(err, &notFoundEx) {
   log.Printf("Table %v does not exist.\n", basics.TableName)
   err = nil
  } else {
   log.Printf("Couldn't determine existence of table %v. Here's why: %v\n",
 basics.TableName, err)
  }
  exists = false
 }
 return exists, err
}



// CreateMovieTable creates a DynamoDB table with a composite primary key defined as
// a string sort key named `title`, and a numeric partition key named `year`.
// This function uses NewTableExistsWaiter to wait for the table to be created by
// DynamoDB before it returns.
func (basics TableBasics) CreateMovieTable(ctx context.Context)
 (*types.TableDescription, error) {
 var tableDesc *types.TableDescription
 table, err := basics.DynamoDbClient.CreateTable(ctx, &dynamodb.CreateTableInput{
  AttributeDefinitions: []types.AttributeDefinition{{
   AttributeName: aws.String("year"),
   AttributeType: types.ScalarAttributeTypeN,
```

```
  }, {
   AttributeName: aws.String("title"),
   AttributeType: types.ScalarAttributeTypeS,
  }},
  KeySchema: []types.KeySchemaElement{{
   AttributeName: aws.String("year"),
   KeyType:       types.KeyTypeHash,
  }, {
   AttributeName: aws.String("title"),
   KeyType:       types.KeyTypeRange,
  }},
  TableName:   aws.String(basics.TableName),
  BillingMode: types.BillingModePayPerRequest,
 })
 if err != nil {
  log.Printf("Couldn't create table %v. Here's why: %v\n", basics.TableName, err)
 } else {
  waiter := dynamodb.NewTableExistsWaiter(basics.DynamoDbClient)
  err = waiter.Wait(ctx, &dynamodb.DescribeTableInput{
   TableName: aws.String(basics.TableName)}, 5*time.Minute)
  if err != nil {
   log.Printf("Wait for table exists failed. Here's why: %v\n", err)
  }
  tableDesc = table.TableDescription
  log.Printf("Ccreating table test")
 }
 return tableDesc, err
}



// ListTables lists the DynamoDB table names for the current account.
func (basics TableBasics) ListTables(ctx context.Context) ([]string, error) {
 var tableNames []string
 var output *dynamodb.ListTablesOutput
 var err error
 tablePaginator := dynamodb.NewListTablesPaginator(basics.DynamoDbClient,
 &dynamodb.ListTablesInput{})
 for tablePaginator.HasMorePages() {
  output, err = tablePaginator.NextPage(ctx)
  if err != nil {
   log.Printf("Couldn't list tables. Here's why: %v\n", err)
   break
  } else {
```

```
    tableNames = append(tableNames, output.TableNames...)
   }
  }
  return tableNames, err
}



// AddMovie adds a movie the DynamoDB table.
func (basics TableBasics) AddMovie(ctx context.Context, movie Movie) error {
  item, err := attributevalue.MarshalMap(movie)
  if err != nil {
   panic(err)
  }
  _, err = basics.DynamoDbClient.PutItem(ctx, &dynamodb.PutItemInput{
   TableName: aws.String(basics.TableName), Item: item,
  })
  if err != nil {
   log.Printf("Couldn't add item to table. Here's why: %v\n", err)
  }
  return err
}



// UpdateMovie updates the rating and plot of a movie that already exists in the
// DynamoDB table. This function uses the `expression` package to build the update
// expression.
func (basics TableBasics) UpdateMovie(ctx context.Context, movie Movie)
  (map[string]map[string]interface{}, error) {
  var err error
  var response *dynamodb.UpdateItemOutput
  var attributeMap map[string]map[string]interface{}
  update := expression.Set(expression.Name("info.rating"),
  expression.Value(movie.Info["rating"]))
  update.Set(expression.Name("info.plot"), expression.Value(movie.Info["plot"]))
  expr, err := expression.NewBuilder().WithUpdate(update).Build()
  if err != nil {
   log.Printf("Couldn't build expression for update. Here's why: %v\n", err)
  } else {
   response, err = basics.DynamoDbClient.UpdateItem(ctx, &dynamodb.UpdateItemInput{
    TableName:                 aws.String(basics.TableName),
    Key:                       movie.GetKey(),
    ExpressionAttributeNames:  expr.Names(),
```

```
    ExpressionAttributeValues: expr.Values(),
    UpdateExpression:          expr.Update(),
    ReturnValues:              types.ReturnValueUpdatedNew,
   })
   if err != nil {
    log.Printf("Couldn't update movie %v. Here's why: %v\n", movie.Title, err)
   } else {
    err = attributevalue.UnmarshalMap(response.Attributes, &attributeMap)
    if err != nil {
     log.Printf("Couldn't unmarshall update response. Here's why: %v\n", err)
    }
   }
  }
  return attributeMap, err
}



// AddMovieBatch adds a slice of movies to the DynamoDB table. The function sends
// batches of 25 movies to DynamoDB until all movies are added or it reaches the
// specified maximum.
func (basics TableBasics) AddMovieBatch(ctx context.Context, movies []Movie,
 maxMovies int) (int, error) {
 var err error
 var item map[string]types.AttributeValue
 written := 0
 batchSize := 25 // DynamoDB allows a maximum batch size of 25 items.
 start := 0
 end := start + batchSize
 for start < maxMovies && start < len(movies) {
  var writeReqs []types.WriteRequest
  if end > len(movies) {
   end = len(movies)
  }
  for _, movie := range movies[start:end] {
   item, err = attributevalue.MarshalMap(movie)
   if err != nil {
    log.Printf("Couldn't marshal movie %v for batch writing. Here's why: %v\n",
 movie.Title, err)
   } else {
    writeReqs = append(
     writeReqs,
     types.WriteRequest{PutRequest: &types.PutRequest{Item: item}},
    )
```

```
   }
  }
  _, err = basics.DynamoDbClient.BatchWriteItem(ctx, &dynamodb.BatchWriteItemInput{
   RequestItems: map[string][]types.WriteRequest{basics.TableName: writeReqs}})
  if err != nil {
   log.Printf("Couldn't add a batch of movies to %v. Here's why: %v\n",
 basics.TableName, err)
  } else {
   written += len(writeReqs)
  }
  start = end
  end += batchSize
 }

 return written, err
}



// GetMovie gets movie data from the DynamoDB table by using the primary composite
 key
// made of title and year.
func (basics TableBasics) GetMovie(ctx context.Context, title string, year int)
 (Movie, error) {
 movie := Movie{Title: title, Year: year}
 response, err := basics.DynamoDbClient.GetItem(ctx, &dynamodb.GetItemInput{
  Key: movie.GetKey(), TableName: aws.String(basics.TableName),
 })
 if err != nil {
  log.Printf("Couldn't get info about %v. Here's why: %v\n", title, err)
 } else {
  err = attributevalue.UnmarshalMap(response.Item, &movie)
  if err != nil {
   log.Printf("Couldn't unmarshal response. Here's why: %v\n", err)
  }
 }
 return movie, err
}



// Query gets all movies in the DynamoDB table that were released in the specified
 year.
// The function uses the `expression` package to build the key condition expression
```

```
// that is used in the query.
func (basics TableBasics) Query(ctx context.Context, releaseYear int) ([]Movie,
 error) {
 var err error
 var response *dynamodb.QueryOutput
 var movies []Movie
 keyEx := expression.Key("year").Equal(expression.Value(releaseYear))
 expr, err := expression.NewBuilder().WithKeyCondition(keyEx).Build()
 if err != nil {
  log.Printf("Couldn't build expression for query. Here's why: %v\n", err)
 } else {
  queryPaginator := dynamodb.NewQueryPaginator(basics.DynamoDbClient,
 &dynamodb.QueryInput{
   TableName:                 aws.String(basics.TableName),
   ExpressionAttributeNames:  expr.Names(),
   ExpressionAttributeValues: expr.Values(),
   KeyConditionExpression:    expr.KeyCondition(),
  })
  for queryPaginator.HasMorePages() {
   response, err = queryPaginator.NextPage(ctx)
   if err != nil {
    log.Printf("Couldn't query for movies released in %v. Here's why: %v\n",
 releaseYear, err)
    break
   } else {
    var moviePage []Movie
    err = attributevalue.UnmarshalListOfMaps(response.Items, &moviePage)
    if err != nil {
     log.Printf("Couldn't unmarshal query response. Here's why: %v\n", err)
     break
    } else {
     movies = append(movies, moviePage...)
    }
   }
  }
 }
 return movies, err
}



// Scan gets all movies in the DynamoDB table that were released in a range of years
// and projects them to return a reduced set of fields.
// The function uses the `expression` package to build the filter and projection
```

```
// expressions.
func (basics TableBasics) Scan(ctx context.Context, startYear int, endYear int)
 ([]Movie, error) {
 var movies []Movie
 var err error
 var response *dynamodb.ScanOutput
 filtEx := expression.Name("year").Between(expression.Value(startYear),
 expression.Value(endYear))
 projEx := expression.NamesList(
  expression.Name("year"), expression.Name("title"), expression.Name("info.rating"))
 expr, err :=
 expression.NewBuilder().WithFilter(filtEx).WithProjection(projEx).Build()
 if err != nil {
  log.Printf("Couldn't build expressions for scan. Here's why: %v\n", err)
 } else {
  scanPaginator := dynamodb.NewScanPaginator(basics.DynamoDbClient,
 &dynamodb.ScanInput{
   TableName:                 aws.String(basics.TableName),
   ExpressionAttributeNames:  expr.Names(),
   ExpressionAttributeValues: expr.Values(),
   FilterExpression:          expr.Filter(),
   ProjectionExpression:      expr.Projection(),
  })
  for scanPaginator.HasMorePages() {
   response, err = scanPaginator.NextPage(ctx)
   if err != nil {
    log.Printf("Couldn't scan for movies released between %v and %v. Here's why: %v
\n",
     startYear, endYear, err)
    break
   } else {
    var moviePage []Movie
    err = attributevalue.UnmarshalListOfMaps(response.Items, &moviePage)
    if err != nil {
     log.Printf("Couldn't unmarshal query response. Here's why: %v\n", err)
     break
    } else {
     movies = append(movies, moviePage...)
    }
   }
  }
 }
 return movies, err
}
```

```
// DeleteMovie removes a movie from the DynamoDB table.
func (basics TableBasics) DeleteMovie(ctx context.Context, movie Movie) error {
 _, err := basics.DynamoDbClient.DeleteItem(ctx, &dynamodb.DeleteItemInput{
  TableName: aws.String(basics.TableName), Key: movie.GetKey(),
 })
 if err != nil {
  log.Printf("Couldn't delete %v from the table. Here's why: %v\n", movie.Title,
 err)
 }
 return err
}



// DeleteTable deletes the DynamoDB table and all of its data.
func (basics TableBasics) DeleteTable(ctx context.Context) error {
 _, err := basics.DynamoDbClient.DeleteTable(ctx, &dynamodb.DeleteTableInput{
  TableName: aws.String(basics.TableName)})
 if err != nil {
  log.Printf("Couldn't delete table %v. Here's why: %v\n", basics.TableName, err)
 }
 return err
}
```

- API-Details finden Sie in den folgenden Themen der AWS SDK für Go -API-Referenz.
  - BatchWriteItem
  - CreateTable
  - DeleteItem
  - DeleteTable
  - DescribeTable
  - GetItem
  - PutItem
  - Abfrage
  - Scan

- UpdateItem

Aktionen

`BatchExecuteStatement`

Das folgende Codebeispiel zeigt, wie man es benutzt`BatchExecuteStatement`.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Definieren Sie eine Funktionsempfängerstruktur für das Beispiel.

```
import (
 "context"
 "fmt"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// PartiQLRunner encapsulates the Amazon DynamoDB service actions used in the
// PartiQL examples. It contains a DynamoDB service client that is used to act on
 the
// specified table.
type PartiQLRunner struct {
 DynamoDbClient *dynamodb.Client
 TableName      string
}
```

Verwenden Sie Stapel von INSERT-Anweisungen, um Elemente hinzuzufügen.

```
// AddMovieBatch runs a batch of PartiQL INSERT statements to add multiple movies to
 the
// DynamoDB table.
func (runner PartiQLRunner) AddMovieBatch(ctx context.Context, movies []Movie) error
 {
 statementRequests := make([]types.BatchStatementRequest, len(movies))
 for index, movie := range movies {
  params, err := attributevalue.MarshalList([]interface{}{movie.Title, movie.Year,
 movie.Info})
  if err != nil {
   panic(err)
  }
  statementRequests[index] = types.BatchStatementRequest{
   Statement: aws.String(fmt.Sprintf(
    "INSERT INTO \"%v\" VALUE {'title': ?, 'year': ?, 'info': ?}",
 runner.TableName)),
   Parameters: params,
  }
 }

 _, err := runner.DynamoDbClient.BatchExecuteStatement(ctx,
 &dynamodb.BatchExecuteStatementInput{
  Statements: statementRequests,
 })
 if err != nil {
  log.Printf("Couldn't insert a batch of items with PartiQL. Here's why: %v\n", err)
 }
 return err
}
```

Verwenden Sie Stapel von SELECT-Anweisungen, um Elemente abzurufen.

```
// GetMovieBatch runs a batch of PartiQL SELECT statements to get multiple movies
 from
// the DynamoDB table by title and year.
func (runner PartiQLRunner) GetMovieBatch(ctx context.Context, movies []Movie)
 ([]Movie, error) {
```

```
 statementRequests := make([]types.BatchStatementRequest, len(movies))
 for index, movie := range movies {
  params, err := attributevalue.MarshalList([]interface{}{movie.Title, movie.Year})
  if err != nil {
   panic(err)
  }
  statementRequests[index] = types.BatchStatementRequest{
   Statement: aws.String(
    fmt.Sprintf("SELECT * FROM \"%v\" WHERE title=? AND year=?", runner.TableName)),
   Parameters: params,
  }
 }

 output, err := runner.DynamoDbClient.BatchExecuteStatement(ctx,
 &dynamodb.BatchExecuteStatementInput{
  Statements: statementRequests,
 })
 var outMovies []Movie
 if err != nil {
  log.Printf("Couldn't get a batch of items with PartiQL. Here's why: %v\n", err)
 } else {
  for _, response := range output.Responses {
   var movie Movie
   err = attributevalue.UnmarshalMap(response.Item, &movie)
   if err != nil {
    log.Printf("Couldn't unmarshal response. Here's why: %v\n", err)
   } else {
    outMovies = append(outMovies, movie)
   }
  }
 }
 return outMovies, err
}
```

Verwenden Sie Stapel von UPDATE-Anweisungen, um Elemente zu aktualisieren.

```
// UpdateMovieBatch runs a batch of PartiQL UPDATE statements to update the rating
 of
// multiple movies that already exist in the DynamoDB table.
```

```
func (runner PartiQLRunner) UpdateMovieBatch(ctx context.Context, movies []Movie,
 ratings []float64) error {
 statementRequests := make([]types.BatchStatementRequest, len(movies))
 for index, movie := range movies {
  params, err := attributevalue.MarshalList([]interface{}{ratings[index],
 movie.Title, movie.Year})
  if err != nil {
   panic(err)
  }
  statementRequests[index] = types.BatchStatementRequest{
   Statement: aws.String(
    fmt.Sprintf("UPDATE \"%v\" SET info.rating=? WHERE title=? AND year=?",
 runner.TableName)),
   Parameters: params,
  }
 }

 _, err := runner.DynamoDbClient.BatchExecuteStatement(ctx,
 &dynamodb.BatchExecuteStatementInput{
  Statements: statementRequests,
 })
 if err != nil {
  log.Printf("Couldn't update the batch of movies. Here's why: %v\n", err)
 }
 return err
}
```

Verwenden Sie Stapel von DELETE-Anweisungen, um Elemente zu löschen.

```
// DeleteMovieBatch runs a batch of PartiQL DELETE statements to remove multiple
 movies
// from the DynamoDB table.
func (runner PartiQLRunner) DeleteMovieBatch(ctx context.Context, movies []Movie)
 error {
 statementRequests := make([]types.BatchStatementRequest, len(movies))
 for index, movie := range movies {
  params, err := attributevalue.MarshalList([]interface{}{movie.Title, movie.Year})
  if err != nil {
   panic(err)
  }
```

```
  statementRequests[index] = types.BatchStatementRequest{
   Statement: aws.String(
    fmt.Sprintf("DELETE FROM \"%v\" WHERE title=? AND year=?", runner.TableName)),
   Parameters: params,
  }
 }

 _, err := runner.DynamoDbClient.BatchExecuteStatement(ctx,
 &dynamodb.BatchExecuteStatementInput{
  Statements: statementRequests,
 })
 if err != nil {
  log.Printf("Couldn't delete the batch of movies. Here's why: %v\n", err)
 }
 return err
}
```

Definieren Sie eine Movie-Struktur, die in diesem Beispiel verwendet wird.

```
import (
 "archive/zip"
 "bytes"
 "encoding/json"
 "fmt"
 "io"
 "log"
 "net/http"

 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// Movie encapsulates data about a movie. Title and Year are the composite primary
 key
// of the movie in Amazon DynamoDB. Title is the sort key, Year is the partition
 key,
// and Info is additional data.
type Movie struct {
 Title string                `dynamodbav:"title"`
 Year  int                   `dynamodbav:"year"`
```

```
 Info  map[string]interface{} `dynamodbav:"info"`
}

// GetKey returns the composite primary key of the movie in a format that can be
// sent to DynamoDB.
func (movie Movie) GetKey() map[string]types.AttributeValue {
 title, err := attributevalue.Marshal(movie.Title)
 if err != nil {
  panic(err)
 }
 year, err := attributevalue.Marshal(movie.Year)
 if err != nil {
  panic(err)
 }
 return map[string]types.AttributeValue{"title": title, "year": year}
}

// String returns the title, year, rating, and plot of a movie, formatted for the
 example.
func (movie Movie) String() string {
 return fmt.Sprintf("%v\n\tReleased: %v\n\tRating: %v\n\tPlot: %v\n",
  movie.Title, movie.Year, movie.Info["rating"], movie.Info["plot"])
}
```

- Einzelheiten zur API finden Sie BatchExecuteStatementunter AWS SDK für Go API-Referenz.

`BatchWriteItem`

Das folgende Codebeispiel zeigt die Verwendung`BatchWriteItem`.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import (
```

```
 "context"
 "errors"
 "log"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/expression"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// TableBasics encapsulates the Amazon DynamoDB service actions used in the
 examples.
// It contains a DynamoDB service client that is used to act on the specified table.
type TableBasics struct {
 DynamoDbClient *dynamodb.Client
 TableName      string
}



// AddMovieBatch adds a slice of movies to the DynamoDB table. The function sends
// batches of 25 movies to DynamoDB until all movies are added or it reaches the
// specified maximum.
func (basics TableBasics) AddMovieBatch(ctx context.Context, movies []Movie,
 maxMovies int) (int, error) {
 var err error
 var item map[string]types.AttributeValue
 written := 0
 batchSize := 25 // DynamoDB allows a maximum batch size of 25 items.
 start := 0
 end := start + batchSize
 for start < maxMovies && start < len(movies) {
  var writeReqs []types.WriteRequest
  if end > len(movies) {
   end = len(movies)
  }
  for _, movie := range movies[start:end] {
   item, err = attributevalue.MarshalMap(movie)
   if err != nil {
    log.Printf("Couldn't marshal movie %v for batch writing. Here's why: %v\n",
 movie.Title, err)
   } else {
```

```
    writeReqs = append(
     writeReqs,
     types.WriteRequest{PutRequest: &types.PutRequest{Item: item}},
    )
   }
  }
  _, err = basics.DynamoDbClient.BatchWriteItem(ctx, &dynamodb.BatchWriteItemInput{
   RequestItems: map[string][]types.WriteRequest{basics.TableName: writeReqs}})
  if err != nil {
   log.Printf("Couldn't add a batch of movies to %v. Here's why: %v\n",
 basics.TableName, err)
  } else {
   written += len(writeReqs)
  }
  start = end
  end += batchSize
 }

 return written, err
}
```

Definieren Sie eine Movie-Struktur, die in diesem Beispiel verwendet wird.

```
import (
 "archive/zip"
 "bytes"
 "encoding/json"
 "fmt"
 "io"
 "log"
 "net/http"

 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// Movie encapsulates data about a movie. Title and Year are the composite primary
 key
// of the movie in Amazon DynamoDB. Title is the sort key, Year is the partition
 key,
```

```
// and Info is additional data.
type Movie struct {
 Title string                 `dynamodbav:"title"`
 Year  int                    `dynamodbav:"year"`
 Info  map[string]interface{} `dynamodbav:"info"`
}

// GetKey returns the composite primary key of the movie in a format that can be
// sent to DynamoDB.
func (movie Movie) GetKey() map[string]types.AttributeValue {
 title, err := attributevalue.Marshal(movie.Title)
 if err != nil {
  panic(err)
 }
 year, err := attributevalue.Marshal(movie.Year)
 if err != nil {
  panic(err)
 }
 return map[string]types.AttributeValue{"title": title, "year": year}
}

// String returns the title, year, rating, and plot of a movie, formatted for the
 example.
func (movie Movie) String() string {
 return fmt.Sprintf("%v\n\tReleased: %v\n\tRating: %v\n\tPlot: %v\n",
  movie.Title, movie.Year, movie.Info["rating"], movie.Info["plot"])
}
```

- Einzelheiten zur API finden Sie BatchWriteItemunter AWS SDK für Go API-Referenz.

`CreateTable`

Das folgende Codebeispiel zeigt die Verwendung`CreateTable`.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import (
 "context"
 "errors"
 "log"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/expression"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// TableBasics encapsulates the Amazon DynamoDB service actions used in the
 examples.
// It contains a DynamoDB service client that is used to act on the specified table.
type TableBasics struct {
 DynamoDbClient *dynamodb.Client
 TableName      string
}



// CreateMovieTable creates a DynamoDB table with a composite primary key defined as
// a string sort key named `title`, and a numeric partition key named `year`.
// This function uses NewTableExistsWaiter to wait for the table to be created by
// DynamoDB before it returns.
func (basics TableBasics) CreateMovieTable(ctx context.Context)
 (*types.TableDescription, error) {
 var tableDesc *types.TableDescription
 table, err := basics.DynamoDbClient.CreateTable(ctx, &dynamodb.CreateTableInput{
  AttributeDefinitions: []types.AttributeDefinition{{
   AttributeName: aws.String("year"),
   AttributeType: types.ScalarAttributeTypeN,
  }, {
   AttributeName: aws.String("title"),
   AttributeType: types.ScalarAttributeTypeS,
  }},
  KeySchema: []types.KeySchemaElement{{
   AttributeName: aws.String("year"),
   KeyType:       types.KeyTypeHash,
  }, {
```

```
   AttributeName: aws.String("title"),
   KeyType:       types.KeyTypeRange,
  }},
  TableName:   aws.String(basics.TableName),
  BillingMode: types.BillingModePayPerRequest,
 })
 if err != nil {
  log.Printf("Couldn't create table %v. Here's why: %v\n", basics.TableName, err)
 } else {
  waiter := dynamodb.NewTableExistsWaiter(basics.DynamoDbClient)
  err = waiter.Wait(ctx, &dynamodb.DescribeTableInput{
   TableName: aws.String(basics.TableName)}, 5*time.Minute)
  if err != nil {
   log.Printf("Wait for table exists failed. Here's why: %v\n", err)
  }
  tableDesc = table.TableDescription
  log.Printf("Ccreating table test")
 }
 return tableDesc, err
}
```

- Einzelheiten zur API finden Sie CreateTablein der AWS SDK für Go API-Referenz.

`DeleteItem`

Das folgende Codebeispiel zeigt die Verwendung`DeleteItem`.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import (
 "context"
 "errors"
 "log"
```

```
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/expression"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// TableBasics encapsulates the Amazon DynamoDB service actions used in the
 examples.
// It contains a DynamoDB service client that is used to act on the specified table.
type TableBasics struct {
 DynamoDbClient *dynamodb.Client
 TableName      string
}



// DeleteMovie removes a movie from the DynamoDB table.
func (basics TableBasics) DeleteMovie(ctx context.Context, movie Movie) error {
 _, err := basics.DynamoDbClient.DeleteItem(ctx, &dynamodb.DeleteItemInput{
  TableName: aws.String(basics.TableName), Key: movie.GetKey(),
 })
 if err != nil {
  log.Printf("Couldn't delete %v from the table. Here's why: %v\n", movie.Title,
 err)
 }
 return err
}
```

Definieren Sie eine Movie-Struktur, die in diesem Beispiel verwendet wird.

```
import (
 "archive/zip"
 "bytes"
 "encoding/json"
 "fmt"
 "io"
 "log"
```

```
 "net/http"

 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// Movie encapsulates data about a movie. Title and Year are the composite primary
 key
// of the movie in Amazon DynamoDB. Title is the sort key, Year is the partition
 key,
// and Info is additional data.
type Movie struct {
 Title string                 `dynamodbav:"title"`
 Year  int                    `dynamodbav:"year"`
 Info  map[string]interface{} `dynamodbav:"info"`
}

// GetKey returns the composite primary key of the movie in a format that can be
// sent to DynamoDB.
func (movie Movie) GetKey() map[string]types.AttributeValue {
 title, err := attributevalue.Marshal(movie.Title)
 if err != nil {
  panic(err)
 }
 year, err := attributevalue.Marshal(movie.Year)
 if err != nil {
  panic(err)
 }
 return map[string]types.AttributeValue{"title": title, "year": year}
}

// String returns the title, year, rating, and plot of a movie, formatted for the
 example.
func (movie Movie) String() string {
 return fmt.Sprintf("%v\n\tReleased: %v\n\tRating: %v\n\tPlot: %v\n",
  movie.Title, movie.Year, movie.Info["rating"], movie.Info["plot"])
}
```

- Einzelheiten zur API finden Sie DeleteItemunter AWS SDK für Go API-Referenz.

`DeleteTable`

Das folgende Codebeispiel zeigt die Verwendung`DeleteTable`.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import (
 "context"
 "errors"
 "log"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/expression"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// TableBasics encapsulates the Amazon DynamoDB service actions used in the
 examples.
// It contains a DynamoDB service client that is used to act on the specified table.
type TableBasics struct {
 DynamoDbClient *dynamodb.Client
 TableName      string
}



// DeleteTable deletes the DynamoDB table and all of its data.
func (basics TableBasics) DeleteTable(ctx context.Context) error {
 _, err := basics.DynamoDbClient.DeleteTable(ctx, &dynamodb.DeleteTableInput{
  TableName: aws.String(basics.TableName)})
 if err != nil {
  log.Printf("Couldn't delete table %v. Here's why: %v\n", basics.TableName, err)
 }
```

```
 return err
}
```

- Einzelheiten zur API finden Sie DeleteTablein der AWS SDK für Go API-Referenz.

`DescribeTable`

Das folgende Codebeispiel zeigt die Verwendung`DescribeTable`.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import (
 "context"
 "errors"
 "log"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/expression"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// TableBasics encapsulates the Amazon DynamoDB service actions used in the
 examples.
// It contains a DynamoDB service client that is used to act on the specified table.
type TableBasics struct {
 DynamoDbClient *dynamodb.Client
 TableName      string
}
```

```
// TableExists determines whether a DynamoDB table exists.
func (basics TableBasics) TableExists(ctx context.Context) (bool, error) {
 exists := true
 _, err := basics.DynamoDbClient.DescribeTable(
  ctx, &dynamodb.DescribeTableInput{TableName: aws.String(basics.TableName)},
 )
 if err != nil {
  var notFoundEx *types.ResourceNotFoundException
  if errors.As(err, &notFoundEx) {
   log.Printf("Table %v does not exist.\n", basics.TableName)
   err = nil
  } else {
   log.Printf("Couldn't determine existence of table %v. Here's why: %v\n",
 basics.TableName, err)
  }
  exists = false
 }
 return exists, err
}
```

- Einzelheiten zur API finden Sie DescribeTablein der AWS SDK für Go API-Referenz.

`ExecuteStatement`

Das folgende Codebeispiel zeigt die Verwendung`ExecuteStatement`.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Definieren Sie eine Funktionsempfängerstruktur für das Beispiel.

```
import (
 "context"
```

```
 "fmt"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// PartiQLRunner encapsulates the Amazon DynamoDB service actions used in the
// PartiQL examples. It contains a DynamoDB service client that is used to act on
 the
// specified table.
type PartiQLRunner struct {
 DynamoDbClient *dynamodb.Client
 TableName      string
}
```

Verwenden Sie eine INSERT-Anweisung, um ein Element hinzuzufügen.

```
// AddMovie runs a PartiQL INSERT statement to add a movie to the DynamoDB table.
func (runner PartiQLRunner) AddMovie(ctx context.Context, movie Movie) error {
 params, err := attributevalue.MarshalList([]interface{}{movie.Title, movie.Year,
 movie.Info})
 if err != nil {
  panic(err)
 }
 _, err = runner.DynamoDbClient.ExecuteStatement(ctx,
 &dynamodb.ExecuteStatementInput{
  Statement: aws.String(
   fmt.Sprintf("INSERT INTO \"%v\" VALUE {'title': ?, 'year': ?, 'info': ?}",
    runner.TableName)),
  Parameters: params,
 })
 if err != nil {
  log.Printf("Couldn't insert an item with PartiQL. Here's why: %v\n", err)
 }
 return err
}
```

Verwenden Sie eine SELECT-Anweisung, um ein Element abzurufen.

```
// GetMovie runs a PartiQL SELECT statement to get a movie from the DynamoDB table
 by
// title and year.
func (runner PartiQLRunner) GetMovie(ctx context.Context, title string, year int)
 (Movie, error) {
 var movie Movie
 params, err := attributevalue.MarshalList([]interface{}{title, year})
 if err != nil {
  panic(err)
 }
 response, err := runner.DynamoDbClient.ExecuteStatement(ctx,
 &dynamodb.ExecuteStatementInput{
  Statement: aws.String(
   fmt.Sprintf("SELECT * FROM \"%v\" WHERE title=? AND year=?",
    runner.TableName)),
  Parameters: params,
 })
 if err != nil {
  log.Printf("Couldn't get info about %v. Here's why: %v\n", title, err)
 } else {
  err = attributevalue.UnmarshalMap(response.Items[0], &movie)
  if err != nil {
   log.Printf("Couldn't unmarshal response. Here's why: %v\n", err)
  }
 }
 return movie, err
}
```

Verwenden Sie eine SELECT-Anweisung, um eine Liste der Elemente abzurufen und die Ergebnisse zu projizieren.

```
// GetAllMovies runs a PartiQL SELECT statement to get all movies from the DynamoDB
 table.
// pageSize is not typically required and is used to show how to paginate the
 results.
```

```
// The results are projected to return only the title and rating of each movie.
func (runner PartiQLRunner) GetAllMovies(ctx context.Context, pageSize int32)
 ([]map[string]interface{}, error) {
 var output []map[string]interface{}
 var response *dynamodb.ExecuteStatementOutput
 var err error
 var nextToken *string
 for moreData := true; moreData; {
  response, err = runner.DynamoDbClient.ExecuteStatement(ctx,
 &dynamodb.ExecuteStatementInput{
   Statement: aws.String(
    fmt.Sprintf("SELECT title, info.rating FROM \"%v\"", runner.TableName)),
   Limit:     aws.Int32(pageSize),
   NextToken: nextToken,
  })
  if err != nil {
   log.Printf("Couldn't get movies. Here's why: %v\n", err)
   moreData = false
  } else {
   var pageOutput []map[string]interface{}
   err = attributevalue.UnmarshalListOfMaps(response.Items, &pageOutput)
   if err != nil {
    log.Printf("Couldn't unmarshal response. Here's why: %v\n", err)
   } else {
    log.Printf("Got a page of length %v.\n", len(response.Items))
    output = append(output, pageOutput...)
   }
   nextToken = response.NextToken
   moreData = nextToken != nil
  }
 }
 return output, err
}
```

Verwenden Sie eine UPDATE-Anweisung, um ein Element zu aktualisieren.

```
// UpdateMovie runs a PartiQL UPDATE statement to update the rating of a movie that
// already exists in the DynamoDB table.
func (runner PartiQLRunner) UpdateMovie(ctx context.Context, movie Movie, rating
 float64) error {
```

```
 params, err := attributevalue.MarshalList([]interface{}{rating, movie.Title,
 movie.Year})
 if err != nil {
  panic(err)
 }
 _, err = runner.DynamoDbClient.ExecuteStatement(ctx,
 &dynamodb.ExecuteStatementInput{
  Statement: aws.String(
   fmt.Sprintf("UPDATE \"%v\" SET info.rating=? WHERE title=? AND year=?",
    runner.TableName)),
  Parameters: params,
 })
 if err != nil {
  log.Printf("Couldn't update movie %v. Here's why: %v\n", movie.Title, err)
 }
 return err
}
```

Verwenden Sie eine DELETE-Anweisung, um ein Element zu löschen.

```
// DeleteMovie runs a PartiQL DELETE statement to remove a movie from the DynamoDB
 table.
func (runner PartiQLRunner) DeleteMovie(ctx context.Context, movie Movie) error {
 params, err := attributevalue.MarshalList([]interface{}{movie.Title, movie.Year})
 if err != nil {
  panic(err)
 }
 _, err = runner.DynamoDbClient.ExecuteStatement(ctx,
 &dynamodb.ExecuteStatementInput{
  Statement: aws.String(
   fmt.Sprintf("DELETE FROM \"%v\" WHERE title=? AND year=?",
    runner.TableName)),
  Parameters: params,
 })
 if err != nil {
  log.Printf("Couldn't delete %v from the table. Here's why: %v\n", movie.Title,
 err)
 }
 return err
}
```

Definieren Sie eine Movie-Struktur, die in diesem Beispiel verwendet wird.

```
import (
 "archive/zip"
 "bytes"
 "encoding/json"
 "fmt"
 "io"
 "log"
 "net/http"

 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// Movie encapsulates data about a movie. Title and Year are the composite primary
 key
// of the movie in Amazon DynamoDB. Title is the sort key, Year is the partition
 key,
// and Info is additional data.
type Movie struct {
 Title string                 `dynamodbav:"title"`
 Year  int                    `dynamodbav:"year"`
 Info  map[string]interface{} `dynamodbav:"info"`
}

// GetKey returns the composite primary key of the movie in a format that can be
// sent to DynamoDB.
func (movie Movie) GetKey() map[string]types.AttributeValue {
 title, err := attributevalue.Marshal(movie.Title)
 if err != nil {
  panic(err)
 }
 year, err := attributevalue.Marshal(movie.Year)
 if err != nil {
  panic(err)
 }
 return map[string]types.AttributeValue{"title": title, "year": year}
}
```

```
// String returns the title, year, rating, and plot of a movie, formatted for the
 example.
func (movie Movie) String() string {
 return fmt.Sprintf("%v\n\tReleased: %v\n\tRating: %v\n\tPlot: %v\n",
  movie.Title, movie.Year, movie.Info["rating"], movie.Info["plot"])
}
```

- Einzelheiten zur API finden Sie ExecuteStatementunter AWS SDK für Go API-Referenz.

`GetItem`

Das folgende Codebeispiel zeigt die Verwendung`GetItem`.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import (
 "context"
 "errors"
 "log"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/expression"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// TableBasics encapsulates the Amazon DynamoDB service actions used in the
 examples.
// It contains a DynamoDB service client that is used to act on the specified table.
type TableBasics struct {
```

```
 DynamoDbClient *dynamodb.Client
 TableName      string
}



// GetMovie gets movie data from the DynamoDB table by using the primary composite
 key
// made of title and year.
func (basics TableBasics) GetMovie(ctx context.Context, title string, year int)
 (Movie, error) {
 movie := Movie{Title: title, Year: year}
 response, err := basics.DynamoDbClient.GetItem(ctx, &dynamodb.GetItemInput{
  Key: movie.GetKey(), TableName: aws.String(basics.TableName),
 })
 if err != nil {
  log.Printf("Couldn't get info about %v. Here's why: %v\n", title, err)
 } else {
  err = attributevalue.UnmarshalMap(response.Item, &movie)
  if err != nil {
   log.Printf("Couldn't unmarshal response. Here's why: %v\n", err)
  }
 }
 return movie, err
}
```

Definieren Sie eine Movie-Struktur, die in diesem Beispiel verwendet wird.

```
import (
 "archive/zip"
 "bytes"
 "encoding/json"
 "fmt"
 "io"
 "log"
 "net/http"

 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)
```

```
// Movie encapsulates data about a movie. Title and Year are the composite primary
 key
// of the movie in Amazon DynamoDB. Title is the sort key, Year is the partition
 key,
// and Info is additional data.
type Movie struct {
 Title string                 `dynamodbav:"title"`
 Year  int                    `dynamodbav:"year"`
 Info  map[string]interface{} `dynamodbav:"info"`
}

// GetKey returns the composite primary key of the movie in a format that can be
// sent to DynamoDB.
func (movie Movie) GetKey() map[string]types.AttributeValue {
 title, err := attributevalue.Marshal(movie.Title)
 if err != nil {
  panic(err)
 }
 year, err := attributevalue.Marshal(movie.Year)
 if err != nil {
  panic(err)
 }
 return map[string]types.AttributeValue{"title": title, "year": year}
}

// String returns the title, year, rating, and plot of a movie, formatted for the
 example.
func (movie Movie) String() string {
 return fmt.Sprintf("%v\n\tReleased: %v\n\tRating: %v\n\tPlot: %v\n",
  movie.Title, movie.Year, movie.Info["rating"], movie.Info["plot"])
}
```

- Einzelheiten zur API finden Sie GetItemunter AWS SDK für Go API-Referenz.

`ListTables`

Das folgende Codebeispiel zeigt die Verwendung`ListTables`.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import (
 "context"
 "errors"
 "log"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/expression"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// TableBasics encapsulates the Amazon DynamoDB service actions used in the
 examples.
// It contains a DynamoDB service client that is used to act on the specified table.
type TableBasics struct {
 DynamoDbClient *dynamodb.Client
 TableName      string
}



// ListTables lists the DynamoDB table names for the current account.
func (basics TableBasics) ListTables(ctx context.Context) ([]string, error) {
 var tableNames []string
 var output *dynamodb.ListTablesOutput
 var err error
 tablePaginator := dynamodb.NewListTablesPaginator(basics.DynamoDbClient,
 &dynamodb.ListTablesInput{})
 for tablePaginator.HasMorePages() {
  output, err = tablePaginator.NextPage(ctx)
  if err != nil {
```

```
    log.Printf("Couldn't list tables. Here's why: %v\n", err)
    break
   } else {
    tableNames = append(tableNames, output.TableNames...)
   }
  }
  return tableNames, err
}
```

- Einzelheiten zur API finden Sie ListTablesin der AWS SDK für Go API-Referenz.

`PutItem`

Das folgende Codebeispiel zeigt die Verwendung`PutItem`.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import (
 "context"
 "errors"
 "log"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/expression"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// TableBasics encapsulates the Amazon DynamoDB service actions used in the
 examples.
```

```
// It contains a DynamoDB service client that is used to act on the specified table.
type TableBasics struct {
 DynamoDbClient *dynamodb.Client
 TableName      string
}



// AddMovie adds a movie the DynamoDB table.
func (basics TableBasics) AddMovie(ctx context.Context, movie Movie) error {
 item, err := attributevalue.MarshalMap(movie)
 if err != nil {
  panic(err)
 }
 _, err = basics.DynamoDbClient.PutItem(ctx, &dynamodb.PutItemInput{
  TableName: aws.String(basics.TableName), Item: item,
 })
 if err != nil {
  log.Printf("Couldn't add item to table. Here's why: %v\n", err)
 }
 return err
}
```

Definieren Sie eine Movie-Struktur, die in diesem Beispiel verwendet wird.

```
import (
 "archive/zip"
 "bytes"
 "encoding/json"
 "fmt"
 "io"
 "log"
 "net/http"

 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// Movie encapsulates data about a movie. Title and Year are the composite primary
 key
```

```
// of the movie in Amazon DynamoDB. Title is the sort key, Year is the partition
 key,
// and Info is additional data.
type Movie struct {
 Title string                 `dynamodbav:"title"`
 Year  int                    `dynamodbav:"year"`
 Info  map[string]interface{} `dynamodbav:"info"`
}

// GetKey returns the composite primary key of the movie in a format that can be
// sent to DynamoDB.
func (movie Movie) GetKey() map[string]types.AttributeValue {
 title, err := attributevalue.Marshal(movie.Title)
 if err != nil {
  panic(err)
 }
 year, err := attributevalue.Marshal(movie.Year)
 if err != nil {
  panic(err)
 }
 return map[string]types.AttributeValue{"title": title, "year": year}
}

// String returns the title, year, rating, and plot of a movie, formatted for the
 example.
func (movie Movie) String() string {
 return fmt.Sprintf("%v\n\tReleased: %v\n\tRating: %v\n\tPlot: %v\n",
  movie.Title, movie.Year, movie.Info["rating"], movie.Info["plot"])
}
```

- Einzelheiten zur API finden Sie PutItemunter AWS SDK für Go API-Referenz.

`Query`

Das folgende Codebeispiel zeigt die Verwendung`Query`.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import (
 "context"
 "errors"
 "log"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/expression"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// TableBasics encapsulates the Amazon DynamoDB service actions used in the
 examples.
// It contains a DynamoDB service client that is used to act on the specified table.
type TableBasics struct {
 DynamoDbClient *dynamodb.Client
 TableName      string
}



// Query gets all movies in the DynamoDB table that were released in the specified
 year.
// The function uses the `expression` package to build the key condition expression
// that is used in the query.
func (basics TableBasics) Query(ctx context.Context, releaseYear int) ([]Movie,
 error) {
 var err error
 var response *dynamodb.QueryOutput
 var movies []Movie
 keyEx := expression.Key("year").Equal(expression.Value(releaseYear))
```

```
 expr, err := expression.NewBuilder().WithKeyCondition(keyEx).Build()
 if err != nil {
  log.Printf("Couldn't build expression for query. Here's why: %v\n", err)
 } else {
  queryPaginator := dynamodb.NewQueryPaginator(basics.DynamoDbClient,
 &dynamodb.QueryInput{
   TableName:                 aws.String(basics.TableName),
   ExpressionAttributeNames:  expr.Names(),
   ExpressionAttributeValues: expr.Values(),
   KeyConditionExpression:    expr.KeyCondition(),
  })
  for queryPaginator.HasMorePages() {
   response, err = queryPaginator.NextPage(ctx)
   if err != nil {
    log.Printf("Couldn't query for movies released in %v. Here's why: %v\n",
 releaseYear, err)
    break
   } else {
    var moviePage []Movie
    err = attributevalue.UnmarshalListOfMaps(response.Items, &moviePage)
    if err != nil {
     log.Printf("Couldn't unmarshal query response. Here's why: %v\n", err)
     break
    } else {
     movies = append(movies, moviePage...)
    }
   }
  }
 }
 return movies, err
}
```

Definieren Sie eine Movie-Struktur, die in diesem Beispiel verwendet wird.

```
import (
 "archive/zip"
 "bytes"
 "encoding/json"
 "fmt"
 "io"
```

```
 "log"
 "net/http"

 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// Movie encapsulates data about a movie. Title and Year are the composite primary
 key
// of the movie in Amazon DynamoDB. Title is the sort key, Year is the partition
 key,
// and Info is additional data.
type Movie struct {
 Title string                 `dynamodbav:"title"`
 Year  int                    `dynamodbav:"year"`
 Info  map[string]interface{} `dynamodbav:"info"`
}

// GetKey returns the composite primary key of the movie in a format that can be
// sent to DynamoDB.
func (movie Movie) GetKey() map[string]types.AttributeValue {
 title, err := attributevalue.Marshal(movie.Title)
 if err != nil {
  panic(err)
 }
 year, err := attributevalue.Marshal(movie.Year)
 if err != nil {
  panic(err)
 }
 return map[string]types.AttributeValue{"title": title, "year": year}
}

// String returns the title, year, rating, and plot of a movie, formatted for the
 example.
func (movie Movie) String() string {
 return fmt.Sprintf("%v\n\tReleased: %v\n\tRating: %v\n\tPlot: %v\n",
  movie.Title, movie.Year, movie.Info["rating"], movie.Info["plot"])
}
```

- Weitere API-Informationen finden Sie unter Query in der AWS SDK für Go -API-Referenz.

Scan

Das folgende Codebeispiel zeigt, wie man es benutztScan.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import (
 "context"
 "errors"
 "log"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/expression"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// TableBasics encapsulates the Amazon DynamoDB service actions used in the
 examples.
// It contains a DynamoDB service client that is used to act on the specified table.
type TableBasics struct {
 DynamoDbClient *dynamodb.Client
 TableName      string
}



// Scan gets all movies in the DynamoDB table that were released in a range of years
// and projects them to return a reduced set of fields.
// The function uses the `expression` package to build the filter and projection
// expressions.
func (basics TableBasics) Scan(ctx context.Context, startYear int, endYear int)
 ([]Movie, error) {
 var movies []Movie
```

```
 var err error
 var response *dynamodb.ScanOutput
 filtEx := expression.Name("year").Between(expression.Value(startYear),
 expression.Value(endYear))
 projEx := expression.NamesList(
  expression.Name("year"), expression.Name("title"), expression.Name("info.rating"))
 expr, err :=
 expression.NewBuilder().WithFilter(filtEx).WithProjection(projEx).Build()
 if err != nil {
  log.Printf("Couldn't build expressions for scan. Here's why: %v\n", err)
 } else {
  scanPaginator := dynamodb.NewScanPaginator(basics.DynamoDbClient,
 &dynamodb.ScanInput{
   TableName:                 aws.String(basics.TableName),
   ExpressionAttributeNames:  expr.Names(),
   ExpressionAttributeValues: expr.Values(),
   FilterExpression:          expr.Filter(),
   ProjectionExpression:      expr.Projection(),
  })
  for scanPaginator.HasMorePages() {
   response, err = scanPaginator.NextPage(ctx)
   if err != nil {
    log.Printf("Couldn't scan for movies released between %v and %v. Here's why: %v
\n",
     startYear, endYear, err)
    break
   } else {
    var moviePage []Movie
    err = attributevalue.UnmarshalListOfMaps(response.Items, &moviePage)
    if err != nil {
     log.Printf("Couldn't unmarshal query response. Here's why: %v\n", err)
     break
    } else {
     movies = append(movies, moviePage...)
    }
   }
  }
 }
 return movies, err
}
```

Definieren Sie eine Movie-Struktur, die in diesem Beispiel verwendet wird.

```
import (
 "archive/zip"
 "bytes"
 "encoding/json"
 "fmt"
 "io"
 "log"
 "net/http"

 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// Movie encapsulates data about a movie. Title and Year are the composite primary
 key
// of the movie in Amazon DynamoDB. Title is the sort key, Year is the partition
 key,
// and Info is additional data.
type Movie struct {
 Title string                 `dynamodbav:"title"`
 Year  int                    `dynamodbav:"year"`
 Info  map[string]interface{} `dynamodbav:"info"`
}

// GetKey returns the composite primary key of the movie in a format that can be
// sent to DynamoDB.
func (movie Movie) GetKey() map[string]types.AttributeValue {
 title, err := attributevalue.Marshal(movie.Title)
 if err != nil {
  panic(err)
 }
 year, err := attributevalue.Marshal(movie.Year)
 if err != nil {
  panic(err)
 }
 return map[string]types.AttributeValue{"title": title, "year": year}
}

// String returns the title, year, rating, and plot of a movie, formatted for the
 example.
func (movie Movie) String() string {
```

```
 return fmt.Sprintf("%v\n\tReleased: %v\n\tRating: %v\n\tPlot: %v\n",
  movie.Title, movie.Year, movie.Info["rating"], movie.Info["plot"])
}
```

- Weitere API-Informationen finden Sie unter Scan in der AWS SDK für Go -API-Referenz.

`UpdateItem`

Das folgende Codebeispiel zeigt, wie man es benutzt`UpdateItem`.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import (
 "context"
 "errors"
 "log"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/expression"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// TableBasics encapsulates the Amazon DynamoDB service actions used in the
 examples.
// It contains a DynamoDB service client that is used to act on the specified table.
type TableBasics struct {
 DynamoDbClient *dynamodb.Client
 TableName      string
}
```

```
// UpdateMovie updates the rating and plot of a movie that already exists in the
// DynamoDB table. This function uses the `expression` package to build the update
// expression.
func (basics TableBasics) UpdateMovie(ctx context.Context, movie Movie)
 (map[string]map[string]interface{}, error) {
 var err error
 var response *dynamodb.UpdateItemOutput
 var attributeMap map[string]map[string]interface{}
 update := expression.Set(expression.Name("info.rating"),
 expression.Value(movie.Info["rating"]))
 update.Set(expression.Name("info.plot"), expression.Value(movie.Info["plot"]))
 expr, err := expression.NewBuilder().WithUpdate(update).Build()
 if err != nil {
  log.Printf("Couldn't build expression for update. Here's why: %v\n", err)
 } else {
  response, err = basics.DynamoDbClient.UpdateItem(ctx, &dynamodb.UpdateItemInput{
   TableName:                 aws.String(basics.TableName),
   Key:                       movie.GetKey(),
   ExpressionAttributeNames:  expr.Names(),
   ExpressionAttributeValues: expr.Values(),
   UpdateExpression:          expr.Update(),
   ReturnValues:              types.ReturnValueUpdatedNew,
  })
  if err != nil {
   log.Printf("Couldn't update movie %v. Here's why: %v\n", movie.Title, err)
  } else {
   err = attributevalue.UnmarshalMap(response.Attributes, &attributeMap)
   if err != nil {
    log.Printf("Couldn't unmarshall update response. Here's why: %v\n", err)
   }
  }
 }
 return attributeMap, err
}
```

Definieren Sie eine Movie-Struktur, die in diesem Beispiel verwendet wird.

```
import (
```

```
 "archive/zip"
 "bytes"
 "encoding/json"
 "fmt"
 "io"
 "log"
 "net/http"

 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// Movie encapsulates data about a movie. Title and Year are the composite primary
 key
// of the movie in Amazon DynamoDB. Title is the sort key, Year is the partition
 key,
// and Info is additional data.
type Movie struct {
 Title string                 `dynamodbav:"title"`
 Year  int                    `dynamodbav:"year"`
 Info  map[string]interface{} `dynamodbav:"info"`
}

// GetKey returns the composite primary key of the movie in a format that can be
// sent to DynamoDB.
func (movie Movie) GetKey() map[string]types.AttributeValue {
 title, err := attributevalue.Marshal(movie.Title)
 if err != nil {
  panic(err)
 }
 year, err := attributevalue.Marshal(movie.Year)
 if err != nil {
  panic(err)
 }
 return map[string]types.AttributeValue{"title": title, "year": year}
}

// String returns the title, year, rating, and plot of a movie, formatted for the
 example.
func (movie Movie) String() string {
 return fmt.Sprintf("%v\n\tReleased: %v\n\tRating: %v\n\tPlot: %v\n",
  movie.Title, movie.Year, movie.Info["rating"], movie.Info["plot"])
}
```

- Einzelheiten zur API finden Sie UpdateItemunter AWS SDK für Go API-Referenz.

Szenarien

Abfragen einer Tabelle mithilfe von Stapeln von PartiQL-Anweisungen

Wie das aussehen kann, sehen Sie am nachfolgenden Beispielcode:

- Abrufen eines Stapels von Elementen mithilfe mehrerer SELECT-Anweisungen.
- Hinzufügen eines Stapels von Elementen hinzu, indem mehrere INSERT-Anweisungen ausgeführt werden.
- Aktualisieren eines Stapels von Elementen mithilfe mehrerer UPDATE-Anweisungen.
- Löschen eines Stapels von Elementen mithilfe mehrerer DELETE-Anweisungen.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Führen Sie ein Szenario aus, das eine Tabelle erstellt und Stapel von PartiQL-Abfragen ausführt.

```
import (
 "context"
 "fmt"
 "log"
 "strings"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/awsdocs/aws-doc-sdk-examples/gov2/dynamodb/actions"
)
```

```
// RunPartiQLBatchScenario shows you how to use the AWS SDK for Go
// to run batches of PartiQL statements to query a table that stores data about
 movies.
//
//   - Use batches of PartiQL statements to add, get, update, and delete data for
//     individual movies.
//
// This example creates an Amazon DynamoDB service client from the specified
 sdkConfig so that
// you can replace it with a mocked or stubbed config for unit testing.
//
// This example creates and deletes a DynamoDB table to use during the scenario.
func RunPartiQLBatchScenario(ctx context.Context, sdkConfig aws.Config, tableName
 string) {
 defer func() {
  if r := recover(); r != nil {
   fmt.Printf("Something went wrong with the demo.")
  }
 }()

 log.Println(strings.Repeat("-", 88))
 log.Println("Welcome to the Amazon DynamoDB PartiQL batch demo.")
 log.Println(strings.Repeat("-", 88))

 tableBasics := actions.TableBasics{
  DynamoDbClient: dynamodb.NewFromConfig(sdkConfig),
  TableName:      tableName,
 }
 runner := actions.PartiQLRunner{
  DynamoDbClient: dynamodb.NewFromConfig(sdkConfig),
  TableName:      tableName,
 }

 exists, err := tableBasics.TableExists(ctx)
 if err != nil {
  panic(err)
 }
 if !exists {
  log.Printf("Creating table %v...\n", tableName)
  _, err = tableBasics.CreateMovieTable(ctx)
  if err != nil {
   panic(err)
  } else {
   log.Printf("Created table %v.\n", tableName)
```

```
 }
} else {
 log.Printf("Table %v already exists.\n", tableName)
}
log.Println(strings.Repeat("-", 88))

currentYear, _, _ := time.Now().Date()
customMovies := []actions.Movie{{
 Title: "House PartiQL",
 Year:  currentYear - 5,
 Info: map[string]interface{}{
  "plot":   "Wacky high jinks result from querying a mysterious database.",
  "rating": 8.5}}, {
 Title: "House PartiQL 2",
 Year:  currentYear - 3,
 Info: map[string]interface{}{
  "plot":   "Moderate high jinks result from querying another mysterious
database.",
  "rating": 6.5}}, {
 Title: "House PartiQL 3",
 Year:  currentYear - 1,
 Info: map[string]interface{}{
  "plot":   "Tepid high jinks result from querying yet another mysterious
database.",
  "rating": 2.5},
},
}

log.Printf("Inserting a batch of movies into table '%v'.\n", tableName)
err = runner.AddMovieBatch(ctx, customMovies)
if err == nil {
 log.Printf("Added %v movies to the table.\n", len(customMovies))
}
log.Println(strings.Repeat("-", 88))

log.Println("Getting data for a batch of movies.")
movies, err := runner.GetMovieBatch(ctx, customMovies)
if err == nil {
 for _, movie := range movies {
  log.Println(movie)
 }
}
log.Println(strings.Repeat("-", 88))
```

```
 newRatings := []float64{7.7, 4.4, 1.1}
 log.Println("Updating a batch of movies with new ratings.")
 err = runner.UpdateMovieBatch(ctx, customMovies, newRatings)
 if err == nil {
  log.Printf("Updated %v movies with new ratings.\n", len(customMovies))
 }
 log.Println(strings.Repeat("-", 88))

 log.Println("Getting projected data from the table to verify our update.")
 log.Println("Using a page size of 2 to demonstrate paging.")
 projections, err := runner.GetAllMovies(ctx, 2)
 if err == nil {
  log.Println("All movies:")
  for _, projection := range projections {
   log.Println(projection)
  }
 }
 log.Println(strings.Repeat("-", 88))

 log.Println("Deleting a batch of movies.")
 err = runner.DeleteMovieBatch(ctx, customMovies)
 if err == nil {
  log.Printf("Deleted %v movies.\n", len(customMovies))
 }

 err = tableBasics.DeleteTable(ctx)
 if err == nil {
  log.Printf("Deleted table %v.\n", tableBasics.TableName)
 }

 log.Println(strings.Repeat("-", 88))
 log.Println("Thanks for watching!")
 log.Println(strings.Repeat("-", 88))
}
```

Definieren Sie eine Movie-Struktur, die in diesem Beispiel verwendet wird.

```
import (
 "archive/zip"
 "bytes"
```

```
 "encoding/json"
 "fmt"
 "io"
 "log"
 "net/http"

 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// Movie encapsulates data about a movie. Title and Year are the composite primary
 key
// of the movie in Amazon DynamoDB. Title is the sort key, Year is the partition
 key,
// and Info is additional data.
type Movie struct {
 Title string                 `dynamodbav:"title"`
 Year  int                    `dynamodbav:"year"`
 Info  map[string]interface{} `dynamodbav:"info"`
}

// GetKey returns the composite primary key of the movie in a format that can be
// sent to DynamoDB.
func (movie Movie) GetKey() map[string]types.AttributeValue {
 title, err := attributevalue.Marshal(movie.Title)
 if err != nil {
  panic(err)
 }
 year, err := attributevalue.Marshal(movie.Year)
 if err != nil {
  panic(err)
 }
 return map[string]types.AttributeValue{"title": title, "year": year}
}

// String returns the title, year, rating, and plot of a movie, formatted for the
 example.
func (movie Movie) String() string {
 return fmt.Sprintf("%v\n\tReleased: %v\n\tRating: %v\n\tPlot: %v\n",
  movie.Title, movie.Year, movie.Info["rating"], movie.Info["plot"])
}
```

Erstellen Sie eine Struktur und Methoden, die PartiQL-Anweisungen ausführen.

```
import (
 "context"
 "fmt"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// PartiQLRunner encapsulates the Amazon DynamoDB service actions used in the
// PartiQL examples. It contains a DynamoDB service client that is used to act on
 the
// specified table.
type PartiQLRunner struct {
 DynamoDbClient *dynamodb.Client
 TableName      string
}



// AddMovieBatch runs a batch of PartiQL INSERT statements to add multiple movies to
 the
// DynamoDB table.
func (runner PartiQLRunner) AddMovieBatch(ctx context.Context, movies []Movie) error
 {
 statementRequests := make([]types.BatchStatementRequest, len(movies))
 for index, movie := range movies {
  params, err := attributevalue.MarshalList([]interface{}{movie.Title, movie.Year,
 movie.Info})
  if err != nil {
   panic(err)
  }
  statementRequests[index] = types.BatchStatementRequest{
   Statement: aws.String(fmt.Sprintf(
    "INSERT INTO \"%v\" VALUE {'title': ?, 'year': ?, 'info': ?}",
 runner.TableName)),
   Parameters: params,
  }
 }
```

```
 _, err := runner.DynamoDbClient.BatchExecuteStatement(ctx,
 &dynamodb.BatchExecuteStatementInput{
  Statements: statementRequests,
 })
 if err != nil {
  log.Printf("Couldn't insert a batch of items with PartiQL. Here's why: %v\n", err)
 }
 return err
}



// GetMovieBatch runs a batch of PartiQL SELECT statements to get multiple movies
 from
// the DynamoDB table by title and year.
func (runner PartiQLRunner) GetMovieBatch(ctx context.Context, movies []Movie)
 ([]Movie, error) {
 statementRequests := make([]types.BatchStatementRequest, len(movies))
 for index, movie := range movies {
  params, err := attributevalue.MarshalList([]interface{}{movie.Title, movie.Year})
  if err != nil {
   panic(err)
  }
  statementRequests[index] = types.BatchStatementRequest{
   Statement: aws.String(
    fmt.Sprintf("SELECT * FROM \"%v\" WHERE title=? AND year=?", runner.TableName)),
   Parameters: params,
  }
 }

 output, err := runner.DynamoDbClient.BatchExecuteStatement(ctx,
 &dynamodb.BatchExecuteStatementInput{
  Statements: statementRequests,
 })
 var outMovies []Movie
 if err != nil {
  log.Printf("Couldn't get a batch of items with PartiQL. Here's why: %v\n", err)
 } else {
  for _, response := range output.Responses {
   var movie Movie
   err = attributevalue.UnmarshalMap(response.Item, &movie)
   if err != nil {
    log.Printf("Couldn't unmarshal response. Here's why: %v\n", err)
```

```
   } else {
    outMovies = append(outMovies, movie)
   }
  }
 }
 return outMovies, err
}



// GetAllMovies runs a PartiQL SELECT statement to get all movies from the DynamoDB
 table.
// pageSize is not typically required and is used to show how to paginate the
 results.
// The results are projected to return only the title and rating of each movie.
func (runner PartiQLRunner) GetAllMovies(ctx context.Context, pageSize int32)
 ([]map[string]interface{}, error) {
 var output []map[string]interface{}
 var response *dynamodb.ExecuteStatementOutput
 var err error
 var nextToken *string
 for moreData := true; moreData; {
  response, err = runner.DynamoDbClient.ExecuteStatement(ctx,
 &dynamodb.ExecuteStatementInput{
   Statement: aws.String(
    fmt.Sprintf("SELECT title, info.rating FROM \"%v\"", runner.TableName)),
   Limit:     aws.Int32(pageSize),
   NextToken: nextToken,
  })
  if err != nil {
   log.Printf("Couldn't get movies. Here's why: %v\n", err)
   moreData = false
  } else {
   var pageOutput []map[string]interface{}
   err = attributevalue.UnmarshalListOfMaps(response.Items, &pageOutput)
   if err != nil {
    log.Printf("Couldn't unmarshal response. Here's why: %v\n", err)
   } else {
    log.Printf("Got a page of length %v.\n", len(response.Items))
    output = append(output, pageOutput...)
   }
   nextToken = response.NextToken
   moreData = nextToken != nil
  }
```

```
 }
 return output, err
}



// UpdateMovieBatch runs a batch of PartiQL UPDATE statements to update the rating
 of
// multiple movies that already exist in the DynamoDB table.
func (runner PartiQLRunner) UpdateMovieBatch(ctx context.Context, movies []Movie,
 ratings []float64) error {
 statementRequests := make([]types.BatchStatementRequest, len(movies))
 for index, movie := range movies {
  params, err := attributevalue.MarshalList([]interface{}{ratings[index],
 movie.Title, movie.Year})
  if err != nil {
   panic(err)
  }
  statementRequests[index] = types.BatchStatementRequest{
   Statement: aws.String(
    fmt.Sprintf("UPDATE \"%v\" SET info.rating=? WHERE title=? AND year=?",
 runner.TableName)),
   Parameters: params,
  }
 }

 _, err := runner.DynamoDbClient.BatchExecuteStatement(ctx,
 &dynamodb.BatchExecuteStatementInput{
  Statements: statementRequests,
 })
 if err != nil {
  log.Printf("Couldn't update the batch of movies. Here's why: %v\n", err)
 }
 return err
}



// DeleteMovieBatch runs a batch of PartiQL DELETE statements to remove multiple
 movies
// from the DynamoDB table.
func (runner PartiQLRunner) DeleteMovieBatch(ctx context.Context, movies []Movie)
 error {
 statementRequests := make([]types.BatchStatementRequest, len(movies))
```

```
 for index, movie := range movies {
  params, err := attributevalue.MarshalList([]interface{}{movie.Title, movie.Year})
  if err != nil {
   panic(err)
  }
  statementRequests[index] = types.BatchStatementRequest{
   Statement: aws.String(
    fmt.Sprintf("DELETE FROM \"%v\" WHERE title=? AND year=?", runner.TableName)),
   Parameters: params,
  }
 }

 _, err := runner.DynamoDbClient.BatchExecuteStatement(ctx,
 &dynamodb.BatchExecuteStatementInput{
  Statements: statementRequests,
 })
 if err != nil {
  log.Printf("Couldn't delete the batch of movies. Here's why: %v\n", err)
 }
 return err
}
```

- Einzelheiten zur API finden Sie BatchExecuteStatementunter AWS SDK für Go API-Referenz.

Abfragen einer Tabelle mit PartiQL

Wie das aussehen kann, sehen Sie am nachfolgenden Beispielcode:

- Abrufen eines Elementes durch Ausführen einer SELECT-Anweisung.
- Hinzufügen eines Elementes durch Ausführung einer INSERT-Anweisung.
- Aktualisieren eines Elementes durch Ausführung einer UPDATE-Anweisung.
- Löschen eines Elementes durch Ausführung einer DELETE-Anweisung.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Führen Sie ein Szenario aus, das eine Tabelle erstellt und PartiQL-Abfragen ausführt.

```
import (
 "context"
 "fmt"
 "log"
 "strings"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/awsdocs/aws-doc-sdk-examples/gov2/dynamodb/actions"
)

// RunPartiQLSingleScenario shows you how to use the AWS SDK for Go
// to use PartiQL to query a table that stores data about movies.
//
// * Use PartiQL statements to add, get, update, and delete data for individual
 movies.
//
// This example creates an Amazon DynamoDB service client from the specified
 sdkConfig so that
// you can replace it with a mocked or stubbed config for unit testing.
//
// This example creates and deletes a DynamoDB table to use during the scenario.
func RunPartiQLSingleScenario(ctx context.Context, sdkConfig aws.Config, tableName
 string) {
 defer func() {
  if r := recover(); r != nil {
   fmt.Printf("Something went wrong with the demo.")
  }
 }()

 log.Println(strings.Repeat("-", 88))
```

```
log.Println("Welcome to the Amazon DynamoDB PartiQL single action demo.")
log.Println(strings.Repeat("-", 88))

tableBasics := actions.TableBasics{
 DynamoDbClient: dynamodb.NewFromConfig(sdkConfig),
 TableName:      tableName,
}
runner := actions.PartiQLRunner{
 DynamoDbClient: dynamodb.NewFromConfig(sdkConfig),
 TableName:      tableName,
}

exists, err := tableBasics.TableExists(ctx)
if err != nil {
 panic(err)
}
if !exists {
 log.Printf("Creating table %v...\n", tableName)
 _, err = tableBasics.CreateMovieTable(ctx)
 if err != nil {
  panic(err)
 } else {
  log.Printf("Created table %v.\n", tableName)
 }
} else {
 log.Printf("Table %v already exists.\n", tableName)
}
log.Println(strings.Repeat("-", 88))

currentYear, _, _ := time.Now().Date()
customMovie := actions.Movie{
 Title: "24 Hour PartiQL People",
 Year:  currentYear,
 Info: map[string]interface{}{
  "plot":   "A group of data developers discover a new query language they can't
stop using.",
  "rating": 9.9,
 },
}

log.Printf("Inserting movie '%v' released in %v.", customMovie.Title,
customMovie.Year)
err = runner.AddMovie(ctx, customMovie)
if err == nil {
```

```
  log.Printf("Added %v to the movie table.\n", customMovie.Title)
 }
 log.Println(strings.Repeat("-", 88))

 log.Printf("Getting data for movie '%v' released in %v.", customMovie.Title,
 customMovie.Year)
 movie, err := runner.GetMovie(ctx, customMovie.Title, customMovie.Year)
 if err == nil {
  log.Println(movie)
 }
 log.Println(strings.Repeat("-", 88))

 newRating := 6.6
 log.Printf("Updating movie '%v' with a rating of %v.", customMovie.Title,
 newRating)
 err = runner.UpdateMovie(ctx, customMovie, newRating)
 if err == nil {
  log.Printf("Updated %v with a new rating.\n", customMovie.Title)
 }
 log.Println(strings.Repeat("-", 88))

 log.Printf("Getting data again to verify the update.")
 movie, err = runner.GetMovie(ctx, customMovie.Title, customMovie.Year)
 if err == nil {
  log.Println(movie)
 }
 log.Println(strings.Repeat("-", 88))

 log.Printf("Deleting movie '%v'.\n", customMovie.Title)
 err = runner.DeleteMovie(ctx, customMovie)
 if err == nil {
  log.Printf("Deleted %v.\n", customMovie.Title)
 }

 err = tableBasics.DeleteTable(ctx)
 if err == nil {
  log.Printf("Deleted table %v.\n", tableBasics.TableName)
 }

 log.Println(strings.Repeat("-", 88))
 log.Println("Thanks for watching!")
 log.Println(strings.Repeat("-", 88))
}
```

Definieren Sie eine Movie-Struktur, die in diesem Beispiel verwendet wird.

```
import (
 "archive/zip"
 "bytes"
 "encoding/json"
 "fmt"
 "io"
 "log"
 "net/http"

 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// Movie encapsulates data about a movie. Title and Year are the composite primary
 key
// of the movie in Amazon DynamoDB. Title is the sort key, Year is the partition
 key,
// and Info is additional data.
type Movie struct {
 Title string                 `dynamodbav:"title"`
 Year  int                    `dynamodbav:"year"`
 Info  map[string]interface{} `dynamodbav:"info"`
}

// GetKey returns the composite primary key of the movie in a format that can be
// sent to DynamoDB.
func (movie Movie) GetKey() map[string]types.AttributeValue {
 title, err := attributevalue.Marshal(movie.Title)
 if err != nil {
  panic(err)
 }
 year, err := attributevalue.Marshal(movie.Year)
 if err != nil {
  panic(err)
 }
 return map[string]types.AttributeValue{"title": title, "year": year}
}
```

```
// String returns the title, year, rating, and plot of a movie, formatted for the
 example.
func (movie Movie) String() string {
 return fmt.Sprintf("%v\n\tReleased: %v\n\tRating: %v\n\tPlot: %v\n",
  movie.Title, movie.Year, movie.Info["rating"], movie.Info["plot"])
}
```

Erstellen Sie eine Struktur und Methoden, die PartiQL-Anweisungen ausführen.

```
import (
 "context"
 "fmt"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// PartiQLRunner encapsulates the Amazon DynamoDB service actions used in the
// PartiQL examples. It contains a DynamoDB service client that is used to act on
 the
// specified table.
type PartiQLRunner struct {
 DynamoDbClient *dynamodb.Client
 TableName      string
}



// AddMovie runs a PartiQL INSERT statement to add a movie to the DynamoDB table.
func (runner PartiQLRunner) AddMovie(ctx context.Context, movie Movie) error {
 params, err := attributevalue.MarshalList([]interface{}{movie.Title, movie.Year,
 movie.Info})
 if err != nil {
  panic(err)
 }
 _, err = runner.DynamoDbClient.ExecuteStatement(ctx,
 &dynamodb.ExecuteStatementInput{
```

```
  Statement: aws.String(
   fmt.Sprintf("INSERT INTO \"%v\" VALUE {'title': ?, 'year': ?, 'info': ?}",
    runner.TableName)),
  Parameters: params,
 })
 if err != nil {
  log.Printf("Couldn't insert an item with PartiQL. Here's why: %v\n", err)
 }
 return err
}



// GetMovie runs a PartiQL SELECT statement to get a movie from the DynamoDB table
 by
// title and year.
func (runner PartiQLRunner) GetMovie(ctx context.Context, title string, year int)
 (Movie, error) {
 var movie Movie
 params, err := attributevalue.MarshalList([]interface{}{title, year})
 if err != nil {
  panic(err)
 }
 response, err := runner.DynamoDbClient.ExecuteStatement(ctx,
 &dynamodb.ExecuteStatementInput{
  Statement: aws.String(
   fmt.Sprintf("SELECT * FROM \"%v\" WHERE title=? AND year=?",
    runner.TableName)),
  Parameters: params,
 })
 if err != nil {
  log.Printf("Couldn't get info about %v. Here's why: %v\n", title, err)
 } else {
  err = attributevalue.UnmarshalMap(response.Items[0], &movie)
  if err != nil {
   log.Printf("Couldn't unmarshal response. Here's why: %v\n", err)
  }
 }
 return movie, err
}



// UpdateMovie runs a PartiQL UPDATE statement to update the rating of a movie that
```

```
// already exists in the DynamoDB table.
func (runner PartiQLRunner) UpdateMovie(ctx context.Context, movie Movie, rating
 float64) error {
 params, err := attributevalue.MarshalList([]interface{}{rating, movie.Title,
 movie.Year})
 if err != nil {
  panic(err)
 }
 _, err = runner.DynamoDbClient.ExecuteStatement(ctx,
 &dynamodb.ExecuteStatementInput{
  Statement: aws.String(
   fmt.Sprintf("UPDATE \"%v\" SET info.rating=? WHERE title=? AND year=?",
    runner.TableName)),
  Parameters: params,
 })
 if err != nil {
  log.Printf("Couldn't update movie %v. Here's why: %v\n", movie.Title, err)
 }
 return err
}



// DeleteMovie runs a PartiQL DELETE statement to remove a movie from the DynamoDB
 table.
func (runner PartiQLRunner) DeleteMovie(ctx context.Context, movie Movie) error {
 params, err := attributevalue.MarshalList([]interface{}{movie.Title, movie.Year})
 if err != nil {
  panic(err)
 }
 _, err = runner.DynamoDbClient.ExecuteStatement(ctx,
 &dynamodb.ExecuteStatementInput{
  Statement: aws.String(
   fmt.Sprintf("DELETE FROM \"%v\" WHERE title=? AND year=?",
    runner.TableName)),
  Parameters: params,
 })
 if err != nil {
  log.Printf("Couldn't delete %v from the table. Here's why: %v\n", movie.Title,
 err)
 }
 return err
}
```

- Einzelheiten zur API finden Sie ExecuteStatementunter AWS SDK für Go API-Referenz.

Serverless-Beispiele

Aufrufen einer Lambda-Funktion über einen DynamoDB-Auslöser

Das folgende Codebeispiel zeigt, wie eine Lambda-Funktion implementiert wird, die ein Ereignis empfängt, das durch den Empfang von Datensätzen aus einem DynamoDB-Stream ausgelöst wird. Die Funktion ruft die DynamoDB-Nutzdaten ab und protokolliert den Inhalt des Datensatzes.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu. GitHub Das vollständige Beispiel sowie eine Anleitung zum Einrichten und Ausführen finden Sie im Repository mit Serverless-Beispielen.

Nutzen eines DynamoDB-Ereignisses mit Lambda unter Verwendung von Go.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
package main

import (
 "context"
 "github.com/aws/aws-lambda-go/lambda"
 "github.com/aws/aws-lambda-go/events"
 "fmt"
)

func HandleRequest(ctx context.Context, event events.DynamoDBEvent) (*string, error)
 {
 if len(event.Records) == 0 {
  return nil, fmt.Errorf("received empty event")
 }

 for _, record := range event.Records {
   LogDynamoDBRecord(record)
```

```
 }

 message := fmt.Sprintf("Records processed: %d", len(event.Records))
 return &message, nil
}

func main() {
 lambda.Start(HandleRequest)
}

func LogDynamoDBRecord(record events.DynamoDBEventRecord){
 fmt.Println(record.EventID)
 fmt.Println(record.EventName)
 fmt.Printf("%+v\n", record.Change)
}
```

Melden von Batch-Elementfehlern für Lambda-Funktionen mit einem DynamoDB-Auslöser

Das folgende Codebeispiel zeigt, wie eine partielle Batch-Antwort für Lambda-Funktionen implementiert wird, die Ereignisse aus einem DynamoDB-Stream empfangen. Die Funktion meldet die Batch-Elementfehler in der Antwort und signalisiert Lambda, diese Nachrichten später erneut zu versuchen.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu. GitHub Das vollständige Beispiel sowie eine Anleitung zum Einrichten und Ausführen finden Sie im Repository mit Serverless-Beispielen.

Melden von DynamoDB-Batchelementfehlern mit Lambda unter Verwendung von Go.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
package main

import (
 "context"
 "github.com/aws/aws-lambda-go/events"
 "github.com/aws/aws-lambda-go/lambda"
```

```
)

type BatchItemFailure struct {
 ItemIdentifier string `json:"ItemIdentifier"`
}

type BatchResult struct {
 BatchItemFailures []BatchItemFailure `json:"BatchItemFailures"`
}

func HandleRequest(ctx context.Context, event events.DynamoDBEvent) (*BatchResult,
 error) {
 var batchItemFailures []BatchItemFailure
 curRecordSequenceNumber := ""

 for _, record := range event.Records {
  // Process your record
  curRecordSequenceNumber = record.Change.SequenceNumber
 }

 if curRecordSequenceNumber != "" {
  batchItemFailures = append(batchItemFailures, BatchItemFailure{ItemIdentifier:
 curRecordSequenceNumber})
 }

 batchResult := BatchResult{
  BatchItemFailures: batchItemFailures,
 }

 return &batchResult, nil
}

func main() {
 lambda.Start(HandleRequest)
}
```

AWS Beiträge der Gemeinschaft

Erstellen und testen Sie eine serverlose Anwendung

Das folgende Codebeispiel zeigt, wie eine serverlose Anwendung mithilfe von API Gateway mit Lambda und DynamoDB erstellt und getestet wird.

SDK für Go V2

Zeigt, wie eine serverlose Anwendung, die aus einem API Gateway mit Lambda und DynamoDB besteht, mithilfe des Go SDK erstellt und getestet wird.

Den vollständigen Quellcode und Anweisungen zur Einrichtung und Ausführung finden Sie im vollständigen Beispiel unter. GitHub

In diesem Beispiel verwendete Dienste

- API Gateway
- DynamoDB
- Lambda

IAM-Beispiele mit SDK for Go V2

Die folgenden Codebeispiele zeigen Ihnen, wie Sie mithilfe von AWS SDK für Go V2 mit IAM Aktionen ausführen und allgemeine Szenarien implementieren.

Bei Grundlagen handelt es sich um Code-Beispiele, die Ihnen zeigen, wie Sie die wesentlichen Vorgänge innerhalb eines Services ausführen.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Service-Funktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarios anzeigen.

Jedes Beispiel enthält einen Link zum vollständigen Quellcode, in dem Sie Anweisungen zum Einrichten und Ausführen des Codes im Kontext finden.

Erste Schritte

Hallo IAM

Die folgenden Codebeispiele veranschaulichen, wie Sie mit der Verwendung von IAM beginnen.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
package main

import (
 "context"
 "fmt"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/config"
 "github.com/aws/aws-sdk-go-v2/service/iam"
)

// main uses the AWS SDK for Go (v2) to create an AWS Identity and Access Management
 (IAM)
// client and list up to 10 policies in your account.
// This example uses the default settings specified in your shared credentials
// and config files.
func main() {
 ctx := context.Background()
 sdkConfig, err := config.LoadDefaultConfig(ctx)
 if err != nil {
  fmt.Println("Couldn't load default configuration. Have you set up your AWS
 account?")
  fmt.Println(err)
  return
 }
 iamClient := iam.NewFromConfig(sdkConfig)
 const maxPols = 10
 fmt.Printf("Let's list up to %v policies for your account.\n", maxPols)
 result, err := iamClient.ListPolicies(ctx, &iam.ListPoliciesInput{
  MaxItems: aws.Int32(maxPols),
 })
 if err != nil {
  fmt.Printf("Couldn't list policies for your account. Here's why: %v\n", err)
```

```
  return
 }
 if len(result.Policies) == 0 {
  fmt.Println("You don't have any policies!")
 } else {
  for _, policy := range result.Policies {
   fmt.Printf("\t%v\n", *policy.PolicyName)
  }
 }
}
```

- Einzelheiten zur API finden Sie ListPoliciesin der AWS SDK für Go API-Referenz.

Themen

- Grundlagen
- Aktionen

Grundlagen

Erlernen der Grundlagen

Das folgende Codebeispiel veranschaulicht, wie Sie einen Benutzer erstellen und eine Rolle annehmen lassen.

> **Warning**
>
> Um Sicherheitsrisiken zu vermeiden, sollten Sie IAM-Benutzer nicht zur Authentifizierung verwenden, wenn Sie speziell entwickelte Software entwickeln oder mit echten Daten arbeiten. Verwenden Sie stattdessen den Verbund mit einem Identitätsanbieter wie AWS IAM Identity Center.

- Erstellen Sie einen Benutzer ohne Berechtigungen.
- Erstellen einer Rolle, die die Berechtigung zum Auflisten von Amazon-S3-Buckets für das Konto erteilt.
- Hinzufügen einer Richtlinie, damit der Benutzer die Rolle übernehmen kann.

- Übernehmen Sie die Rolle und listen Sie S3-Buckets mit temporären Anmeldeinformationen auf, und bereinigen Sie dann die Ressourcen.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Führen Sie ein interaktives Szenario an einer Eingabeaufforderung aus.

```
import (
 "context"
 "errors"
 "fmt"
 "log"
 "math/rand"
 "strings"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/config"
 "github.com/aws/aws-sdk-go-v2/credentials"
 "github.com/aws/aws-sdk-go-v2/service/iam"
 "github.com/aws/aws-sdk-go-v2/service/iam/types"
 "github.com/aws/aws-sdk-go-v2/service/s3"
 "github.com/aws/aws-sdk-go-v2/service/sts"
 "github.com/aws/smithy-go"
 "github.com/awsdocs/aws-doc-sdk-examples/gov2/demotools"
 "github.com/awsdocs/aws-doc-sdk-examples/gov2/iam/actions"
)

// AssumeRoleScenario shows you how to use the AWS Identity and Access Management
 (IAM)
// service to perform the following actions:
//
//  1. Create a user who has no permissions.
//  2. Create a role that grants permission to list Amazon Simple Storage Service
//     (Amazon S3) buckets for the account.
```

```
//  3. Add a policy to let the user assume the role.
//  4. Try and fail to list buckets without permissions.
//  5. Assume the role and list S3 buckets using temporary credentials.
//  6. Delete the policy, role, and user.
type AssumeRoleScenario struct {
 sdkConfig      aws.Config
 accountWrapper actions.AccountWrapper
 policyWrapper  actions.PolicyWrapper
 roleWrapper    actions.RoleWrapper
 userWrapper    actions.UserWrapper
 questioner     demotools.IQuestioner
 helper         IScenarioHelper
 isTestRun      bool
}

// NewAssumeRoleScenario constructs an AssumeRoleScenario instance from a
 configuration.
// It uses the specified config to get an IAM client and create wrappers for the
 actions
// used in the scenario.
func NewAssumeRoleScenario(sdkConfig aws.Config, questioner demotools.IQuestioner,
 helper IScenarioHelper) AssumeRoleScenario {
 iamClient := iam.NewFromConfig(sdkConfig)
 return AssumeRoleScenario{
  sdkConfig:      sdkConfig,
  accountWrapper: actions.AccountWrapper{IamClient: iamClient},
  policyWrapper:  actions.PolicyWrapper{IamClient: iamClient},
  roleWrapper:    actions.RoleWrapper{IamClient: iamClient},
  userWrapper:    actions.UserWrapper{IamClient: iamClient},
  questioner:     questioner,
  helper:         helper,
 }
}

// addTestOptions appends the API options specified in the original configuration to
// another configuration. This is used to attach the middleware stubber to clients
// that are constructed during the scenario, which is needed for unit testing.
func (scenario AssumeRoleScenario) addTestOptions(scenarioConfig *aws.Config) {
 if scenario.isTestRun {
  scenarioConfig.APIOptions = append(scenarioConfig.APIOptions,
 scenario.sdkConfig.APIOptions...)
 }
}
```

```
// Run runs the interactive scenario.
func (scenario AssumeRoleScenario) Run(ctx context.Context) {
 defer func() {
  if r := recover(); r != nil {
   log.Printf("Something went wrong with the demo.\n")
   log.Println(r)
  }
 }()

 log.Println(strings.Repeat("-", 88))
 log.Println("Welcome to the AWS Identity and Access Management (IAM) assume role
 demo.")
 log.Println(strings.Repeat("-", 88))

 user := scenario.CreateUser(ctx)
 accessKey := scenario.CreateAccessKey(ctx, user)
 role := scenario.CreateRoleAndPolicies(ctx, user)
 noPermsConfig := scenario.ListBucketsWithoutPermissions(ctx, accessKey)
 scenario.ListBucketsWithAssumedRole(ctx, noPermsConfig, role)
 scenario.Cleanup(ctx, user, role)

 log.Println(strings.Repeat("-", 88))
 log.Println("Thanks for watching!")
 log.Println(strings.Repeat("-", 88))
}

// CreateUser creates a new IAM user. This user has no permissions.
func (scenario AssumeRoleScenario) CreateUser(ctx context.Context) *types.User {
 log.Println("Let's create an example user with no permissions.")
 userName := scenario.questioner.Ask("Enter a name for the example user:",
 demotools.NotEmpty{})
 user, err := scenario.userWrapper.GetUser(ctx, userName)
 if err != nil {
  panic(err)
 }
 if user == nil {
  user, err = scenario.userWrapper.CreateUser(ctx, userName)
  if err != nil {
   panic(err)
  }
  log.Printf("Created user %v.\n", *user.UserName)
 } else {
  log.Printf("User %v already exists.\n", *user.UserName)
 }
```

```
 log.Println(strings.Repeat("-", 88))
 return user
}

// CreateAccessKey creates an access key for the user.
func (scenario AssumeRoleScenario) CreateAccessKey(ctx context.Context, user
 *types.User) *types.AccessKey {
 accessKey, err := scenario.userWrapper.CreateAccessKeyPair(ctx, *user.UserName)
 if err != nil {
  panic(err)
 }
 log.Printf("Created access key %v for your user.", *accessKey.AccessKeyId)
 log.Println("Waiting a few seconds for your user to be ready...")
 scenario.helper.Pause(10)
 log.Println(strings.Repeat("-", 88))
 return accessKey
}

// CreateRoleAndPolicies creates a policy that grants permission to list S3 buckets
 for
// the current account and attaches the policy to a newly created role. It also adds
 an
// inline policy to the specified user that grants the user permission to assume the
 role.
func (scenario AssumeRoleScenario) CreateRoleAndPolicies(ctx context.Context, user
 *types.User) *types.Role {
 log.Println("Let's create a role and policy that grant permission to list S3
 buckets.")
 scenario.questioner.Ask("Press Enter when you're ready.")
 listBucketsRole, err := scenario.roleWrapper.CreateRole(ctx,
 scenario.helper.GetName(), *user.Arn)
 if err != nil {
  panic(err)
 }
 log.Printf("Created role %v.\n", *listBucketsRole.RoleName)
 listBucketsPolicy, err := scenario.policyWrapper.CreatePolicy(
  ctx, scenario.helper.GetName(), []string{"s3:ListAllMyBuckets"}, "arn:aws:s3:::*")
 if err != nil {
  panic(err)
 }
 log.Printf("Created policy %v.\n", *listBucketsPolicy.PolicyName)
 err = scenario.roleWrapper.AttachRolePolicy(ctx, *listBucketsPolicy.Arn,
 *listBucketsRole.RoleName)
 if err != nil {
```

```
  panic(err)
 }
 log.Printf("Attached policy %v to role %v.\n", *listBucketsPolicy.PolicyName,
  *listBucketsRole.RoleName)
 err = scenario.userWrapper.CreateUserPolicy(ctx, *user.UserName,
 scenario.helper.GetName(),
  []string{"sts:AssumeRole"}, *listBucketsRole.Arn)
 if err != nil {
  panic(err)
 }
 log.Printf("Created an inline policy for user %v that lets the user assume the
 role.\n",
  *user.UserName)
 log.Println("Let's give AWS a few seconds to propagate these new resources and
 connections...")
 scenario.helper.Pause(10)
 log.Println(strings.Repeat("-", 88))
 return listBucketsRole
}

// ListBucketsWithoutPermissions creates an Amazon S3 client from the user's access
 key
// credentials and tries to list buckets for the account. Because the user does not
 have
// permission to perform this action, the action fails.
func (scenario AssumeRoleScenario) ListBucketsWithoutPermissions(ctx
 context.Context, accessKey *types.AccessKey) *aws.Config {
 log.Println("Let's try to list buckets without permissions. This should return an
 AccessDenied error.")
 scenario.questioner.Ask("Press Enter when you're ready.")
 noPermsConfig, err := config.LoadDefaultConfig(ctx,
  config.WithCredentialsProvider(credentials.NewStaticCredentialsProvider(
   *accessKey.AccessKeyId, *accessKey.SecretAccessKey, ""),
  ))
 if err != nil {
  panic(err)
 }

 // Add test options if this is a test run. This is needed only for testing
 purposes.
 scenario.addTestOptions(&noPermsConfig)

 s3Client := s3.NewFromConfig(noPermsConfig)
 _, err = s3Client.ListBuckets(ctx, &s3.ListBucketsInput{})
```

```
 if err != nil {
  // The SDK for Go does not model the AccessDenied error, so check ErrorCode
 directly.
  var ae smithy.APIError
  if errors.As(err, &ae) {
   switch ae.ErrorCode() {
   case "AccessDenied":
    log.Println("Got AccessDenied error, which is the expected result because\n" +
     "the ListBuckets call was made without permissions.")
   default:
    log.Println("Expected AccessDenied, got something else.")
    panic(err)
   }
  }
 } else {
  log.Println("Expected AccessDenied error when calling ListBuckets without
 permissions,\n" +
   "but the call succeeded. Continuing the example anyway...")
 }
 log.Println(strings.Repeat("-", 88))
 return &noPermsConfig
}

// ListBucketsWithAssumedRole performs the following actions:
//
//  1. Creates an AWS Security Token Service (AWS STS) client from the config
 created from
//     the user's access key credentials.
//  2. Gets temporary credentials by assuming the role that grants permission to
 list the
//     buckets.
//  3. Creates an Amazon S3 client from the temporary credentials.
//  4. Lists buckets for the account. Because the temporary credentials are
 generated by
//     assuming the role that grants permission, the action succeeds.
func (scenario AssumeRoleScenario) ListBucketsWithAssumedRole(ctx context.Context,
 noPermsConfig *aws.Config, role *types.Role) {
 log.Println("Let's assume the role that grants permission to list buckets and try
 again.")
 scenario.questioner.Ask("Press Enter when you're ready.")
 stsClient := sts.NewFromConfig(*noPermsConfig)
 tempCredentials, err := stsClient.AssumeRole(ctx, &sts.AssumeRoleInput{
  RoleArn:         role.Arn,
  RoleSessionName: aws.String("AssumeRoleExampleSession"),
```

```
  DurationSeconds: aws.Int32(900),
 })
 if err != nil {
  log.Printf("Couldn't assume role %v.\n", *role.RoleName)
  panic(err)
 }
 log.Printf("Assumed role %v, got temporary credentials.\n", *role.RoleName)
 assumeRoleConfig, err := config.LoadDefaultConfig(ctx,
  config.WithCredentialsProvider(credentials.NewStaticCredentialsProvider(
   *tempCredentials.Credentials.AccessKeyId,
   *tempCredentials.Credentials.SecretAccessKey,
   *tempCredentials.Credentials.SessionToken),
  ),
 )
 if err != nil {
  panic(err)
 }

 // Add test options if this is a test run. This is needed only for testing
 purposes.
 scenario.addTestOptions(&assumeRoleConfig)

 s3Client := s3.NewFromConfig(assumeRoleConfig)
 result, err := s3Client.ListBuckets(ctx, &s3.ListBucketsInput{})
 if err != nil {
  log.Println("Couldn't list buckets with assumed role credentials.")
  panic(err)
 }
 log.Println("Successfully called ListBuckets with assumed role credentials, \n" +
  "here are some of them:")
 for i := 0; i < len(result.Buckets) && i < 5; i++ {
  log.Printf("\t%v\n", *result.Buckets[i].Name)
 }
 log.Println(strings.Repeat("-", 88))
}

// Cleanup deletes all resources created for the scenario.
func (scenario AssumeRoleScenario) Cleanup(ctx context.Context, user *types.User,
 role *types.Role) {
 if scenario.questioner.AskBool(
  "Do you want to delete the resources created for this example? (y/n)", "y",
 ) {
  policies, err := scenario.roleWrapper.ListAttachedRolePolicies(ctx,
 *role.RoleName)
```

```
 if err != nil {
  panic(err)
 }
 for _, policy := range policies {
  err = scenario.roleWrapper.DetachRolePolicy(ctx, *role.RoleName,
*policy.PolicyArn)
  if err != nil {
   panic(err)
  }
  err = scenario.policyWrapper.DeletePolicy(ctx, *policy.PolicyArn)
  if err != nil {
   panic(err)
  }
  log.Printf("Detached policy %v from role %v and deleted the policy.\n",
   *policy.PolicyName, *role.RoleName)
 }
 err = scenario.roleWrapper.DeleteRole(ctx, *role.RoleName)
 if err != nil {
  panic(err)
 }
 log.Printf("Deleted role %v.\n", *role.RoleName)

 userPols, err := scenario.userWrapper.ListUserPolicies(ctx, *user.UserName)
 if err != nil {
  panic(err)
 }
 for _, userPol := range userPols {
  err = scenario.userWrapper.DeleteUserPolicy(ctx, *user.UserName, userPol)
  if err != nil {
   panic(err)
  }
  log.Printf("Deleted policy %v from user %v.\n", userPol, *user.UserName)
 }
 keys, err := scenario.userWrapper.ListAccessKeys(ctx, *user.UserName)
 if err != nil {
  panic(err)
 }
 for _, key := range keys {
  err = scenario.userWrapper.DeleteAccessKey(ctx, *user.UserName, *key.AccessKeyId)
  if err != nil {
   panic(err)
  }
  log.Printf("Deleted access key %v from user %v.\n", *key.AccessKeyId,
*user.UserName)
```

```
  }
  err = scenario.userWrapper.DeleteUser(ctx, *user.UserName)
  if err != nil {
   panic(err)
  }
  log.Printf("Deleted user %v.\n", *user.UserName)
  log.Println(strings.Repeat("-", 88))
 }

}

// IScenarioHelper abstracts input and wait functions from a scenario so that they
// can be mocked for unit testing.
type IScenarioHelper interface {
 GetName() string
 Pause(secs int)
}

const rMax = 100000

type ScenarioHelper struct {
 Prefix string
 Random *rand.Rand
}

// GetName returns a unique name formed of a prefix and a random number.
func (helper *ScenarioHelper) GetName() string {
 return fmt.Sprintf("%v%v", helper.Prefix, helper.Random.Intn(rMax))
}

// Pause waits for the specified number of seconds.
func (helper ScenarioHelper) Pause(secs int) {
 time.Sleep(time.Duration(secs) * time.Second)
}
```

Definieren Sie eine Struktur, die Kontoaktionen umschließt.

```
import (
 "context"
 "log"
```

```
 "github.com/aws/aws-sdk-go-v2/service/iam"
 "github.com/aws/aws-sdk-go-v2/service/iam/types"
)

// AccountWrapper encapsulates AWS Identity and Access Management (IAM) account
 actions
// used in the examples.
// It contains an IAM service client that is used to perform account actions.
type AccountWrapper struct {
 IamClient *iam.Client
}



// GetAccountPasswordPolicy gets the account password policy for the current
 account.
// If no policy has been set, a NoSuchEntityException is error is returned.
func (wrapper AccountWrapper) GetAccountPasswordPolicy(ctx context.Context)
 (*types.PasswordPolicy, error) {
 var pwPolicy *types.PasswordPolicy
 result, err := wrapper.IamClient.GetAccountPasswordPolicy(ctx,
  &iam.GetAccountPasswordPolicyInput{})
 if err != nil {
  log.Printf("Couldn't get account password policy. Here's why: %v\n", err)
 } else {
  pwPolicy = result.PasswordPolicy
 }
 return pwPolicy, err
}



// ListSAMLProviders gets the SAML providers for the account.
func (wrapper AccountWrapper) ListSAMLProviders(ctx context.Context)
 ([]types.SAMLProviderListEntry, error) {
 var providers []types.SAMLProviderListEntry
 result, err := wrapper.IamClient.ListSAMLProviders(ctx,
 &iam.ListSAMLProvidersInput{})
 if err != nil {
  log.Printf("Couldn't list SAML providers. Here's why: %v\n", err)
 } else {
  providers = result.SAMLProviderList
 }
```

```
 return providers, err
}
```

Definieren Sie eine Struktur, die Richtlinienaktionen umschließt.

```
import (
 "context"
 "encoding/json"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/iam"
 "github.com/aws/aws-sdk-go-v2/service/iam/types"
)

// PolicyWrapper encapsulates AWS Identity and Access Management (IAM) policy
 actions
// used in the examples.
// It contains an IAM service client that is used to perform policy actions.
type PolicyWrapper struct {
 IamClient *iam.Client
}



// ListPolicies gets up to maxPolicies policies.
func (wrapper PolicyWrapper) ListPolicies(ctx context.Context, maxPolicies int32)
 ([]types.Policy, error) {
 var policies []types.Policy
 result, err := wrapper.IamClient.ListPolicies(ctx, &iam.ListPoliciesInput{
  MaxItems: aws.Int32(maxPolicies),
 })
 if err != nil {
  log.Printf("Couldn't list policies. Here's why: %v\n", err)
 } else {
  policies = result.Policies
 }
 return policies, err
}
```

```
// PolicyDocument defines a policy document as a Go struct that can be serialized
// to JSON.
type PolicyDocument struct {
 Version   string
 Statement []PolicyStatement
}

// PolicyStatement defines a statement in a policy document.
type PolicyStatement struct {
 Effect    string
 Action    []string
 Principal map[string]string `json:",omitempty"`
 Resource  *string           `json:",omitempty"`
}

// CreatePolicy creates a policy that grants a list of actions to the specified
 resource.
// PolicyDocument shows how to work with a policy document as a data structure and
// serialize it to JSON by using Go's JSON marshaler.
func (wrapper PolicyWrapper) CreatePolicy(ctx context.Context, policyName string,
 actions []string,
 resourceArn string) (*types.Policy, error) {
 var policy *types.Policy
 policyDoc := PolicyDocument{
  Version: "2012-10-17",
  Statement: []PolicyStatement{{
   Effect:   "Allow",
   Action:   actions,
   Resource: aws.String(resourceArn),
  }},
 }
 policyBytes, err := json.Marshal(policyDoc)
 if err != nil {
  log.Printf("Couldn't create policy document for %v. Here's why: %v\n",
 resourceArn, err)
  return nil, err
 }
 result, err := wrapper.IamClient.CreatePolicy(ctx, &iam.CreatePolicyInput{
  PolicyDocument: aws.String(string(policyBytes)),
  PolicyName:     aws.String(policyName),
 })
 if err != nil {
```

```
  log.Printf("Couldn't create policy %v. Here's why: %v\n", policyName, err)
 } else {
  policy = result.Policy
 }
 return policy, err
}



// GetPolicy gets data about a policy.
func (wrapper PolicyWrapper) GetPolicy(ctx context.Context, policyArn string)
 (*types.Policy, error) {
 var policy *types.Policy
 result, err := wrapper.IamClient.GetPolicy(ctx, &iam.GetPolicyInput{
  PolicyArn: aws.String(policyArn),
 })
 if err != nil {
  log.Printf("Couldn't get policy %v. Here's why: %v\n", policyArn, err)
 } else {
  policy = result.Policy
 }
 return policy, err
}



// DeletePolicy deletes a policy.
func (wrapper PolicyWrapper) DeletePolicy(ctx context.Context, policyArn string)
 error {
 _, err := wrapper.IamClient.DeletePolicy(ctx, &iam.DeletePolicyInput{
  PolicyArn: aws.String(policyArn),
 })
 if err != nil {
  log.Printf("Couldn't delete policy %v. Here's why: %v\n", policyArn, err)
 }
 return err
}
```

Definieren Sie eine Struktur, die Rollenaktionen umschließt.

```
import (
 "context"
 "encoding/json"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/iam"
 "github.com/aws/aws-sdk-go-v2/service/iam/types"
)

// RoleWrapper encapsulates AWS Identity and Access Management (IAM) role actions
// used in the examples.
// It contains an IAM service client that is used to perform role actions.
type RoleWrapper struct {
 IamClient *iam.Client
}



// ListRoles gets up to maxRoles roles.
func (wrapper RoleWrapper) ListRoles(ctx context.Context, maxRoles int32)
 ([]types.Role, error) {
 var roles []types.Role
 result, err := wrapper.IamClient.ListRoles(ctx,
  &iam.ListRolesInput{MaxItems: aws.Int32(maxRoles)},
 )
 if err != nil {
  log.Printf("Couldn't list roles. Here's why: %v\n", err)
 } else {
  roles = result.Roles
 }
 return roles, err
}



// CreateRole creates a role that trusts a specified user. The trusted user can
 assume
// the role to acquire its permissions.
// PolicyDocument shows how to work with a policy document as a data structure and
// serialize it to JSON by using Go's JSON marshaler.
func (wrapper RoleWrapper) CreateRole(ctx context.Context, roleName string,
 trustedUserArn string) (*types.Role, error) {
 var role *types.Role
```

```
 trustPolicy := PolicyDocument{
  Version: "2012-10-17",
  Statement: []PolicyStatement{{
   Effect:    "Allow",
   Principal: map[string]string{"AWS": trustedUserArn},
   Action:    []string{"sts:AssumeRole"},
  }},
 }
 policyBytes, err := json.Marshal(trustPolicy)
 if err != nil {
  log.Printf("Couldn't create trust policy for %v. Here's why: %v\n",
 trustedUserArn, err)
  return nil, err
 }
 result, err := wrapper.IamClient.CreateRole(ctx, &iam.CreateRoleInput{
  AssumeRolePolicyDocument: aws.String(string(policyBytes)),
  RoleName:                 aws.String(roleName),
 })
 if err != nil {
  log.Printf("Couldn't create role %v. Here's why: %v\n", roleName, err)
 } else {
  role = result.Role
 }
 return role, err
}



// GetRole gets data about a role.
func (wrapper RoleWrapper) GetRole(ctx context.Context, roleName string)
 (*types.Role, error) {
 var role *types.Role
 result, err := wrapper.IamClient.GetRole(ctx,
  &iam.GetRoleInput{RoleName: aws.String(roleName)})
 if err != nil {
  log.Printf("Couldn't get role %v. Here's why: %v\n", roleName, err)
 } else {
  role = result.Role
 }
 return role, err
}
```

```
// CreateServiceLinkedRole creates a service-linked role that is owned by the
 specified service.
func (wrapper RoleWrapper) CreateServiceLinkedRole(ctx context.Context, serviceName
 string, description string) (
 *types.Role, error) {
 var role *types.Role
 result, err := wrapper.IamClient.CreateServiceLinkedRole(ctx,
 &iam.CreateServiceLinkedRoleInput{
  AWSServiceName: aws.String(serviceName),
  Description:    aws.String(description),
 })
 if err != nil {
  log.Printf("Couldn't create service-linked role %v. Here's why: %v\n",
 serviceName, err)
 } else {
  role = result.Role
 }
 return role, err
}



// DeleteServiceLinkedRole deletes a service-linked role.
func (wrapper RoleWrapper) DeleteServiceLinkedRole(ctx context.Context, roleName
 string) error {
 _, err := wrapper.IamClient.DeleteServiceLinkedRole(ctx,
 &iam.DeleteServiceLinkedRoleInput{
  RoleName: aws.String(roleName)},
 )
 if err != nil {
  log.Printf("Couldn't delete service-linked role %v. Here's why: %v\n", roleName,
 err)
 }
 return err
}



// AttachRolePolicy attaches a policy to a role.
func (wrapper RoleWrapper) AttachRolePolicy(ctx context.Context, policyArn string,
 roleName string) error {
 _, err := wrapper.IamClient.AttachRolePolicy(ctx, &iam.AttachRolePolicyInput{
  PolicyArn: aws.String(policyArn),
  RoleName:  aws.String(roleName),
```

```
 })
 if err != nil {
  log.Printf("Couldn't attach policy %v to role %v. Here's why: %v\n", policyArn,
 roleName, err)
 }
 return err
}



// ListAttachedRolePolicies lists the policies that are attached to the specified
 role.
func (wrapper RoleWrapper) ListAttachedRolePolicies(ctx context.Context, roleName
 string) ([]types.AttachedPolicy, error) {
 var policies []types.AttachedPolicy
 result, err := wrapper.IamClient.ListAttachedRolePolicies(ctx,
 &iam.ListAttachedRolePoliciesInput{
  RoleName: aws.String(roleName),
 })
 if err != nil {
  log.Printf("Couldn't list attached policies for role %v. Here's why: %v\n",
 roleName, err)
 } else {
  policies = result.AttachedPolicies
 }
 return policies, err
}



// DetachRolePolicy detaches a policy from a role.
func (wrapper RoleWrapper) DetachRolePolicy(ctx context.Context, roleName string,
 policyArn string) error {
 _, err := wrapper.IamClient.DetachRolePolicy(ctx, &iam.DetachRolePolicyInput{
  PolicyArn: aws.String(policyArn),
  RoleName:  aws.String(roleName),
 })
 if err != nil {
  log.Printf("Couldn't detach policy from role %v. Here's why: %v\n", roleName, err)
 }
 return err
}
```

```
// ListRolePolicies lists the inline policies for a role.
func (wrapper RoleWrapper) ListRolePolicies(ctx context.Context, roleName string)
 ([]string, error) {
 var policies []string
 result, err := wrapper.IamClient.ListRolePolicies(ctx, &iam.ListRolePoliciesInput{
  RoleName: aws.String(roleName),
 })
 if err != nil {
  log.Printf("Couldn't list policies for role %v. Here's why: %v\n", roleName, err)
 } else {
  policies = result.PolicyNames
 }
 return policies, err
}



// DeleteRole deletes a role. All attached policies must be detached before a
// role can be deleted.
func (wrapper RoleWrapper) DeleteRole(ctx context.Context, roleName string) error {
 _, err := wrapper.IamClient.DeleteRole(ctx, &iam.DeleteRoleInput{
  RoleName: aws.String(roleName),
 })
 if err != nil {
  log.Printf("Couldn't delete role %v. Here's why: %v\n", roleName, err)
 }
 return err
}
```

Definieren Sie eine Struktur, die Benutzeraktionen umschließt.

```
import (
 "context"
 "encoding/json"
 "errors"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
```

```
 "github.com/aws/aws-sdk-go-v2/service/iam"
 "github.com/aws/aws-sdk-go-v2/service/iam/types"
 "github.com/aws/smithy-go"
)

// UserWrapper encapsulates user actions used in the examples.
// It contains an IAM service client that is used to perform user actions.
type UserWrapper struct {
 IamClient *iam.Client
}



// ListUsers gets up to maxUsers number of users.
func (wrapper UserWrapper) ListUsers(ctx context.Context, maxUsers int32)
 ([]types.User, error) {
 var users []types.User
 result, err := wrapper.IamClient.ListUsers(ctx, &iam.ListUsersInput{
  MaxItems: aws.Int32(maxUsers),
 })
 if err != nil {
  log.Printf("Couldn't list users. Here's why: %v\n", err)
 } else {
  users = result.Users
 }
 return users, err
}



// GetUser gets data about a user.
func (wrapper UserWrapper) GetUser(ctx context.Context, userName string)
 (*types.User, error) {
 var user *types.User
 result, err := wrapper.IamClient.GetUser(ctx, &iam.GetUserInput{
  UserName: aws.String(userName),
 })
 if err != nil {
  var apiError smithy.APIError
  if errors.As(err, &apiError) {
   switch apiError.(type) {
   case *types.NoSuchEntityException:
    log.Printf("User %v does not exist.\n", userName)
    err = nil
```

```
   default:
    log.Printf("Couldn't get user %v. Here's why: %v\n", userName, err)
   }
  }
 } else {
  user = result.User
 }
 return user, err
}



// CreateUser creates a new user with the specified name.
func (wrapper UserWrapper) CreateUser(ctx context.Context, userName string)
 (*types.User, error) {
 var user *types.User
 result, err := wrapper.IamClient.CreateUser(ctx, &iam.CreateUserInput{
  UserName: aws.String(userName),
 })
 if err != nil {
  log.Printf("Couldn't create user %v. Here's why: %v\n", userName, err)
 } else {
  user = result.User
 }
 return user, err
}



// CreateUserPolicy adds an inline policy to a user. This example creates a policy
 that
// grants a list of actions on a specified role.
// PolicyDocument shows how to work with a policy document as a data structure and
// serialize it to JSON by using Go's JSON marshaler.
func (wrapper UserWrapper) CreateUserPolicy(ctx context.Context, userName string,
 policyName string, actions []string,
 roleArn string) error {
 policyDoc := PolicyDocument{
  Version: "2012-10-17",
  Statement: []PolicyStatement{{
   Effect:   "Allow",
   Action:   actions,
   Resource: aws.String(roleArn),
  }},
```

```
 }
 policyBytes, err := json.Marshal(policyDoc)
 if err != nil {
  log.Printf("Couldn't create policy document for %v. Here's why: %v\n", roleArn,
 err)
  return err
 }
 _, err = wrapper.IamClient.PutUserPolicy(ctx, &iam.PutUserPolicyInput{
  PolicyDocument: aws.String(string(policyBytes)),
  PolicyName:     aws.String(policyName),
  UserName:       aws.String(userName),
 })
 if err != nil {
  log.Printf("Couldn't create policy for user %v. Here's why: %v\n", userName, err)
 }
 return err
}



// ListUserPolicies lists the inline policies for the specified user.
func (wrapper UserWrapper) ListUserPolicies(ctx context.Context, userName string)
 ([]string, error) {
 var policies []string
 result, err := wrapper.IamClient.ListUserPolicies(ctx, &iam.ListUserPoliciesInput{
  UserName: aws.String(userName),
 })
 if err != nil {
  log.Printf("Couldn't list policies for user %v. Here's why: %v\n", userName, err)
 } else {
  policies = result.PolicyNames
 }
 return policies, err
}



// DeleteUserPolicy deletes an inline policy from a user.
func (wrapper UserWrapper) DeleteUserPolicy(ctx context.Context, userName string,
 policyName string) error {
 _, err := wrapper.IamClient.DeleteUserPolicy(ctx, &iam.DeleteUserPolicyInput{
  PolicyName: aws.String(policyName),
  UserName:   aws.String(userName),
 })
```

```
 if err != nil {
  log.Printf("Couldn't delete policy from user %v. Here's why: %v\n", userName, err)
 }
 return err
}



// DeleteUser deletes a user.
func (wrapper UserWrapper) DeleteUser(ctx context.Context, userName string) error {
 _, err := wrapper.IamClient.DeleteUser(ctx, &iam.DeleteUserInput{
  UserName: aws.String(userName),
 })
 if err != nil {
  log.Printf("Couldn't delete user %v. Here's why: %v\n", userName, err)
 }
 return err
}



// CreateAccessKeyPair creates an access key for a user. The returned access key
 contains
// the ID and secret credentials needed to use the key.
func (wrapper UserWrapper) CreateAccessKeyPair(ctx context.Context, userName string)
 (*types.AccessKey, error) {
 var key *types.AccessKey
 result, err := wrapper.IamClient.CreateAccessKey(ctx, &iam.CreateAccessKeyInput{
  UserName: aws.String(userName)})
 if err != nil {
  log.Printf("Couldn't create access key pair for user %v. Here's why: %v\n",
 userName, err)
 } else {
  key = result.AccessKey
 }
 return key, err
}



// DeleteAccessKey deletes an access key from a user.
func (wrapper UserWrapper) DeleteAccessKey(ctx context.Context, userName string,
 keyId string) error {
 _, err := wrapper.IamClient.DeleteAccessKey(ctx, &iam.DeleteAccessKeyInput{
```

```
  AccessKeyId: aws.String(keyId),
  UserName:    aws.String(userName),
 })
 if err != nil {
  log.Printf("Couldn't delete access key %v. Here's why: %v\n", keyId, err)
 }
 return err
}



// ListAccessKeys lists the access keys for the specified user.
func (wrapper UserWrapper) ListAccessKeys(ctx context.Context, userName string)
 ([]types.AccessKeyMetadata, error) {
 var keys []types.AccessKeyMetadata
 result, err := wrapper.IamClient.ListAccessKeys(ctx, &iam.ListAccessKeysInput{
  UserName: aws.String(userName),
 })
 if err != nil {
  log.Printf("Couldn't list access keys for user %v. Here's why: %v\n", userName,
 err)
 } else {
  keys = result.AccessKeyMetadata
 }
 return keys, err
}
```

- API-Details finden Sie in den folgenden Themen der AWS SDK für Go -API-Referenz.
  - AttachRolePolicy
  - CreateAccessKey
  - CreatePolicy
  - CreateRole
  - CreateUser
  - DeleteAccessKey
  - DeletePolicy
  - DeleteRole
  - DeleteUser

- DeleteUserPolicy
- DetachRolePolicy
- PutUserPolicy

Aktionen

`AttachRolePolicy`

Das folgende Codebeispiel zeigt, wie man es benutzt`AttachRolePolicy`.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import (
 "context"
 "encoding/json"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/iam"
 "github.com/aws/aws-sdk-go-v2/service/iam/types"
)

// RoleWrapper encapsulates AWS Identity and Access Management (IAM) role actions
// used in the examples.
// It contains an IAM service client that is used to perform role actions.
type RoleWrapper struct {
 IamClient *iam.Client
}



// AttachRolePolicy attaches a policy to a role.
```

```
func (wrapper RoleWrapper) AttachRolePolicy(ctx context.Context, policyArn string,
 roleName string) error {
 _, err := wrapper.IamClient.AttachRolePolicy(ctx, &iam.AttachRolePolicyInput{
  PolicyArn: aws.String(policyArn),
  RoleName:  aws.String(roleName),
 })
 if err != nil {
  log.Printf("Couldn't attach policy %v to role %v. Here's why: %v\n", policyArn,
 roleName, err)
 }
 return err
}
```

- Einzelheiten zur API finden Sie AttachRolePolicyin der AWS SDK für Go API-Referenz.

`CreateAccessKey`

Das folgende Codebeispiel zeigt die Verwendung`CreateAccessKey`.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import (
 "context"
 "encoding/json"
 "errors"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/iam"
 "github.com/aws/aws-sdk-go-v2/service/iam/types"
 "github.com/aws/smithy-go"
)
```

```
// UserWrapper encapsulates user actions used in the examples.
// It contains an IAM service client that is used to perform user actions.
type UserWrapper struct {
 IamClient *iam.Client
}



// CreateAccessKeyPair creates an access key for a user. The returned access key
 contains
// the ID and secret credentials needed to use the key.
func (wrapper UserWrapper) CreateAccessKeyPair(ctx context.Context, userName string)
 (*types.AccessKey, error) {
 var key *types.AccessKey
 result, err := wrapper.IamClient.CreateAccessKey(ctx, &iam.CreateAccessKeyInput{
  UserName: aws.String(userName)})
 if err != nil {
  log.Printf("Couldn't create access key pair for user %v. Here's why: %v\n",
 userName, err)
 } else {
  key = result.AccessKey
 }
 return key, err
}
```

- Einzelheiten zur API finden Sie CreateAccessKeyin der AWS SDK für Go API-Referenz.

`CreatePolicy`

Das folgende Codebeispiel zeigt die Verwendung`CreatePolicy`.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import (
 "context"
 "encoding/json"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/iam"
 "github.com/aws/aws-sdk-go-v2/service/iam/types"
)

// PolicyWrapper encapsulates AWS Identity and Access Management (IAM) policy
 actions
// used in the examples.
// It contains an IAM service client that is used to perform policy actions.
type PolicyWrapper struct {
 IamClient *iam.Client
}



// PolicyDocument defines a policy document as a Go struct that can be serialized
// to JSON.
type PolicyDocument struct {
 Version   string
 Statement []PolicyStatement
}

// PolicyStatement defines a statement in a policy document.
type PolicyStatement struct {
 Effect    string
 Action    []string
 Principal map[string]string `json:",omitempty"`
 Resource  *string           `json:",omitempty"`
}

// CreatePolicy creates a policy that grants a list of actions to the specified
 resource.
// PolicyDocument shows how to work with a policy document as a data structure and
// serialize it to JSON by using Go's JSON marshaler.
func (wrapper PolicyWrapper) CreatePolicy(ctx context.Context, policyName string,
 actions []string,
 resourceArn string) (*types.Policy, error) {
 var policy *types.Policy
 policyDoc := PolicyDocument{
```

```
  Version: "2012-10-17",
  Statement: []PolicyStatement{{
   Effect:   "Allow",
   Action:   actions,
   Resource: aws.String(resourceArn),
  }},
 }
 policyBytes, err := json.Marshal(policyDoc)
 if err != nil {
  log.Printf("Couldn't create policy document for %v. Here's why: %v\n",
 resourceArn, err)
  return nil, err
 }
 result, err := wrapper.IamClient.CreatePolicy(ctx, &iam.CreatePolicyInput{
  PolicyDocument: aws.String(string(policyBytes)),
  PolicyName:     aws.String(policyName),
 })
 if err != nil {
  log.Printf("Couldn't create policy %v. Here's why: %v\n", policyName, err)
 } else {
  policy = result.Policy
 }
 return policy, err
}
```

- Einzelheiten zur API finden Sie CreatePolicyin der AWS SDK für Go API-Referenz.

`CreateRole`

Das folgende Codebeispiel zeigt die Verwendung`CreateRole`.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import (
 "context"
 "encoding/json"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/iam"
 "github.com/aws/aws-sdk-go-v2/service/iam/types"
)

// RoleWrapper encapsulates AWS Identity and Access Management (IAM) role actions
// used in the examples.
// It contains an IAM service client that is used to perform role actions.
type RoleWrapper struct {
 IamClient *iam.Client
}



// CreateRole creates a role that trusts a specified user. The trusted user can
 assume
// the role to acquire its permissions.
// PolicyDocument shows how to work with a policy document as a data structure and
// serialize it to JSON by using Go's JSON marshaler.
func (wrapper RoleWrapper) CreateRole(ctx context.Context, roleName string,
 trustedUserArn string) (*types.Role, error) {
 var role *types.Role
 trustPolicy := PolicyDocument{
  Version: "2012-10-17",
  Statement: []PolicyStatement{{
   Effect:    "Allow",
   Principal: map[string]string{"AWS": trustedUserArn},
   Action:    []string{"sts:AssumeRole"},
  }},
 }
 policyBytes, err := json.Marshal(trustPolicy)
 if err != nil {
  log.Printf("Couldn't create trust policy for %v. Here's why: %v\n",
 trustedUserArn, err)
  return nil, err
 }
 result, err := wrapper.IamClient.CreateRole(ctx, &iam.CreateRoleInput{
  AssumeRolePolicyDocument: aws.String(string(policyBytes)),
  RoleName:                 aws.String(roleName),
```

```
 })
 if err != nil {
  log.Printf("Couldn't create role %v. Here's why: %v\n", roleName, err)
 } else {
  role = result.Role
 }
 return role, err
}
```

- Einzelheiten zur API finden Sie CreateRolein der AWS SDK für Go API-Referenz.

`CreateServiceLinkedRole`

Das folgende Codebeispiel zeigt die Verwendung`CreateServiceLinkedRole`.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import (
 "context"
 "encoding/json"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/iam"
 "github.com/aws/aws-sdk-go-v2/service/iam/types"
)

// RoleWrapper encapsulates AWS Identity and Access Management (IAM) role actions
// used in the examples.
// It contains an IAM service client that is used to perform role actions.
type RoleWrapper struct {
 IamClient *iam.Client
```

```
}



// CreateServiceLinkedRole creates a service-linked role that is owned by the
 specified service.
func (wrapper RoleWrapper) CreateServiceLinkedRole(ctx context.Context, serviceName
 string, description string) (
 *types.Role, error) {
 var role *types.Role
 result, err := wrapper.IamClient.CreateServiceLinkedRole(ctx,
 &iam.CreateServiceLinkedRoleInput{
  AWSServiceName: aws.String(serviceName),
  Description:    aws.String(description),
 })
 if err != nil {
  log.Printf("Couldn't create service-linked role %v. Here's why: %v\n",
 serviceName, err)
 } else {
  role = result.Role
 }
 return role, err
}
```

- Einzelheiten zur API finden Sie CreateServiceLinkedRolein der AWS SDK für Go API-Referenz.

`CreateUser`

Das folgende Codebeispiel zeigt die Verwendung`CreateUser`.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import (
 "context"
 "encoding/json"
 "errors"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/iam"
 "github.com/aws/aws-sdk-go-v2/service/iam/types"
 "github.com/aws/smithy-go"
)

// UserWrapper encapsulates user actions used in the examples.
// It contains an IAM service client that is used to perform user actions.
type UserWrapper struct {
 IamClient *iam.Client
}



// CreateUser creates a new user with the specified name.
func (wrapper UserWrapper) CreateUser(ctx context.Context, userName string)
 (*types.User, error) {
 var user *types.User
 result, err := wrapper.IamClient.CreateUser(ctx, &iam.CreateUserInput{
  UserName: aws.String(userName),
 })
 if err != nil {
  log.Printf("Couldn't create user %v. Here's why: %v\n", userName, err)
 } else {
  user = result.User
 }
 return user, err
}
```

- Einzelheiten zur API finden Sie CreateUserin der AWS SDK für Go API-Referenz.

`DeleteAccessKey`

Das folgende Codebeispiel zeigt die Verwendung`DeleteAccessKey`.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import (
 "context"
 "encoding/json"
 "errors"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/iam"
 "github.com/aws/aws-sdk-go-v2/service/iam/types"
 "github.com/aws/smithy-go"
)

// UserWrapper encapsulates user actions used in the examples.
// It contains an IAM service client that is used to perform user actions.
type UserWrapper struct {
 IamClient *iam.Client
}



// DeleteAccessKey deletes an access key from a user.
func (wrapper UserWrapper) DeleteAccessKey(ctx context.Context, userName string,
 keyId string) error {
 _, err := wrapper.IamClient.DeleteAccessKey(ctx, &iam.DeleteAccessKeyInput{
  AccessKeyId: aws.String(keyId),
  UserName:    aws.String(userName),
 })
 if err != nil {
  log.Printf("Couldn't delete access key %v. Here's why: %v\n", keyId, err)
 }
 return err
}
```

- Einzelheiten zur API finden Sie DeleteAccessKeyin der AWS SDK für Go API-Referenz.

`DeletePolicy`

Das folgende Codebeispiel zeigt die Verwendung`DeletePolicy`.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import (
 "context"
 "encoding/json"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/iam"
 "github.com/aws/aws-sdk-go-v2/service/iam/types"
)

// PolicyWrapper encapsulates AWS Identity and Access Management (IAM) policy
 actions
// used in the examples.
// It contains an IAM service client that is used to perform policy actions.
type PolicyWrapper struct {
 IamClient *iam.Client
}



// DeletePolicy deletes a policy.
func (wrapper PolicyWrapper) DeletePolicy(ctx context.Context, policyArn string)
 error {
 _, err := wrapper.IamClient.DeletePolicy(ctx, &iam.DeletePolicyInput{
  PolicyArn: aws.String(policyArn),
```

```
 })
 if err != nil {
  log.Printf("Couldn't delete policy %v. Here's why: %v\n", policyArn, err)
 }
 return err
}
```

- Einzelheiten zur API finden Sie DeletePolicyin der AWS SDK für Go API-Referenz.

`DeleteRole`

Das folgende Codebeispiel zeigt die Verwendung`DeleteRole`.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import (
 "context"
 "encoding/json"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/iam"
 "github.com/aws/aws-sdk-go-v2/service/iam/types"
)

// RoleWrapper encapsulates AWS Identity and Access Management (IAM) role actions
// used in the examples.
// It contains an IAM service client that is used to perform role actions.
type RoleWrapper struct {
 IamClient *iam.Client
}
```

```
// DeleteRole deletes a role. All attached policies must be detached before a
// role can be deleted.
func (wrapper RoleWrapper) DeleteRole(ctx context.Context, roleName string) error {
 _, err := wrapper.IamClient.DeleteRole(ctx, &iam.DeleteRoleInput{
  RoleName: aws.String(roleName),
 })
 if err != nil {
  log.Printf("Couldn't delete role %v. Here's why: %v\n", roleName, err)
 }
 return err
}
```

- Einzelheiten zur API finden Sie DeleteRolein der AWS SDK für Go API-Referenz.

`DeleteServiceLinkedRole`

Das folgende Codebeispiel zeigt die Verwendung`DeleteServiceLinkedRole`.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import (
 "context"
 "encoding/json"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/iam"
 "github.com/aws/aws-sdk-go-v2/service/iam/types"
)

// RoleWrapper encapsulates AWS Identity and Access Management (IAM) role actions
// used in the examples.
```

```
// It contains an IAM service client that is used to perform role actions.
type RoleWrapper struct {
 IamClient *iam.Client
}



// DeleteServiceLinkedRole deletes a service-linked role.
func (wrapper RoleWrapper) DeleteServiceLinkedRole(ctx context.Context, roleName
 string) error {
 _, err := wrapper.IamClient.DeleteServiceLinkedRole(ctx,
 &iam.DeleteServiceLinkedRoleInput{
  RoleName: aws.String(roleName)},
 )
 if err != nil {
  log.Printf("Couldn't delete service-linked role %v. Here's why: %v\n", roleName,
 err)
 }
 return err
}
```

- Einzelheiten zur API finden Sie DeleteServiceLinkedRolein der AWS SDK für Go API-Referenz.

`DeleteUser`

Das folgende Codebeispiel zeigt die Verwendung`DeleteUser`.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import (
 "context"
 "encoding/json"
 "errors"
```

```
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/iam"
 "github.com/aws/aws-sdk-go-v2/service/iam/types"
 "github.com/aws/smithy-go"
)

// UserWrapper encapsulates user actions used in the examples.
// It contains an IAM service client that is used to perform user actions.
type UserWrapper struct {
 IamClient *iam.Client
}



// DeleteUser deletes a user.
func (wrapper UserWrapper) DeleteUser(ctx context.Context, userName string) error {
 _, err := wrapper.IamClient.DeleteUser(ctx, &iam.DeleteUserInput{
  UserName: aws.String(userName),
 })
 if err != nil {
  log.Printf("Couldn't delete user %v. Here's why: %v\n", userName, err)
 }
 return err
}
```

- Einzelheiten zur API finden Sie DeleteUserin der AWS SDK für Go API-Referenz.

`DeleteUserPolicy`

Das folgende Codebeispiel zeigt die Verwendung`DeleteUserPolicy`.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import (
 "context"
 "encoding/json"
 "errors"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/iam"
 "github.com/aws/aws-sdk-go-v2/service/iam/types"
 "github.com/aws/smithy-go"
)

// UserWrapper encapsulates user actions used in the examples.
// It contains an IAM service client that is used to perform user actions.
type UserWrapper struct {
 IamClient *iam.Client
}



// DeleteUserPolicy deletes an inline policy from a user.
func (wrapper UserWrapper) DeleteUserPolicy(ctx context.Context, userName string,
 policyName string) error {
 _, err := wrapper.IamClient.DeleteUserPolicy(ctx, &iam.DeleteUserPolicyInput{
  PolicyName: aws.String(policyName),
  UserName:   aws.String(userName),
 })
 if err != nil {
  log.Printf("Couldn't delete policy from user %v. Here's why: %v\n", userName, err)
 }
 return err
}
```

- Einzelheiten zur API finden Sie DeleteUserPolicyin der AWS SDK für Go API-Referenz.

`DetachRolePolicy`

Das folgende Codebeispiel zeigt die Verwendung`DetachRolePolicy`.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import (
 "context"
 "encoding/json"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/iam"
 "github.com/aws/aws-sdk-go-v2/service/iam/types"
)

// RoleWrapper encapsulates AWS Identity and Access Management (IAM) role actions
// used in the examples.
// It contains an IAM service client that is used to perform role actions.
type RoleWrapper struct {
 IamClient *iam.Client
}



// DetachRolePolicy detaches a policy from a role.
func (wrapper RoleWrapper) DetachRolePolicy(ctx context.Context, roleName string,
 policyArn string) error {
 _, err := wrapper.IamClient.DetachRolePolicy(ctx, &iam.DetachRolePolicyInput{
  PolicyArn: aws.String(policyArn),
  RoleName:  aws.String(roleName),
 })
 if err != nil {
  log.Printf("Couldn't detach policy from role %v. Here's why: %v\n", roleName, err)
 }
 return err
}
```

- Einzelheiten zur API finden Sie DetachRolePolicyin der AWS SDK für Go API-Referenz.

`GetAccountPasswordPolicy`

Das folgende Codebeispiel zeigt die Verwendung`GetAccountPasswordPolicy`.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import (
 "context"
 "log"

 "github.com/aws/aws-sdk-go-v2/service/iam"
 "github.com/aws/aws-sdk-go-v2/service/iam/types"
)

// AccountWrapper encapsulates AWS Identity and Access Management (IAM) account
 actions
// used in the examples.
// It contains an IAM service client that is used to perform account actions.
type AccountWrapper struct {
 IamClient *iam.Client
}



// GetAccountPasswordPolicy gets the account password policy for the current
 account.
// If no policy has been set, a NoSuchEntityException is error is returned.
func (wrapper AccountWrapper) GetAccountPasswordPolicy(ctx context.Context)
 (*types.PasswordPolicy, error) {
 var pwPolicy *types.PasswordPolicy
 result, err := wrapper.IamClient.GetAccountPasswordPolicy(ctx,
  &iam.GetAccountPasswordPolicyInput{})
 if err != nil {
```

```
  log.Printf("Couldn't get account password policy. Here's why: %v\n", err)
 } else {
  pwPolicy = result.PasswordPolicy
 }
 return pwPolicy, err
}
```

- Einzelheiten zur API finden Sie GetAccountPasswordPolicyin der AWS SDK für Go API-Referenz.

`GetPolicy`

Das folgende Codebeispiel zeigt die Verwendung`GetPolicy`.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import (
 "context"
 "encoding/json"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/iam"
 "github.com/aws/aws-sdk-go-v2/service/iam/types"
)

// PolicyWrapper encapsulates AWS Identity and Access Management (IAM) policy
 actions
// used in the examples.
// It contains an IAM service client that is used to perform policy actions.
type PolicyWrapper struct {
 IamClient *iam.Client
```

```
}



// GetPolicy gets data about a policy.
func (wrapper PolicyWrapper) GetPolicy(ctx context.Context, policyArn string)
 (*types.Policy, error) {
 var policy *types.Policy
 result, err := wrapper.IamClient.GetPolicy(ctx, &iam.GetPolicyInput{
  PolicyArn: aws.String(policyArn),
 })
 if err != nil {
  log.Printf("Couldn't get policy %v. Here's why: %v\n", policyArn, err)
 } else {
  policy = result.Policy
 }
 return policy, err
}
```

- Einzelheiten zur API finden Sie GetPolicyin der AWS SDK für Go API-Referenz.

`GetRole`

Das folgende Codebeispiel zeigt die Verwendung`GetRole`.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import (
 "context"
 "encoding/json"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
```

```
 "github.com/aws/aws-sdk-go-v2/service/iam"
 "github.com/aws/aws-sdk-go-v2/service/iam/types"
)

// RoleWrapper encapsulates AWS Identity and Access Management (IAM) role actions
// used in the examples.
// It contains an IAM service client that is used to perform role actions.
type RoleWrapper struct {
 IamClient *iam.Client
}



// GetRole gets data about a role.
func (wrapper RoleWrapper) GetRole(ctx context.Context, roleName string)
 (*types.Role, error) {
 var role *types.Role
 result, err := wrapper.IamClient.GetRole(ctx,
  &iam.GetRoleInput{RoleName: aws.String(roleName)})
 if err != nil {
  log.Printf("Couldn't get role %v. Here's why: %v\n", roleName, err)
 } else {
  role = result.Role
 }
 return role, err
}
```

- Einzelheiten zur API finden Sie GetRolein der AWS SDK für Go API-Referenz.

`GetUser`

Das folgende Codebeispiel zeigt die Verwendung`GetUser`.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import (
 "context"
 "encoding/json"
 "errors"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/iam"
 "github.com/aws/aws-sdk-go-v2/service/iam/types"
 "github.com/aws/smithy-go"
)

// UserWrapper encapsulates user actions used in the examples.
// It contains an IAM service client that is used to perform user actions.
type UserWrapper struct {
 IamClient *iam.Client
}



// GetUser gets data about a user.
func (wrapper UserWrapper) GetUser(ctx context.Context, userName string)
 (*types.User, error) {
 var user *types.User
 result, err := wrapper.IamClient.GetUser(ctx, &iam.GetUserInput{
  UserName: aws.String(userName),
 })
 if err != nil {
  var apiError smithy.APIError
  if errors.As(err, &apiError) {
   switch apiError.(type) {
   case *types.NoSuchEntityException:
    log.Printf("User %v does not exist.\n", userName)
    err = nil
   default:
    log.Printf("Couldn't get user %v. Here's why: %v\n", userName, err)
   }
  }
 } else {
  user = result.User
 }
 return user, err
```

```
}
```

- Einzelheiten zur API finden Sie GetUserin der AWS SDK für Go API-Referenz.

`ListAccessKeys`

Das folgende Codebeispiel zeigt die Verwendung`ListAccessKeys`.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import (
 "context"
 "encoding/json"
 "errors"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/iam"
 "github.com/aws/aws-sdk-go-v2/service/iam/types"
 "github.com/aws/smithy-go"
)

// UserWrapper encapsulates user actions used in the examples.
// It contains an IAM service client that is used to perform user actions.
type UserWrapper struct {
 IamClient *iam.Client
}



// ListAccessKeys lists the access keys for the specified user.
func (wrapper UserWrapper) ListAccessKeys(ctx context.Context, userName string)
 ([]types.AccessKeyMetadata, error) {
```

```
 var keys []types.AccessKeyMetadata
 result, err := wrapper.IamClient.ListAccessKeys(ctx, &iam.ListAccessKeysInput{
  UserName: aws.String(userName),
 })
 if err != nil {
  log.Printf("Couldn't list access keys for user %v. Here's why: %v\n", userName,
 err)
 } else {
  keys = result.AccessKeyMetadata
 }
 return keys, err
}
```

- Einzelheiten zur API finden Sie ListAccessKeysin der AWS SDK für Go API-Referenz.

`ListAttachedRolePolicies`

Das folgende Codebeispiel zeigt die Verwendung`ListAttachedRolePolicies`.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import (
 "context"
 "encoding/json"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/iam"
 "github.com/aws/aws-sdk-go-v2/service/iam/types"
)

// RoleWrapper encapsulates AWS Identity and Access Management (IAM) role actions
// used in the examples.
```

```
// It contains an IAM service client that is used to perform role actions.
type RoleWrapper struct {
 IamClient *iam.Client
}



// ListAttachedRolePolicies lists the policies that are attached to the specified
 role.
func (wrapper RoleWrapper) ListAttachedRolePolicies(ctx context.Context, roleName
 string) ([]types.AttachedPolicy, error) {
 var policies []types.AttachedPolicy
 result, err := wrapper.IamClient.ListAttachedRolePolicies(ctx,
 &iam.ListAttachedRolePoliciesInput{
  RoleName: aws.String(roleName),
 })
 if err != nil {
  log.Printf("Couldn't list attached policies for role %v. Here's why: %v\n",
 roleName, err)
 } else {
  policies = result.AttachedPolicies
 }
 return policies, err
}
```

- Einzelheiten zur API finden Sie ListAttachedRolePoliciesin der AWS SDK für Go API-Referenz.

`ListGroups`

Das folgende Codebeispiel zeigt die Verwendung`ListGroups`.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import (
 "context"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/iam"
 "github.com/aws/aws-sdk-go-v2/service/iam/types"
)

// GroupWrapper encapsulates AWS Identity and Access Management (IAM) group actions
// used in the examples.
// It contains an IAM service client that is used to perform group actions.
type GroupWrapper struct {
 IamClient *iam.Client
}



// ListGroups lists up to maxGroups number of groups.
func (wrapper GroupWrapper) ListGroups(ctx context.Context, maxGroups int32)
 ([]types.Group, error) {
 var groups []types.Group
 result, err := wrapper.IamClient.ListGroups(ctx, &iam.ListGroupsInput{
  MaxItems: aws.Int32(maxGroups),
 })
 if err != nil {
  log.Printf("Couldn't list groups. Here's why: %v\n", err)
 } else {
  groups = result.Groups
 }
 return groups, err
}
```

- Einzelheiten zur API finden Sie ListGroupsin der AWS SDK für Go API-Referenz.

`ListPolicies`

Das folgende Codebeispiel zeigt die Verwendung`ListPolicies`.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import (
 "context"
 "encoding/json"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/iam"
 "github.com/aws/aws-sdk-go-v2/service/iam/types"
)

// PolicyWrapper encapsulates AWS Identity and Access Management (IAM) policy
 actions
// used in the examples.
// It contains an IAM service client that is used to perform policy actions.
type PolicyWrapper struct {
 IamClient *iam.Client
}



// ListPolicies gets up to maxPolicies policies.
func (wrapper PolicyWrapper) ListPolicies(ctx context.Context, maxPolicies int32)
 ([]types.Policy, error) {
 var policies []types.Policy
 result, err := wrapper.IamClient.ListPolicies(ctx, &iam.ListPoliciesInput{
  MaxItems: aws.Int32(maxPolicies),
 })
 if err != nil {
  log.Printf("Couldn't list policies. Here's why: %v\n", err)
 } else {
  policies = result.Policies
 }
 return policies, err
```

```
}
```

- Einzelheiten zur API finden Sie ListPoliciesin der AWS SDK für Go API-Referenz.

`ListRolePolicies`

Das folgende Codebeispiel zeigt die Verwendung`ListRolePolicies`.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import (
 "context"
 "encoding/json"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/iam"
 "github.com/aws/aws-sdk-go-v2/service/iam/types"
)

// RoleWrapper encapsulates AWS Identity and Access Management (IAM) role actions
// used in the examples.
// It contains an IAM service client that is used to perform role actions.
type RoleWrapper struct {
 IamClient *iam.Client
}



// ListRolePolicies lists the inline policies for a role.
func (wrapper RoleWrapper) ListRolePolicies(ctx context.Context, roleName string)
 ([]string, error) {
```

```
 var policies []string
 result, err := wrapper.IamClient.ListRolePolicies(ctx, &iam.ListRolePoliciesInput{
  RoleName: aws.String(roleName),
 })
 if err != nil {
  log.Printf("Couldn't list policies for role %v. Here's why: %v\n", roleName, err)
 } else {
  policies = result.PolicyNames
 }
 return policies, err
}
```

- Einzelheiten zur API finden Sie ListRolePoliciesin der AWS SDK für Go API-Referenz.

`ListRoles`

Das folgende Codebeispiel zeigt die Verwendung`ListRoles`.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import (
 "context"
 "encoding/json"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/iam"
 "github.com/aws/aws-sdk-go-v2/service/iam/types"
)

// RoleWrapper encapsulates AWS Identity and Access Management (IAM) role actions
// used in the examples.
```

```
// It contains an IAM service client that is used to perform role actions.
type RoleWrapper struct {
 IamClient *iam.Client
}



// ListRoles gets up to maxRoles roles.
func (wrapper RoleWrapper) ListRoles(ctx context.Context, maxRoles int32)
 ([]types.Role, error) {
 var roles []types.Role
 result, err := wrapper.IamClient.ListRoles(ctx,
  &iam.ListRolesInput{MaxItems: aws.Int32(maxRoles)},
 )
 if err != nil {
  log.Printf("Couldn't list roles. Here's why: %v\n", err)
 } else {
  roles = result.Roles
 }
 return roles, err
}
```

- Einzelheiten zur API finden Sie ListRolesin der AWS SDK für Go API-Referenz.

`ListSAMLProviders`

Das folgende Codebeispiel zeigt die Verwendung`ListSAMLProviders`.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import (
 "context"
```

```
 "log"

 "github.com/aws/aws-sdk-go-v2/service/iam"
 "github.com/aws/aws-sdk-go-v2/service/iam/types"
)

// AccountWrapper encapsulates AWS Identity and Access Management (IAM) account
 actions
// used in the examples.
// It contains an IAM service client that is used to perform account actions.
type AccountWrapper struct {
 IamClient *iam.Client
}



// ListSAMLProviders gets the SAML providers for the account.
func (wrapper AccountWrapper) ListSAMLProviders(ctx context.Context)
 ([]types.SAMLProviderListEntry, error) {
 var providers []types.SAMLProviderListEntry
 result, err := wrapper.IamClient.ListSAMLProviders(ctx,
 &iam.ListSAMLProvidersInput{})
 if err != nil {
  log.Printf("Couldn't list SAML providers. Here's why: %v\n", err)
 } else {
  providers = result.SAMLProviderList
 }
 return providers, err
}
```

- Einzelheiten zur API finden Sie unter Liste SAMLProviders in der AWS SDK für Go API-Referenz.

`ListUserPolicies`

Das folgende Codebeispiel zeigt die Verwendung`ListUserPolicies`.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import (
 "context"
 "encoding/json"
 "errors"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/iam"
 "github.com/aws/aws-sdk-go-v2/service/iam/types"
 "github.com/aws/smithy-go"
)

// UserWrapper encapsulates user actions used in the examples.
// It contains an IAM service client that is used to perform user actions.
type UserWrapper struct {
 IamClient *iam.Client
}



// ListUserPolicies lists the inline policies for the specified user.
func (wrapper UserWrapper) ListUserPolicies(ctx context.Context, userName string)
 ([]string, error) {
 var policies []string
 result, err := wrapper.IamClient.ListUserPolicies(ctx, &iam.ListUserPoliciesInput{
  UserName: aws.String(userName),
 })
 if err != nil {
  log.Printf("Couldn't list policies for user %v. Here's why: %v\n", userName, err)
 } else {
  policies = result.PolicyNames
 }
 return policies, err
```

```
}
```

- Einzelheiten zur API finden Sie ListUserPoliciesin der AWS SDK für Go API-Referenz.

`ListUsers`

Das folgende Codebeispiel zeigt die Verwendung`ListUsers`.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import (
 "context"
 "encoding/json"
 "errors"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/iam"
 "github.com/aws/aws-sdk-go-v2/service/iam/types"
 "github.com/aws/smithy-go"
)

// UserWrapper encapsulates user actions used in the examples.
// It contains an IAM service client that is used to perform user actions.
type UserWrapper struct {
 IamClient *iam.Client
}



// ListUsers gets up to maxUsers number of users.
func (wrapper UserWrapper) ListUsers(ctx context.Context, maxUsers int32)
 ([]types.User, error) {
```

```
 var users []types.User
 result, err := wrapper.IamClient.ListUsers(ctx, &iam.ListUsersInput{
  MaxItems: aws.Int32(maxUsers),
 })
 if err != nil {
  log.Printf("Couldn't list users. Here's why: %v\n", err)
 } else {
  users = result.Users
 }
 return users, err
}
```

- Einzelheiten zur API finden Sie ListUsersin der AWS SDK für Go API-Referenz.

`PutUserPolicy`

Das folgende Codebeispiel zeigt die Verwendung`PutUserPolicy`.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import (
 "context"
 "encoding/json"
 "errors"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/iam"
 "github.com/aws/aws-sdk-go-v2/service/iam/types"
 "github.com/aws/smithy-go"
)
```

```
// UserWrapper encapsulates user actions used in the examples.
// It contains an IAM service client that is used to perform user actions.
type UserWrapper struct {
 IamClient *iam.Client
}



// CreateUserPolicy adds an inline policy to a user. This example creates a policy
 that
// grants a list of actions on a specified role.
// PolicyDocument shows how to work with a policy document as a data structure and
// serialize it to JSON by using Go's JSON marshaler.
func (wrapper UserWrapper) CreateUserPolicy(ctx context.Context, userName string,
 policyName string, actions []string,
 roleArn string) error {
 policyDoc := PolicyDocument{
  Version: "2012-10-17",
  Statement: []PolicyStatement{{
   Effect:   "Allow",
   Action:   actions,
   Resource: aws.String(roleArn),
  }},
 }
 policyBytes, err := json.Marshal(policyDoc)
 if err != nil {
  log.Printf("Couldn't create policy document for %v. Here's why: %v\n", roleArn,
 err)
  return err
 }
 _, err = wrapper.IamClient.PutUserPolicy(ctx, &iam.PutUserPolicyInput{
  PolicyDocument: aws.String(string(policyBytes)),
  PolicyName:     aws.String(policyName),
  UserName:       aws.String(userName),
 })
 if err != nil {
  log.Printf("Couldn't create policy for user %v. Here's why: %v\n", userName, err)
 }
 return err
}
```

- Einzelheiten zur API finden Sie PutUserPolicyin der AWS SDK für Go API-Referenz.

Kinesis-Beispiele mit SDK for Go V2

Die folgenden Codebeispiele zeigen Ihnen, wie Sie mithilfe von AWS SDK für Go V2 mit Kinesis Aktionen ausführen und allgemeine Szenarien implementieren.

Jedes Beispiel enthält einen Link zum vollständigen Quellcode, in dem Sie Anweisungen zum Einrichten und Ausführen des Codes im Kontext finden.

Themen

- Serverless-Beispiele

Serverless-Beispiele

Aufrufen einer Lambda-Funktion über einen Kinesis-Auslöser

Das folgende Codebeispiel zeigt, wie eine Lambda-Funktion implementiert wird, die ein Ereignis empfängt, das durch den Empfang von Datensätzen aus einem Kinesis-Stream ausgelöst wird. Die Funktion ruft die Kinesis-Nutzlast ab, dekodiert von Base64 und protokolliert den Datensatzinhalt.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu. GitHub Das vollständige Beispiel sowie eine Anleitung zum Einrichten und Ausführen finden Sie im Repository mit Serverless-Beispielen.

Nutzen eines Kinesis-Ereignisses mit Lambda unter Verwendung von Go.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
package main

import (
 "context"
 "log"

 "github.com/aws/aws-lambda-go/events"
 "github.com/aws/aws-lambda-go/lambda"
)
```

```
func handler(ctx context.Context, kinesisEvent events.KinesisEvent) error {
 if len(kinesisEvent.Records) == 0 {
  log.Printf("empty Kinesis event received")
  return nil
 }

 for _, record := range kinesisEvent.Records {
  log.Printf("processed Kinesis event with EventId: %v", record.EventID)
  recordDataBytes := record.Kinesis.Data
  recordDataText := string(recordDataBytes)
  log.Printf("record data: %v", recordDataText)
  // TODO: Do interesting work based on the new data
 }
 log.Printf("successfully processed %v records", len(kinesisEvent.Records))
 return nil
}

func main() {
 lambda.Start(handler)
}
```

Melden von Batch-Elementfehlern für Lambda-Funktionen mit einem Kinesis-Auslöser

Das folgende Codebeispiel zeigt, wie eine partielle Batch-Antwort für Lambda-Funktionen implementiert wird, die Ereignisse aus einem Kinesis-Stream empfangen. Die Funktion meldet die Batch-Elementfehler in der Antwort und signalisiert Lambda, diese Nachrichten später erneut zu versuchen.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu. GitHub Das vollständige Beispiel sowie eine Anleitung zum Einrichten und Ausführen finden Sie im Repository mit Serverless-Beispielen.

Melden von Fehlern bei Kinesis-Batchelementen mit Lambda unter Verwendung von Go.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
```

```
// SPDX-License-Identifier: Apache-2.0
package main

import (
 "context"
 "fmt"
 "github.com/aws/aws-lambda-go/events"
 "github.com/aws/aws-lambda-go/lambda"
)

func handler(ctx context.Context, kinesisEvent events.KinesisEvent)
 (map[string]interface{}, error) {
 batchItemFailures := []map[string]interface{}{}

 for _, record := range kinesisEvent.Records {
  curRecordSequenceNumber := ""

  // Process your record
  if /* Your record processing condition here */ {
   curRecordSequenceNumber = record.Kinesis.SequenceNumber
  }

  // Add a condition to check if the record processing failed
  if curRecordSequenceNumber != "" {
   batchItemFailures = append(batchItemFailures, map[string]interface{}
{"itemIdentifier": curRecordSequenceNumber})
  }
 }

 kinesisBatchResponse := map[string]interface{}{
  "batchItemFailures": batchItemFailures,
 }
 return kinesisBatchResponse, nil
}

func main() {
 lambda.Start(handler)
}
```

Lambda-Beispiele mit SDK for Go V2

Die folgenden Codebeispiele zeigen Ihnen, wie Sie Aktionen ausführen und allgemeine Szenarien implementieren, indem Sie AWS SDK für Go V2 mit Lambda verwenden.

Bei Grundlagen handelt es sich um Code-Beispiele, die Ihnen zeigen, wie Sie die wesentlichen Vorgänge innerhalb eines Services ausführen.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Service-Funktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarios anzeigen.

Szenarien sind Code-Beispiele, die Ihnen zeigen, wie Sie bestimmte Aufgaben ausführen, indem Sie mehrere Funktionen innerhalb eines Services aufrufen oder mit anderen AWS-Services kombinieren.

AWS Community-Beiträge sind Beispiele, die von mehreren AWS Teams erstellt wurden und verwaltet werden. Um Feedback zu geben, verwende den Mechanismus, der in den verlinkten Repositorien zur Verfügung steht.

Jedes Beispiel enthält einen Link zum vollständigen Quellcode, in dem Sie Anweisungen zum Einrichten und Ausführen des Codes im Kontext finden.

Erste Schritte

Hallo Lambda

Die folgenden Codebeispiele veranschaulichen, wie Sie mit der Verwendung von Lambda beginnen.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
package main

import (
 "context"
```

```
 "fmt"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/config"
 "github.com/aws/aws-sdk-go-v2/service/lambda"
)

// main uses the AWS SDK for Go (v2) to create an AWS Lambda client and list up to
 10
// functions in your account.
// This example uses the default settings specified in your shared credentials
// and config files.
func main() {
 ctx := context.Background()
 sdkConfig, err := config.LoadDefaultConfig(ctx)
 if err != nil {
  fmt.Println("Couldn't load default configuration. Have you set up your AWS
 account?")
  fmt.Println(err)
  return
 }
 lambdaClient := lambda.NewFromConfig(sdkConfig)

 maxItems := 10
 fmt.Printf("Let's list up to %v functions for your account.\n", maxItems)
 result, err := lambdaClient.ListFunctions(ctx, &lambda.ListFunctionsInput{
  MaxItems: aws.Int32(int32(maxItems)),
 })
 if err != nil {
  fmt.Printf("Couldn't list functions for your account. Here's why: %v\n", err)
  return
 }
 if len(result.Functions) == 0 {
  fmt.Println("You don't have any functions!")
 } else {
  for _, function := range result.Functions {
   fmt.Printf("\t%v\n", *function.FunctionName)
  }
 }
}
```

- Einzelheiten zur API finden Sie ListFunctionsin der AWS SDK für Go API-Referenz.

Themen

- Grundlagen
- Aktionen
- Szenarien
- Serverless-Beispiele
- AWS Beiträge der Gemeinschaft

Grundlagen

Erlernen der Grundlagen

Wie das aussehen kann, sehen Sie am nachfolgenden Beispielcode:

- Erstellen Sie eine IAM-Rolle und eine Lambda-Funktion und laden Sie den Handlercode hoch.
- Rufen Sie die Funktion mit einem einzigen Parameter auf und erhalten Sie Ergebnisse.
- Aktualisieren Sie den Funktionscode und konfigurieren Sie mit einer Umgebungsvariablen.
- Rufen Sie die Funktion mit neuen Parametern auf und erhalten Sie Ergebnisse. Zeigt das zurückgegebene Ausführungsprotokoll an.
- Listen Sie die Funktionen für Ihr Konto auf und bereinigen Sie dann die Ressourcen.

Weitere Informationen zur Verwendung von Lambda finden Sie unter Erstellen einer Lambda-Funktion mit der Konsole.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Erstellen Sie ein interaktives Szenario, das Ihnen zeigt, wie Sie mit Lambda-Funktionen loslegen können.

```
import (
```

```
 "archive/zip"
 "bytes"
 "context"
 "encoding/base64"
 "encoding/json"
 "errors"
 "fmt"
 "log"
 "os"
 "strings"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/iam"
 iamtypes "github.com/aws/aws-sdk-go-v2/service/iam/types"
 "github.com/aws/aws-sdk-go-v2/service/lambda"
 "github.com/awsdocs/aws-doc-sdk-examples/gov2/demotools"
 "github.com/awsdocs/aws-doc-sdk-examples/gov2/lambda/actions"
)

// GetStartedFunctionsScenario shows you how to use AWS Lambda to perform the
 following
// actions:
//
//  1. Create an AWS Identity and Access Management (IAM) role and Lambda function,
 then upload handler code.
//  2. Invoke the function with a single parameter and get results.
//  3. Update the function code and configure with an environment variable.
//  4. Invoke the function with new parameters and get results. Display the returned
 execution log.
//  5. List the functions for your account, then clean up resources.
type GetStartedFunctionsScenario struct {
 sdkConfig       aws.Config
 functionWrapper actions.FunctionWrapper
 questioner      demotools.IQuestioner
 helper          IScenarioHelper
 isTestRun       bool
}

// NewGetStartedFunctionsScenario constructs a GetStartedFunctionsScenario instance
 from a configuration.
// It uses the specified config to get a Lambda client and create wrappers for the
 actions
// used in the scenario.
```

```
func NewGetStartedFunctionsScenario(sdkConfig aws.Config, questioner
 demotools.IQuestioner,
 helper IScenarioHelper) GetStartedFunctionsScenario {
 lambdaClient := lambda.NewFromConfig(sdkConfig)
 return GetStartedFunctionsScenario{
  sdkConfig:       sdkConfig,
  functionWrapper: actions.FunctionWrapper{LambdaClient: lambdaClient},
  questioner:      questioner,
  helper:          helper,
 }
}

// Run runs the interactive scenario.
func (scenario GetStartedFunctionsScenario) Run(ctx context.Context) {
 defer func() {
  if r := recover(); r != nil {
   log.Printf("Something went wrong with the demo.\n")
  }
 }()

 log.Println(strings.Repeat("-", 88))
 log.Println("Welcome to the AWS Lambda get started with functions demo.")
 log.Println(strings.Repeat("-", 88))

 role := scenario.GetOrCreateRole(ctx)
 funcName := scenario.CreateFunction(ctx, role)
 scenario.InvokeIncrement(ctx, funcName)
 scenario.UpdateFunction(ctx, funcName)
 scenario.InvokeCalculator(ctx, funcName)
 scenario.ListFunctions(ctx)
 scenario.Cleanup(ctx, role, funcName)

 log.Println(strings.Repeat("-", 88))
 log.Println("Thanks for watching!")
 log.Println(strings.Repeat("-", 88))
}

// GetOrCreateRole checks whether the specified role exists and returns it if it
 does.
// Otherwise, a role is created that specifies Lambda as a trusted principal.
// The AWSLambdaBasicExecutionRole managed policy is attached to the role and the
 role
// is returned.
```

```
func (scenario GetStartedFunctionsScenario) GetOrCreateRole(ctx context.Context)
 *iamtypes.Role {
 var role *iamtypes.Role
 iamClient := iam.NewFromConfig(scenario.sdkConfig)
 log.Println("First, we need an IAM role that Lambda can assume.")
 roleName := scenario.questioner.Ask("Enter a name for the role:",
 demotools.NotEmpty{})
 getOutput, err := iamClient.GetRole(ctx, &iam.GetRoleInput{
  RoleName: aws.String(roleName)})
 if err != nil {
  var noSuch *iamtypes.NoSuchEntityException
  if errors.As(err, &noSuch) {
   log.Printf("Role %v doesn't exist. Creating it....\n", roleName)
  } else {
   log.Panicf("Couldn't check whether role %v exists. Here's why: %v\n",
    roleName, err)
  }
 } else {
  role = getOutput.Role
  log.Printf("Found role %v.\n", *role.RoleName)
 }
 if role == nil {
  trustPolicy := PolicyDocument{
   Version: "2012-10-17",
   Statement: []PolicyStatement{{
    Effect:    "Allow",
    Principal: map[string]string{"Service": "lambda.amazonaws.com"},
    Action:    []string{"sts:AssumeRole"},
   }},
  }
  policyArn := "arn:aws:iam::aws:policy/service-role/AWSLambdaBasicExecutionRole"
  createOutput, err := iamClient.CreateRole(ctx, &iam.CreateRoleInput{
   AssumeRolePolicyDocument: aws.String(trustPolicy.String()),
   RoleName:                 aws.String(roleName),
  })
  if err != nil {
   log.Panicf("Couldn't create role %v. Here's why: %v\n", roleName, err)
  }
  role = createOutput.Role
  _, err = iamClient.AttachRolePolicy(ctx, &iam.AttachRolePolicyInput{
   PolicyArn: aws.String(policyArn),
   RoleName:  aws.String(roleName),
  })
  if err != nil {
```

```
    log.Panicf("Couldn't attach a policy to role %v. Here's why: %v\n", roleName,
 err)
  }
  log.Printf("Created role %v.\n", *role.RoleName)
  log.Println("Let's give AWS a few seconds to propagate resources...")
  scenario.helper.Pause(10)
 }
 log.Println(strings.Repeat("-", 88))
 return role
}

// CreateFunction creates a Lambda function and uploads a handler written in Python.
// The code for the Python handler is packaged as a []byte in .zip format.
func (scenario GetStartedFunctionsScenario) CreateFunction(ctx context.Context, role
 *iamtypes.Role) string {
 log.Println("Let's create a function that increments a number.\n" +
  "The function uses the 'lambda_handler_basic.py' script found in the \n" +
  "'handlers' directory of this project.")
 funcName := scenario.questioner.Ask("Enter a name for the Lambda function:",
 demotools.NotEmpty{})
 zipPackage := scenario.helper.CreateDeploymentPackage("lambda_handler_basic.py",
 fmt.Sprintf("%v.py", funcName))
 log.Printf("Creating function %v and waiting for it to be ready.", funcName)
 funcState := scenario.functionWrapper.CreateFunction(ctx, funcName,
 fmt.Sprintf("%v.lambda_handler", funcName),
  role.Arn, zipPackage)
 log.Printf("Your function is %v.", funcState)
 log.Println(strings.Repeat("-", 88))
 return funcName
}

// InvokeIncrement invokes a Lambda function that increments a number. The function
// parameters are contained in a Go struct that is used to serialize the parameters
 to
// a JSON payload that is passed to the function.
// The result payload is deserialized into a Go struct that contains an int value.
func (scenario GetStartedFunctionsScenario) InvokeIncrement(ctx context.Context,
 funcName string) {
 parameters := actions.IncrementParameters{Action: "increment"}
 log.Println("Let's invoke our function. This function increments a number.")
 parameters.Number = scenario.questioner.AskInt("Enter a number to increment:",
 demotools.NotEmpty{})
 log.Printf("Invoking %v with %v...\n", funcName, parameters.Number)
 invokeOutput := scenario.functionWrapper.Invoke(ctx, funcName, parameters, false)
```

```
 var payload actions.LambdaResultInt
 err := json.Unmarshal(invokeOutput.Payload, &payload)
 if err != nil {
  log.Panicf("Couldn't unmarshal payload from invoking %v. Here's why: %v\n",
   funcName, err)
 }
 log.Printf("Invoking %v with %v returned %v.\n", funcName, parameters.Number,
 payload)
 log.Println(strings.Repeat("-", 88))
}

// UpdateFunction updates the code for a Lambda function by uploading a simple
 arithmetic
// calculator written in Python. The code for the Python handler is packaged as a
// []byte in .zip format.
// After the code is updated, the configuration is also updated with a new log
// level that instructs the handler to log additional information.
func (scenario GetStartedFunctionsScenario) UpdateFunction(ctx context.Context,
 funcName string) {
 log.Println("Let's update the function to an arithmetic calculator.\n" +
  "The function uses the 'lambda_handler_calculator.py' script found in the \n" +
  "'handlers' directory of this project.")
 scenario.questioner.Ask("Press Enter when you're ready.")
 log.Println("Creating deployment package...")
 zipPackage :=
 scenario.helper.CreateDeploymentPackage("lambda_handler_calculator.py",
  fmt.Sprintf("%v.py", funcName))
 log.Println("...and updating the Lambda function and waiting for it to be ready.")
 funcState := scenario.functionWrapper.UpdateFunctionCode(ctx, funcName, zipPackage)
 log.Printf("Updated function %v. Its current state is %v.", funcName, funcState)
 log.Println("This function uses an environment variable to control logging level.")
 log.Println("Let's set it to DEBUG to get the most logging.")
 scenario.functionWrapper.UpdateFunctionConfiguration(ctx, funcName,
  map[string]string{"LOG_LEVEL": "DEBUG"})
 log.Println(strings.Repeat("-", 88))
}

// InvokeCalculator invokes the Lambda calculator function. The parameters are
 stored in a
// Go struct that is used to serialize the parameters to a JSON payload. That
 payload is then passed
// to the function.
// The result payload is deserialized to a Go struct that stores the result as
 either an
```

```
// int or float32, depending on the kind of operation that was specified.
func (scenario GetStartedFunctionsScenario) InvokeCalculator(ctx context.Context,
 funcName string) {
 wantInvoke := true
 choices := []string{"plus", "minus", "times", "divided-by"}
 for wantInvoke {
  choice := scenario.questioner.AskChoice("Select an arithmetic operation:\n",
 choices)
  x := scenario.questioner.AskInt("Enter a value for x:", demotools.NotEmpty{})
  y := scenario.questioner.AskInt("Enter a value for y:", demotools.NotEmpty{})
  log.Printf("Invoking %v %v %v...", x, choices[choice], y)
  calcParameters := actions.CalculatorParameters{
   Action: choices[choice],
   X:      x,
   Y:      y,
  }
  invokeOutput := scenario.functionWrapper.Invoke(ctx, funcName, calcParameters,
 true)
  var payload any
  if choice == 3 { // divide-by results in a float.
   payload = actions.LambdaResultFloat{}
  } else {
   payload = actions.LambdaResultInt{}
  }
  err := json.Unmarshal(invokeOutput.Payload, &payload)
  if err != nil {
   log.Panicf("Couldn't unmarshal payload from invoking %v. Here's why: %v\n",
    funcName, err)
  }
  log.Printf("Invoking %v with %v %v %v returned %v.\n", funcName,
   calcParameters.X, calcParameters.Action, calcParameters.Y, payload)
  scenario.questioner.Ask("Press Enter to see the logs from the call.")
  logRes, err := base64.StdEncoding.DecodeString(*invokeOutput.LogResult)
  if err != nil {
   log.Panicf("Couldn't decode log result. Here's why: %v\n", err)
  }
  log.Println(string(logRes))
  wantInvoke = scenario.questioner.AskBool("Do you want to calculate again? (y/n)",
 "y")
 }
 log.Println(strings.Repeat("-", 88))
}

// ListFunctions lists up to the specified number of functions for your account.
```

```
func (scenario GetStartedFunctionsScenario) ListFunctions(ctx context.Context) {
 count := scenario.questioner.AskInt(
  "Let's list functions for your account. How many do you want to see?",
 demotools.NotEmpty{})
 functions := scenario.functionWrapper.ListFunctions(ctx, count)
 log.Printf("Found %v functions:", len(functions))
 for _, function := range functions {
  log.Printf("\t%v", *function.FunctionName)
 }
 log.Println(strings.Repeat("-", 88))
}

// Cleanup removes the IAM and Lambda resources created by the example.
func (scenario GetStartedFunctionsScenario) Cleanup(ctx context.Context, role
 *iamtypes.Role, funcName string) {
 if scenario.questioner.AskBool("Do you want to clean up resources created for this
 example? (y/n)",
  "y") {
  iamClient := iam.NewFromConfig(scenario.sdkConfig)
  policiesOutput, err := iamClient.ListAttachedRolePolicies(ctx,
   &iam.ListAttachedRolePoliciesInput{RoleName: role.RoleName})
  if err != nil {
   log.Panicf("Couldn't get policies attached to role %v. Here's why: %v\n",
    *role.RoleName, err)
  }
  for _, policy := range policiesOutput.AttachedPolicies {
   _, err = iamClient.DetachRolePolicy(ctx, &iam.DetachRolePolicyInput{
    PolicyArn: policy.PolicyArn, RoleName: role.RoleName,
   })
   if err != nil {
    log.Panicf("Couldn't detach policy %v from role %v. Here's why: %v\n",
     *policy.PolicyArn, *role.RoleName, err)
   }
  }
  _, err = iamClient.DeleteRole(ctx, &iam.DeleteRoleInput{RoleName: role.RoleName})
  if err != nil {
   log.Panicf("Couldn't delete role %v. Here's why: %v\n", *role.RoleName, err)
  }
  log.Printf("Deleted role %v.\n", *role.RoleName)

  scenario.functionWrapper.DeleteFunction(ctx, funcName)
  log.Printf("Deleted function %v.\n", funcName)
 } else {
```

```
  log.Println("Okay. Don't forget to delete the resources when you're done with
 them.")
 }
}

// IScenarioHelper abstracts I/O and wait functions from a scenario so that they
// can be mocked for unit testing.
type IScenarioHelper interface {
 Pause(secs int)
 CreateDeploymentPackage(sourceFile string, destinationFile string) *bytes.Buffer
}

// ScenarioHelper lets the caller specify the path to Lambda handler functions.
type ScenarioHelper struct {
 HandlerPath string
}

// Pause waits for the specified number of seconds.
func (helper *ScenarioHelper) Pause(secs int) {
 time.Sleep(time.Duration(secs) * time.Second)
}

// CreateDeploymentPackage creates an AWS Lambda deployment package from a source
 file. The
// deployment package is stored in .zip format in a bytes.Buffer. The buffer can be
// used to pass a []byte to Lambda when creating the function.
// The specified destinationFile is the name to give the file when it's deployed to
 Lambda.
func (helper *ScenarioHelper) CreateDeploymentPackage(sourceFile string,
 destinationFile string) *bytes.Buffer {
 var err error
 buffer := &bytes.Buffer{}
 writer := zip.NewWriter(buffer)
 zFile, err := writer.Create(destinationFile)
 if err != nil {
  log.Panicf("Couldn't create destination archive %v. Here's why: %v\n",
 destinationFile, err)
 }
 sourceBody, err := os.ReadFile(fmt.Sprintf("%v/%v", helper.HandlerPath,
 sourceFile))
 if err != nil {
  log.Panicf("Couldn't read handler source file %v. Here's why: %v\n",
   sourceFile, err)
 } else {
```

```
  _, err = zFile.Write(sourceBody)
  if err != nil {
   log.Panicf("Couldn't write handler %v to zip archive. Here's why: %v\n",
    sourceFile, err)
  }
 }
 err = writer.Close()
 if err != nil {
  log.Panicf("Couldn't close zip writer. Here's why: %v\n", err)
 }
 return buffer
}
```

Erstellen einer Struktur, die die einzelnen Lambda-Aktionen umschließt.

```
import (
 "bytes"
 "context"
 "encoding/json"
 "errors"
 "log"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/lambda"
 "github.com/aws/aws-sdk-go-v2/service/lambda/types"
)

// FunctionWrapper encapsulates function actions used in the examples.
// It contains an AWS Lambda service client that is used to perform user actions.
type FunctionWrapper struct {
 LambdaClient *lambda.Client
}


// GetFunction gets data about the Lambda function specified by functionName.
func (wrapper FunctionWrapper) GetFunction(ctx context.Context, functionName string)
 types.State {
 var state types.State
 funcOutput, err := wrapper.LambdaClient.GetFunction(ctx, &lambda.GetFunctionInput{
```

```
  FunctionName: aws.String(functionName),
 })
 if err != nil {
  log.Panicf("Couldn't get function %v. Here's why: %v\n", functionName, err)
 } else {
  state = funcOutput.Configuration.State
 }
 return state
}



// CreateFunction creates a new Lambda function from code contained in the
 zipPackage
// buffer. The specified handlerName must match the name of the file and function
// contained in the uploaded code. The role specified by iamRoleArn is assumed by
// Lambda and grants specific permissions.
// When the function already exists, types.StateActive is returned.
// When the function is created, a lambda.FunctionActiveV2Waiter is used to wait
 until the
// function is active.
func (wrapper FunctionWrapper) CreateFunction(ctx context.Context, functionName
 string, handlerName string,
 iamRoleArn *string, zipPackage *bytes.Buffer) types.State {
 var state types.State
 _, err := wrapper.LambdaClient.CreateFunction(ctx, &lambda.CreateFunctionInput{
  Code:         &types.FunctionCode{ZipFile: zipPackage.Bytes()},
  FunctionName: aws.String(functionName),
  Role:         iamRoleArn,
  Handler:      aws.String(handlerName),
  Publish:      true,
  Runtime:      types.RuntimePython39,
 })
 if err != nil {
  var resConflict *types.ResourceConflictException
  if errors.As(err, &resConflict) {
   log.Printf("Function %v already exists.\n", functionName)
   state = types.StateActive
  } else {
   log.Panicf("Couldn't create function %v. Here's why: %v\n", functionName, err)
  }
 } else {
  waiter := lambda.NewFunctionActiveV2Waiter(wrapper.LambdaClient)
  funcOutput, err := waiter.WaitForOutput(ctx, &lambda.GetFunctionInput{
```

```
   FunctionName: aws.String(functionName)}, 1*time.Minute)
  if err != nil {
   log.Panicf("Couldn't wait for function %v to be active. Here's why: %v\n",
 functionName, err)
  } else {
   state = funcOutput.Configuration.State
  }
 }
 return state
}



// UpdateFunctionCode updates the code for the Lambda function specified by
 functionName.
// The existing code for the Lambda function is entirely replaced by the code in the
// zipPackage buffer. After the update action is called, a
 lambda.FunctionUpdatedV2Waiter
// is used to wait until the update is successful.
func (wrapper FunctionWrapper) UpdateFunctionCode(ctx context.Context, functionName
 string, zipPackage *bytes.Buffer) types.State {
 var state types.State
 _, err := wrapper.LambdaClient.UpdateFunctionCode(ctx,
 &lambda.UpdateFunctionCodeInput{
  FunctionName: aws.String(functionName), ZipFile: zipPackage.Bytes(),
 })
 if err != nil {
  log.Panicf("Couldn't update code for function %v. Here's why: %v\n", functionName,
 err)
 } else {
  waiter := lambda.NewFunctionUpdatedV2Waiter(wrapper.LambdaClient)
  funcOutput, err := waiter.WaitForOutput(ctx, &lambda.GetFunctionInput{
   FunctionName: aws.String(functionName)}, 1*time.Minute)
  if err != nil {
   log.Panicf("Couldn't wait for function %v to be active. Here's why: %v\n",
 functionName, err)
  } else {
   state = funcOutput.Configuration.State
  }
 }
 return state
}
```

```
// UpdateFunctionConfiguration updates a map of environment variables configured for
// the Lambda function specified by functionName.
func (wrapper FunctionWrapper) UpdateFunctionConfiguration(ctx context.Context,
 functionName string, envVars map[string]string) {
 _, err := wrapper.LambdaClient.UpdateFunctionConfiguration(ctx,
 &lambda.UpdateFunctionConfigurationInput{
  FunctionName: aws.String(functionName),
  Environment:  &types.Environment{Variables: envVars},
 })
 if err != nil {
  log.Panicf("Couldn't update configuration for %v. Here's why: %v", functionName,
 err)
 }
}



// ListFunctions lists up to maxItems functions for the account. This function uses
 a
// lambda.ListFunctionsPaginator to paginate the results.
func (wrapper FunctionWrapper) ListFunctions(ctx context.Context, maxItems int)
 []types.FunctionConfiguration {
 var functions []types.FunctionConfiguration
 paginator := lambda.NewListFunctionsPaginator(wrapper.LambdaClient,
 &lambda.ListFunctionsInput{
  MaxItems: aws.Int32(int32(maxItems)),
 })
 for paginator.HasMorePages() && len(functions) < maxItems {
  pageOutput, err := paginator.NextPage(ctx)
  if err != nil {
   log.Panicf("Couldn't list functions for your account. Here's why: %v\n", err)
  }
  functions = append(functions, pageOutput.Functions...)
 }
 return functions
}



// DeleteFunction deletes the Lambda function specified by functionName.
func (wrapper FunctionWrapper) DeleteFunction(ctx context.Context, functionName
 string) {
 _, err := wrapper.LambdaClient.DeleteFunction(ctx, &lambda.DeleteFunctionInput{
```

```
  FunctionName: aws.String(functionName),
 })
 if err != nil {
  log.Panicf("Couldn't delete function %v. Here's why: %v\n", functionName, err)
 }
}



// Invoke invokes the Lambda function specified by functionName, passing the
 parameters
// as a JSON payload. When getLog is true, types.LogTypeTail is specified, which
 tells
// Lambda to include the last few log lines in the returned result.
func (wrapper FunctionWrapper) Invoke(ctx context.Context, functionName string,
 parameters any, getLog bool) *lambda.InvokeOutput {
 logType := types.LogTypeNone
 if getLog {
  logType = types.LogTypeTail
 }
 payload, err := json.Marshal(parameters)
 if err != nil {
  log.Panicf("Couldn't marshal parameters to JSON. Here's why %v\n", err)
 }
 invokeOutput, err := wrapper.LambdaClient.Invoke(ctx, &lambda.InvokeInput{
  FunctionName: aws.String(functionName),
  LogType:      logType,
  Payload:      payload,
 })
 if err != nil {
  log.Panicf("Couldn't invoke function %v. Here's why: %v\n", functionName, err)
 }
 return invokeOutput
}



// IncrementParameters is used to serialize parameters to the increment Lambda
 handler.
type IncrementParameters struct {
 Action string `json:"action"`
 Number int    `json:"number"`
}
```

```
// CalculatorParameters is used to serialize parameters to the calculator Lambda
 handler.
type CalculatorParameters struct {
 Action string `json:"action"`
 X      int    `json:"x"`
 Y      int    `json:"y"`
}

// LambdaResultInt is used to deserialize an int result from a Lambda handler.
type LambdaResultInt struct {
 Result int `json:"result"`
}

// LambdaResultFloat is used to deserialize a float32 result from a Lambda handler.
type LambdaResultFloat struct {
 Result float32 `json:"result"`
}
```

Definieren Sie einen Lambda-Handler, der eine Zahl inkrementiert.

```
import logging

logger = logging.getLogger()
logger.setLevel(logging.INFO)


def lambda_handler(event, context):
    """
    Accepts an action and a single number, performs the specified action on the
 number,
    and returns the result. The only allowable action is 'increment'.

    :param event: The event dict that contains the parameters sent when the function
                  is invoked.
    :param context: The context in which the function is called.
    :return: The result of the action.
    """
    result = None
    action = event.get("action")
    if action == "increment":
        result = event.get("number", 0) + 1
```

```
        logger.info("Calculated result of %s", result)
    else:
        logger.error("%s is not a valid action.", action)

    response = {"result": result}
    return response
```

Definieren Sie einen zweiten Lambda-Handler, der arithmetische Operationen ausführt.

```
import logging
import os


logger = logging.getLogger()

# Define a list of Python lambda functions that are called by this AWS Lambda
 function.
ACTIONS = {
    "plus": lambda x, y: x + y,
    "minus": lambda x, y: x - y,
    "times": lambda x, y: x * y,
    "divided-by": lambda x, y: x / y,
}


def lambda_handler(event, context):
    """
    Accepts an action and two numbers, performs the specified action on the numbers,
    and returns the result.

    :param event: The event dict that contains the parameters sent when the function
                  is invoked.
    :param context: The context in which the function is called.
    :return: The result of the specified action.
    """
    # Set the log level based on a variable configured in the Lambda environment.
    logger.setLevel(os.environ.get("LOG_LEVEL", logging.INFO))
    logger.debug("Event: %s", event)

    action = event.get("action")
```

```
    func = ACTIONS.get(action)
    x = event.get("x")
    y = event.get("y")
    result = None
    try:
        if func is not None and x is not None and y is not None:
            result = func(x, y)
            logger.info("%s %s %s is %s", x, action, y, result)
        else:
            logger.error("I can't calculate %s %s %s.", x, action, y)
    except ZeroDivisionError:
        logger.warning("I can't divide %s by 0!", x)

    response = {"result": result}
    return response
```

- API-Details finden Sie in den folgenden Themen der AWS SDK für Go -API-Referenz.
  - CreateFunction
  - DeleteFunction
  - GetFunction
  - Aufrufen
  - ListFunctions
  - UpdateFunctionCode
  - UpdateFunctionConfiguration

Aktionen

`CreateFunction`

Das folgende Codebeispiel zeigt, wie man es benutzt`CreateFunction`.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import (
 "bytes"
 "context"
 "encoding/json"
 "errors"
 "log"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/lambda"
 "github.com/aws/aws-sdk-go-v2/service/lambda/types"
)

// FunctionWrapper encapsulates function actions used in the examples.
// It contains an AWS Lambda service client that is used to perform user actions.
type FunctionWrapper struct {
 LambdaClient *lambda.Client
}



// CreateFunction creates a new Lambda function from code contained in the
 zipPackage
// buffer. The specified handlerName must match the name of the file and function
// contained in the uploaded code. The role specified by iamRoleArn is assumed by
// Lambda and grants specific permissions.
// When the function already exists, types.StateActive is returned.
// When the function is created, a lambda.FunctionActiveV2Waiter is used to wait
 until the
// function is active.
func (wrapper FunctionWrapper) CreateFunction(ctx context.Context, functionName
 string, handlerName string,
 iamRoleArn *string, zipPackage *bytes.Buffer) types.State {
```

```
 var state types.State
 _, err := wrapper.LambdaClient.CreateFunction(ctx, &lambda.CreateFunctionInput{
  Code:         &types.FunctionCode{ZipFile: zipPackage.Bytes()},
  FunctionName: aws.String(functionName),
  Role:         iamRoleArn,
  Handler:      aws.String(handlerName),
  Publish:      true,
  Runtime:      types.RuntimePython39,
 })
 if err != nil {
  var resConflict *types.ResourceConflictException
  if errors.As(err, &resConflict) {
   log.Printf("Function %v already exists.\n", functionName)
   state = types.StateActive
  } else {
   log.Panicf("Couldn't create function %v. Here's why: %v\n", functionName, err)
  }
 } else {
  waiter := lambda.NewFunctionActiveV2Waiter(wrapper.LambdaClient)
  funcOutput, err := waiter.WaitForOutput(ctx, &lambda.GetFunctionInput{
   FunctionName: aws.String(functionName)}, 1*time.Minute)
  if err != nil {
   log.Panicf("Couldn't wait for function %v to be active. Here's why: %v\n",
 functionName, err)
  } else {
   state = funcOutput.Configuration.State
  }
 }
 return state
}
```

- Einzelheiten zur API finden Sie CreateFunctionin der AWS SDK für Go API-Referenz.

`DeleteFunction`

Das folgende Codebeispiel zeigt die Verwendung`DeleteFunction`.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import (
 "bytes"
 "context"
 "encoding/json"
 "errors"
 "log"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/lambda"
 "github.com/aws/aws-sdk-go-v2/service/lambda/types"
)

// FunctionWrapper encapsulates function actions used in the examples.
// It contains an AWS Lambda service client that is used to perform user actions.
type FunctionWrapper struct {
 LambdaClient *lambda.Client
}



// DeleteFunction deletes the Lambda function specified by functionName.
func (wrapper FunctionWrapper) DeleteFunction(ctx context.Context, functionName
 string) {
 _, err := wrapper.LambdaClient.DeleteFunction(ctx, &lambda.DeleteFunctionInput{
  FunctionName: aws.String(functionName),
 })
 if err != nil {
  log.Panicf("Couldn't delete function %v. Here's why: %v\n", functionName, err)
 }
}
```

- Einzelheiten zur API finden Sie DeleteFunctionin der AWS SDK für Go API-Referenz.

`GetFunction`

Das folgende Codebeispiel zeigt die VerwendungGetFunction.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import (
 "bytes"
 "context"
 "encoding/json"
 "errors"
 "log"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/lambda"
 "github.com/aws/aws-sdk-go-v2/service/lambda/types"
)

// FunctionWrapper encapsulates function actions used in the examples.
// It contains an AWS Lambda service client that is used to perform user actions.
type FunctionWrapper struct {
 LambdaClient *lambda.Client
}



// GetFunction gets data about the Lambda function specified by functionName.
func (wrapper FunctionWrapper) GetFunction(ctx context.Context, functionName string)
 types.State {
 var state types.State
 funcOutput, err := wrapper.LambdaClient.GetFunction(ctx, &lambda.GetFunctionInput{
  FunctionName: aws.String(functionName),
```

```
 })
 if err != nil {
  log.Panicf("Couldn't get function %v. Here's why: %v\n", functionName, err)
 } else {
  state = funcOutput.Configuration.State
 }
 return state
}
```

- Einzelheiten zur API finden Sie GetFunctionin der AWS SDK für Go API-Referenz.

`Invoke`

Das folgende Codebeispiel zeigt die Verwendung`Invoke`.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import (
 "bytes"
 "context"
 "encoding/json"
 "errors"
 "log"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/lambda"
 "github.com/aws/aws-sdk-go-v2/service/lambda/types"
)

// FunctionWrapper encapsulates function actions used in the examples.
// It contains an AWS Lambda service client that is used to perform user actions.
```

```
type FunctionWrapper struct {
 LambdaClient *lambda.Client
}



// Invoke invokes the Lambda function specified by functionName, passing the
 parameters
// as a JSON payload. When getLog is true, types.LogTypeTail is specified, which
 tells
// Lambda to include the last few log lines in the returned result.
func (wrapper FunctionWrapper) Invoke(ctx context.Context, functionName string,
 parameters any, getLog bool) *lambda.InvokeOutput {
 logType := types.LogTypeNone
 if getLog {
  logType = types.LogTypeTail
 }
 payload, err := json.Marshal(parameters)
 if err != nil {
  log.Panicf("Couldn't marshal parameters to JSON. Here's why %v\n", err)
 }
 invokeOutput, err := wrapper.LambdaClient.Invoke(ctx, &lambda.InvokeInput{
  FunctionName: aws.String(functionName),
  LogType:      logType,
  Payload:      payload,
 })
 if err != nil {
  log.Panicf("Couldn't invoke function %v. Here's why: %v\n", functionName, err)
 }
 return invokeOutput
}
```

- Weitere API-Informationen finden Sie unter Invoke in der AWS SDK für Go -API-Referenz.

`ListFunctions`

Das folgende Codebeispiel zeigt, wie man es benutzt`ListFunctions`.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import (
 "bytes"
 "context"
 "encoding/json"
 "errors"
 "log"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/lambda"
 "github.com/aws/aws-sdk-go-v2/service/lambda/types"
)

// FunctionWrapper encapsulates function actions used in the examples.
// It contains an AWS Lambda service client that is used to perform user actions.
type FunctionWrapper struct {
 LambdaClient *lambda.Client
}



// ListFunctions lists up to maxItems functions for the account. This function uses
 a
// lambda.ListFunctionsPaginator to paginate the results.
func (wrapper FunctionWrapper) ListFunctions(ctx context.Context, maxItems int)
 []types.FunctionConfiguration {
 var functions []types.FunctionConfiguration
 paginator := lambda.NewListFunctionsPaginator(wrapper.LambdaClient,
 &lambda.ListFunctionsInput{
  MaxItems: aws.Int32(int32(maxItems)),
 })
 for paginator.HasMorePages() && len(functions) < maxItems {
  pageOutput, err := paginator.NextPage(ctx)
```

```
  if err != nil {
   log.Panicf("Couldn't list functions for your account. Here's why: %v\n", err)
  }
  functions = append(functions, pageOutput.Functions...)
 }
 return functions
}
```

- Einzelheiten zur API finden Sie ListFunctionsin der AWS SDK für Go API-Referenz.

`UpdateFunctionCode`

Das folgende Codebeispiel zeigt die Verwendung`UpdateFunctionCode`.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import (
 "bytes"
 "context"
 "encoding/json"
 "errors"
 "log"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/lambda"
 "github.com/aws/aws-sdk-go-v2/service/lambda/types"
)

// FunctionWrapper encapsulates function actions used in the examples.
// It contains an AWS Lambda service client that is used to perform user actions.
type FunctionWrapper struct {
```

```
 LambdaClient *lambda.Client
}



// UpdateFunctionCode updates the code for the Lambda function specified by
 functionName.
// The existing code for the Lambda function is entirely replaced by the code in the
// zipPackage buffer. After the update action is called, a
 lambda.FunctionUpdatedV2Waiter
// is used to wait until the update is successful.
func (wrapper FunctionWrapper) UpdateFunctionCode(ctx context.Context, functionName
 string, zipPackage *bytes.Buffer) types.State {
 var state types.State
 _, err := wrapper.LambdaClient.UpdateFunctionCode(ctx,
 &lambda.UpdateFunctionCodeInput{
  FunctionName: aws.String(functionName), ZipFile: zipPackage.Bytes(),
 })
 if err != nil {
  log.Panicf("Couldn't update code for function %v. Here's why: %v\n", functionName,
 err)
 } else {
  waiter := lambda.NewFunctionUpdatedV2Waiter(wrapper.LambdaClient)
  funcOutput, err := waiter.WaitForOutput(ctx, &lambda.GetFunctionInput{
   FunctionName: aws.String(functionName)}, 1*time.Minute)
  if err != nil {
   log.Panicf("Couldn't wait for function %v to be active. Here's why: %v\n",
 functionName, err)
  } else {
   state = funcOutput.Configuration.State
  }
 }
 return state
}
```

- Einzelheiten zur API finden Sie UpdateFunctionCodein der AWS SDK für Go API-Referenz.

`UpdateFunctionConfiguration`

Das folgende Codebeispiel zeigt die Verwendung`UpdateFunctionConfiguration`.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import (
 "bytes"
 "context"
 "encoding/json"
 "errors"
 "log"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/lambda"
 "github.com/aws/aws-sdk-go-v2/service/lambda/types"
)

// FunctionWrapper encapsulates function actions used in the examples.
// It contains an AWS Lambda service client that is used to perform user actions.
type FunctionWrapper struct {
 LambdaClient *lambda.Client
}



// UpdateFunctionConfiguration updates a map of environment variables configured for
// the Lambda function specified by functionName.
func (wrapper FunctionWrapper) UpdateFunctionConfiguration(ctx context.Context,
 functionName string, envVars map[string]string) {
 _, err := wrapper.LambdaClient.UpdateFunctionConfiguration(ctx,
 &lambda.UpdateFunctionConfigurationInput{
  FunctionName: aws.String(functionName),
  Environment:  &types.Environment{Variables: envVars},
 })
 if err != nil {
  log.Panicf("Couldn't update configuration for %v. Here's why: %v", functionName,
 err)
```

```
 }
}
```

- Einzelheiten zur API finden Sie UpdateFunctionConfigurationin der AWS SDK für Go API-Referenz.

Szenarien

Bestätigen Sie automatisch bekannte Benutzer mit einer Lambda-Funktion

Das folgende Codebeispiel zeigt, wie bekannte Amazon Cognito Cognito-Benutzer automatisch mit einer Lambda-Funktion bestätigt werden.

- Konfigurieren Sie einen Benutzerpool, um eine Lambda-Funktion für den `PreSignUp`-Trigger aufzurufen.
- Registrieren eines Benutzers bei Amazon Cognito.
- Die Lambda-Funktion scannt eine DynamoDB-Tabelle und bestätigt automatisch bekannte Benutzer.
- Melden Sie sich als neuer Benutzer an und bereinigen Sie dann die Ressourcen.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Führen Sie ein interaktives Szenario an einem Prompt aus.

```
import (
 "context"
 "errors"
 "log"
 "strings"
 "user_pools_and_lambda_triggers/actions"
```

```
 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/cognitoidentityprovider"
 "github.com/aws/aws-sdk-go-v2/service/cognitoidentityprovider/types"
 "github.com/awsdocs/aws-doc-sdk-examples/gov2/demotools"
)

// AutoConfirm separates the steps of this scenario into individual functions so
 that
// they are simpler to read and understand.
type AutoConfirm struct {
 helper       IScenarioHelper
 questioner   demotools.IQuestioner
 resources    Resources
 cognitoActor *actions.CognitoActions
}

// NewAutoConfirm constructs a new auto confirm runner.
func NewAutoConfirm(sdkConfig aws.Config, questioner demotools.IQuestioner, helper
 IScenarioHelper) AutoConfirm {
 scenario := AutoConfirm{
  helper:       helper,
  questioner:   questioner,
  resources:    Resources{},
  cognitoActor: &actions.CognitoActions{CognitoClient:
 cognitoidentityprovider.NewFromConfig(sdkConfig)},
 }
 scenario.resources.init(scenario.cognitoActor, questioner)
 return scenario
}

// AddPreSignUpTrigger adds a Lambda handler as an invocation target for the
 PreSignUp trigger.
func (runner *AutoConfirm) AddPreSignUpTrigger(ctx context.Context, userPoolId
 string, functionArn string) {
 log.Printf("Let's add a Lambda function to handle the PreSignUp trigger from
 Cognito.\n" +
  "This trigger happens when a user signs up, and lets your function take action
 before the main Cognito\n" +
  "sign up processing occurs.\n")
 err := runner.cognitoActor.UpdateTriggers(
  ctx, userPoolId,
  actions.TriggerInfo{Trigger: actions.PreSignUp, HandlerArn:
 aws.String(functionArn)})
```

```
 if err != nil {
  panic(err)
 }
 log.Printf("Lambda function %v added to user pool %v to handle the PreSignUp
 trigger.\n",
  functionArn, userPoolId)
}

// SignUpUser signs up a user from the known user table with a password you specify.
func (runner *AutoConfirm) SignUpUser(ctx context.Context, clientId string,
 usersTable string) (string, string) {
 log.Println("Let's sign up a user to your Cognito user pool. When the user's email
 matches an email in the\n" +
  "DynamoDB known users table, it is automatically verified and the user is
 confirmed.")

 knownUsers, err := runner.helper.GetKnownUsers(ctx, usersTable)
 if err != nil {
  panic(err)
 }
 userChoice := runner.questioner.AskChoice("Which user do you want to use?\n",
 knownUsers.UserNameList())
 user := knownUsers.Users[userChoice]

 var signedUp bool
 var userConfirmed bool
 password := runner.questioner.AskPassword("Enter a password that has at least eight
 characters, uppercase, lowercase, numbers and symbols.\n"+
  "(the password will not display as you type):", 8)
 for !signedUp {
  log.Printf("Signing up user '%v' with email '%v' to Cognito.\n", user.UserName,
 user.UserEmail)
  userConfirmed, err = runner.cognitoActor.SignUp(ctx, clientId, user.UserName,
 password, user.UserEmail)
  if err != nil {
   var invalidPassword *types.InvalidPasswordException
   if errors.As(err, &invalidPassword) {
    password = runner.questioner.AskPassword("Enter another password:", 8)
   } else {
    panic(err)
   }
  } else {
   signedUp = true
  }
```

```
 }
 log.Printf("User %v signed up, confirmed = %v.\n", user.UserName, userConfirmed)

 log.Println(strings.Repeat("-", 88))

 return user.UserName, password
}

// SignInUser signs in a user.
func (runner *AutoConfirm) SignInUser(ctx context.Context, clientId string, userName
 string, password string) string {
 runner.questioner.Ask("Press Enter when you're ready to continue.")
 log.Printf("Let's sign in as %v...\n", userName)
 authResult, err := runner.cognitoActor.SignIn(ctx, clientId, userName, password)
 if err != nil {
  panic(err)
 }
 log.Printf("Successfully signed in. Your access token starts with: %v...\n",
 (*authResult.AccessToken)[:10])
 log.Println(strings.Repeat("-", 88))
 return *authResult.AccessToken
}

// Run runs the scenario.
func (runner *AutoConfirm) Run(ctx context.Context, stackName string) {
 defer func() {
  if r := recover(); r != nil {
   log.Println("Something went wrong with the demo.")
   runner.resources.Cleanup(ctx)
  }
 }()

 log.Println(strings.Repeat("-", 88))
 log.Printf("Welcome\n")

 log.Println(strings.Repeat("-", 88))

 stackOutputs, err := runner.helper.GetStackOutputs(ctx, stackName)
 if err != nil {
  panic(err)
 }
 runner.resources.userPoolId = stackOutputs["UserPoolId"]
 runner.helper.PopulateUserTable(ctx, stackOutputs["TableName"])
```

```
 runner.AddPreSignUpTrigger(ctx, stackOutputs["UserPoolId"],
 stackOutputs["AutoConfirmFunctionArn"])
 runner.resources.triggers = append(runner.resources.triggers, actions.PreSignUp)
 userName, password := runner.SignUpUser(ctx, stackOutputs["UserPoolClientId"],
 stackOutputs["TableName"])
 runner.helper.ListRecentLogEvents(ctx, stackOutputs["AutoConfirmFunction"])
 runner.resources.userAccessTokens = append(runner.resources.userAccessTokens,
  runner.SignInUser(ctx, stackOutputs["UserPoolClientId"], userName, password))

 runner.resources.Cleanup(ctx)

 log.Println(strings.Repeat("-", 88))
 log.Println("Thanks for watching!")
 log.Println(strings.Repeat("-", 88))
}
```

Behandeln Sie den `PreSignUp`-Trigger mit einer Lambda-Funktion.

```
import (
 "context"
 "log"
 "os"

 "github.com/aws/aws-lambda-go/events"
 "github.com/aws/aws-lambda-go/lambda"
 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/config"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 dynamodbtypes "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

const TABLE_NAME = "TABLE_NAME"

// UserInfo defines structured user data that can be marshalled to a DynamoDB
 format.
type UserInfo struct {
 UserName  string `dynamodbav:"UserName"`
 UserEmail string `dynamodbav:"UserEmail"`
}
```

```
// GetKey marshals the user email value to a DynamoDB key format.
func (user UserInfo) GetKey() map[string]dynamodbtypes.AttributeValue {
 userEmail, err := attributevalue.Marshal(user.UserEmail)
 if err != nil {
  panic(err)
 }
 return map[string]dynamodbtypes.AttributeValue{"UserEmail": userEmail}
}

type handler struct {
 dynamoClient *dynamodb.Client
}

// HandleRequest handles the PreSignUp event by looking up a user in an Amazon
 DynamoDB table and
// specifying whether they should be confirmed and verified.
func (h *handler) HandleRequest(ctx context.Context, event
 events.CognitoEventUserPoolsPreSignup) (events.CognitoEventUserPoolsPreSignup,
 error) {
 log.Printf("Received presignup from %v for user '%v'", event.TriggerSource,
 event.UserName)
 if event.TriggerSource != "PreSignUp_SignUp" {
  // Other trigger sources, such as PreSignUp_AdminInitiateAuth, ignore the response
 from this handler.
  return event, nil
 }
 tableName := os.Getenv(TABLE_NAME)
 user := UserInfo{
  UserEmail: event.Request.UserAttributes["email"],
 }
 log.Printf("Looking up email %v in table %v.\n", user.UserEmail, tableName)
 output, err := h.dynamoClient.GetItem(ctx, &dynamodb.GetItemInput{
  Key:       user.GetKey(),
  TableName: aws.String(tableName),
 })
 if err != nil {
  log.Printf("Error looking up email %v.\n", user.UserEmail)
  return event, err
 }
 if output.Item == nil {
  log.Printf("Email %v not found. Email verification is required.\n",
 user.UserEmail)
  return event, err
```

```
 }

 err = attributevalue.UnmarshalMap(output.Item, &user)
 if err != nil {
  log.Printf("Couldn't unmarshal DynamoDB item. Here's why: %v\n", err)
  return event, err
 }

 if user.UserName != event.UserName {
  log.Printf("UserEmail %v found, but stored UserName '%v' does not match supplied
 UserName '%v'. Verification is required.\n",
   user.UserEmail, user.UserName, event.UserName)
 } else {
  log.Printf("UserEmail %v found with matching UserName %v. User is confirmed.\n",
 user.UserEmail, user.UserName)
  event.Response.AutoConfirmUser = true
  event.Response.AutoVerifyEmail = true
 }

 return event, err
}

func main() {
 ctx := context.Background()
 sdkConfig, err := config.LoadDefaultConfig(ctx)
 if err != nil {
  log.Panicln(err)
 }
 h := handler{
  dynamoClient: dynamodb.NewFromConfig(sdkConfig),
 }
 lambda.Start(h.HandleRequest)
}
```

Erstellen Sie eine Struktur, die allgemeine Aufgaben ausführt.

```
import (
 "context"
 "log"
 "strings"
```

```
 "time"
 "user_pools_and_lambda_triggers/actions"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/cloudformation"
 "github.com/aws/aws-sdk-go-v2/service/cloudwatchlogs"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/awsdocs/aws-doc-sdk-examples/gov2/demotools"
)

// IScenarioHelper defines common functions used by the workflows in this example.
type IScenarioHelper interface {
 Pause(secs int)
 GetStackOutputs(ctx context.Context, stackName string) (actions.StackOutputs,
 error)
 PopulateUserTable(ctx context.Context, tableName string)
 GetKnownUsers(ctx context.Context, tableName string) (actions.UserList, error)
 AddKnownUser(ctx context.Context, tableName string, user actions.User)
 ListRecentLogEvents(ctx context.Context, functionName string)
}

// ScenarioHelper contains AWS wrapper structs used by the workflows in this
 example.
type ScenarioHelper struct {
 questioner  demotools.IQuestioner
 dynamoActor *actions.DynamoActions
 cfnActor    *actions.CloudFormationActions
 cwlActor    *actions.CloudWatchLogsActions
 isTestRun   bool
}

// NewScenarioHelper constructs a new scenario helper.
func NewScenarioHelper(sdkConfig aws.Config, questioner demotools.IQuestioner)
 ScenarioHelper {
 scenario := ScenarioHelper{
  questioner:  questioner,
  dynamoActor: &actions.DynamoActions{DynamoClient:
 dynamodb.NewFromConfig(sdkConfig)},
  cfnActor:    &actions.CloudFormationActions{CfnClient:
 cloudformation.NewFromConfig(sdkConfig)},
  cwlActor:    &actions.CloudWatchLogsActions{CwlClient:
 cloudwatchlogs.NewFromConfig(sdkConfig)},
 }
 return scenario
```

```
}

// Pause waits for the specified number of seconds.
func (helper ScenarioHelper) Pause(secs int) {
 if !helper.isTestRun {
  time.Sleep(time.Duration(secs) * time.Second)
 }
}

// GetStackOutputs gets the outputs from the specified CloudFormation stack in a
 structured format.
func (helper ScenarioHelper) GetStackOutputs(ctx context.Context, stackName string)
 (actions.StackOutputs, error) {
 return helper.cfnActor.GetOutputs(ctx, stackName), nil
}

// PopulateUserTable fills the known user table with example data.
func (helper ScenarioHelper) PopulateUserTable(ctx context.Context, tableName
 string) {
 log.Printf("First, let's add some users to the DynamoDB %v table we'll use for this
 example.\n", tableName)
 err := helper.dynamoActor.PopulateTable(ctx, tableName)
 if err != nil {
  panic(err)
 }
}

// GetKnownUsers gets the users from the known users table in a structured format.
func (helper ScenarioHelper) GetKnownUsers(ctx context.Context, tableName string)
 (actions.UserList, error) {
 knownUsers, err := helper.dynamoActor.Scan(ctx, tableName)
 if err != nil {
  log.Printf("Couldn't get known users from table %v. Here's why: %v\n", tableName,
 err)
 }
 return knownUsers, err
}

// AddKnownUser adds a user to the known users table.
func (helper ScenarioHelper) AddKnownUser(ctx context.Context, tableName string,
 user actions.User) {
 log.Printf("Adding user '%v' with email '%v' to the DynamoDB known users table...
\n",
  user.UserName, user.UserEmail)
```

```
 err := helper.dynamoActor.AddUser(ctx, tableName, user)
 if err != nil {
  panic(err)
 }
}

// ListRecentLogEvents gets the most recent log stream and events for the specified
 Lambda function and displays them.
func (helper ScenarioHelper) ListRecentLogEvents(ctx context.Context, functionName
 string) {
 log.Println("Waiting a few seconds to let Lambda write to CloudWatch Logs...")
 helper.Pause(10)
 log.Println("Okay, let's check the logs to find what's happened recently with your
 Lambda function.")
 logStream, err := helper.cwlActor.GetLatestLogStream(ctx, functionName)
 if err != nil {
  panic(err)
 }
 log.Printf("Getting some recent events from log stream %v\n",
 *logStream.LogStreamName)
 events, err := helper.cwlActor.GetLogEvents(ctx, functionName,
 *logStream.LogStreamName, 10)
 if err != nil {
  panic(err)
 }
 for _, event := range events {
  log.Printf("\t%v", *event.Message)
 }
 log.Println(strings.Repeat("-", 88))
}
```

Erstellen Sie eine Struktur, die Amazon Cognito-Aktionen umschließt.

```
import (
 "context"
 "errors"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/cognitoidentityprovider"
```

```
 "github.com/aws/aws-sdk-go-v2/service/cognitoidentityprovider/types"
)

type CognitoActions struct {
 CognitoClient *cognitoidentityprovider.Client
}



// Trigger and TriggerInfo define typed data for updating an Amazon Cognito trigger.
type Trigger int

const (
 PreSignUp Trigger = iota
 UserMigration
 PostAuthentication
)

type TriggerInfo struct {
 Trigger    Trigger
 HandlerArn *string
}

// UpdateTriggers adds or removes Lambda triggers for a user pool. When a trigger is
 specified with a `nil` value,
// it is removed from the user pool.
func (actor CognitoActions) UpdateTriggers(ctx context.Context, userPoolId string,
 triggers ...TriggerInfo) error {
 output, err := actor.CognitoClient.DescribeUserPool(ctx,
 &cognitoidentityprovider.DescribeUserPoolInput{
  UserPoolId: aws.String(userPoolId),
 })
 if err != nil {
  log.Printf("Couldn't get info about user pool %v. Here's why: %v\n", userPoolId,
 err)
  return err
 }
 lambdaConfig := output.UserPool.LambdaConfig
 for _, trigger := range triggers {
  switch trigger.Trigger {
  case PreSignUp:
   lambdaConfig.PreSignUp = trigger.HandlerArn
  case UserMigration:
   lambdaConfig.UserMigration = trigger.HandlerArn
```

```
  case PostAuthentication:
   lambdaConfig.PostAuthentication = trigger.HandlerArn
  }
 }
 _, err = actor.CognitoClient.UpdateUserPool(ctx,
 &cognitoidentityprovider.UpdateUserPoolInput{
  UserPoolId:   aws.String(userPoolId),
  LambdaConfig: lambdaConfig,
 })
 if err != nil {
  log.Printf("Couldn't update user pool %v. Here's why: %v\n", userPoolId, err)
 }
 return err
}



// SignUp signs up a user with Amazon Cognito.
func (actor CognitoActions) SignUp(ctx context.Context, clientId string, userName
 string, password string, userEmail string) (bool, error) {
 confirmed := false
 output, err := actor.CognitoClient.SignUp(ctx,
 &cognitoidentityprovider.SignUpInput{
  ClientId: aws.String(clientId),
  Password: aws.String(password),
  Username: aws.String(userName),
  UserAttributes: []types.AttributeType{
   {Name: aws.String("email"), Value: aws.String(userEmail)},
  },
 })
 if err != nil {
  var invalidPassword *types.InvalidPasswordException
  if errors.As(err, &invalidPassword) {
   log.Println(*invalidPassword.Message)
  } else {
   log.Printf("Couldn't sign up user %v. Here's why: %v\n", userName, err)
  }
 } else {
  confirmed = output.UserConfirmed
 }
 return confirmed, err
}
```

```
// SignIn signs in a user to Amazon Cognito using a username and password
 authentication flow.
func (actor CognitoActions) SignIn(ctx context.Context, clientId string, userName
 string, password string) (*types.AuthenticationResultType, error) {
 var authResult *types.AuthenticationResultType
 output, err := actor.CognitoClient.InitiateAuth(ctx,
 &cognitoidentityprovider.InitiateAuthInput{
  AuthFlow:       "USER_PASSWORD_AUTH",
  ClientId:       aws.String(clientId),
  AuthParameters: map[string]string{"USERNAME": userName, "PASSWORD": password},
 })
 if err != nil {
  var resetRequired *types.PasswordResetRequiredException
  if errors.As(err, &resetRequired) {
   log.Println(*resetRequired.Message)
  } else {
   log.Printf("Couldn't sign in user %v. Here's why: %v\n", userName, err)
  }
 } else {
  authResult = output.AuthenticationResult
 }
 return authResult, err
}



// ForgotPassword starts a password recovery flow for a user. This flow typically
 sends a confirmation code
// to the user's configured notification destination, such as email.
func (actor CognitoActions) ForgotPassword(ctx context.Context, clientId string,
 userName string) (*types.CodeDeliveryDetailsType, error) {
 output, err := actor.CognitoClient.ForgotPassword(ctx,
 &cognitoidentityprovider.ForgotPasswordInput{
  ClientId: aws.String(clientId),
  Username: aws.String(userName),
 })
 if err != nil {
  log.Printf("Couldn't start password reset for user '%v'. Here;s why: %v\n",
 userName, err)
 }
 return output.CodeDeliveryDetails, err
}
```

```
// ConfirmForgotPassword confirms a user with a confirmation code and a new
 password.
func (actor CognitoActions) ConfirmForgotPassword(ctx context.Context, clientId
 string, code string, userName string, password string) error {
 _, err := actor.CognitoClient.ConfirmForgotPassword(ctx,
 &cognitoidentityprovider.ConfirmForgotPasswordInput{
  ClientId:         aws.String(clientId),
  ConfirmationCode: aws.String(code),
  Password:         aws.String(password),
  Username:         aws.String(userName),
 })
 if err != nil {
  var invalidPassword *types.InvalidPasswordException
  if errors.As(err, &invalidPassword) {
   log.Println(*invalidPassword.Message)
  } else {
   log.Printf("Couldn't confirm user %v. Here's why: %v", userName, err)
  }
 }
 return err
}



// DeleteUser removes a user from the user pool.
func (actor CognitoActions) DeleteUser(ctx context.Context, userAccessToken string)
 error {
 _, err := actor.CognitoClient.DeleteUser(ctx,
 &cognitoidentityprovider.DeleteUserInput{
  AccessToken: aws.String(userAccessToken),
 })
 if err != nil {
  log.Printf("Couldn't delete user. Here's why: %v\n", err)
 }
 return err
}



// AdminCreateUser uses administrator credentials to add a user to a user pool. This
 method leaves the user
// in a state that requires they enter a new password next time they sign in.
```

```
func (actor CognitoActions) AdminCreateUser(ctx context.Context, userPoolId string,
 userName string, userEmail string) error {
 _, err := actor.CognitoClient.AdminCreateUser(ctx,
 &cognitoidentityprovider.AdminCreateUserInput{
  UserPoolId:     aws.String(userPoolId),
  Username:       aws.String(userName),
  MessageAction:  types.MessageActionTypeSuppress,
  UserAttributes: []types.AttributeType{{Name: aws.String("email"), Value:
 aws.String(userEmail)}},
 })
 if err != nil {
  var userExists *types.UsernameExistsException
  if errors.As(err, &userExists) {
   log.Printf("User %v already exists in the user pool.", userName)
   err = nil
  } else {
   log.Printf("Couldn't create user %v. Here's why: %v\n", userName, err)
  }
 }
 return err
}



// AdminSetUserPassword uses administrator credentials to set a password for a user
 without requiring a
// temporary password.
func (actor CognitoActions) AdminSetUserPassword(ctx context.Context, userPoolId
 string, userName string, password string) error {
 _, err := actor.CognitoClient.AdminSetUserPassword(ctx,
 &cognitoidentityprovider.AdminSetUserPasswordInput{
  Password:   aws.String(password),
  UserPoolId: aws.String(userPoolId),
  Username:   aws.String(userName),
  Permanent:  true,
 })
 if err != nil {
  var invalidPassword *types.InvalidPasswordException
  if errors.As(err, &invalidPassword) {
   log.Println(*invalidPassword.Message)
  } else {
   log.Printf("Couldn't set password for user %v. Here's why: %v\n", userName, err)
  }
 }
```

```
 return err
}
```

Erstellen Sie eine Struktur, die DynamoDB-Aktionen umhüllt.

```
import (
 "context"
 "fmt"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// DynamoActions encapsulates the Amazon Simple Notification Service (Amazon SNS)
 actions
// used in the examples.
type DynamoActions struct {
 DynamoClient *dynamodb.Client
}

// User defines structured user data.
type User struct {
 UserName  string
 UserEmail string
 LastLogin *LoginInfo `dynamodbav:",omitempty"`
}

// LoginInfo defines structured custom login data.
type LoginInfo struct {
 UserPoolId string
 ClientId   string
 Time       string
}

// UserList defines a list of users.
type UserList struct {
 Users []User
```

```
}

// UserNameList returns the usernames contained in a UserList as a list of strings.
func (users *UserList) UserNameList() []string {
 names := make([]string, len(users.Users))
 for i := 0; i < len(users.Users); i++ {
  names[i] = users.Users[i].UserName
 }
 return names
}

// PopulateTable adds a set of test users to the table.
func (actor DynamoActions) PopulateTable(ctx context.Context, tableName string)
 error {
 var err error
 var item map[string]types.AttributeValue
 var writeReqs []types.WriteRequest
 for i := 1; i < 4; i++ {
  item, err = attributevalue.MarshalMap(User{UserName: fmt.Sprintf("test_user_%v",
 i), UserEmail: fmt.Sprintf("test_email_%v@example.com", i)})
  if err != nil {
   log.Printf("Couldn't marshall user into DynamoDB format. Here's why: %v\n", err)
   return err
  }
  writeReqs = append(writeReqs, types.WriteRequest{PutRequest:
 &types.PutRequest{Item: item}})
 }
 _, err = actor.DynamoClient.BatchWriteItem(ctx, &dynamodb.BatchWriteItemInput{
  RequestItems: map[string][]types.WriteRequest{tableName: writeReqs},
 })
 if err != nil {
  log.Printf("Couldn't populate table %v with users. Here's why: %v\n", tableName,
 err)
 }
 return err
}

// Scan scans the table for all items.
func (actor DynamoActions) Scan(ctx context.Context, tableName string) (UserList,
 error) {
 var userList UserList
 output, err := actor.DynamoClient.Scan(ctx, &dynamodb.ScanInput{
  TableName: aws.String(tableName),
 })
```

```
 if err != nil {
  log.Printf("Couldn't scan table %v for items. Here's why: %v\n", tableName, err)
 } else {
  err = attributevalue.UnmarshalListOfMaps(output.Items, &userList.Users)
  if err != nil {
   log.Printf("Couldn't unmarshal items into users. Here's why: %v\n", err)
  }
 }
 return userList, err
}

// AddUser adds a user item to a table.
func (actor DynamoActions) AddUser(ctx context.Context, tableName string, user User)
 error {
 userItem, err := attributevalue.MarshalMap(user)
 if err != nil {
  log.Printf("Couldn't marshall user to item. Here's why: %v\n", err)
 }
 _, err = actor.DynamoClient.PutItem(ctx, &dynamodb.PutItemInput{
  Item:      userItem,
  TableName: aws.String(tableName),
 })
 if err != nil {
  log.Printf("Couldn't put item in table %v. Here's why: %v", tableName, err)
 }
 return err
}
```

Erstellen Sie eine Struktur, die CloudWatch Logs-Aktionen umschließt.

```
import (
 "context"
 "fmt"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/cloudwatchlogs"
 "github.com/aws/aws-sdk-go-v2/service/cloudwatchlogs/types"
)
```

```
type CloudWatchLogsActions struct {
 CwlClient *cloudwatchlogs.Client
}

// GetLatestLogStream gets the most recent log stream for a Lambda function.
func (actor CloudWatchLogsActions) GetLatestLogStream(ctx context.Context,
 functionName string) (types.LogStream, error) {
 var logStream types.LogStream
 logGroupName := fmt.Sprintf("/aws/lambda/%s", functionName)
 output, err := actor.CwlClient.DescribeLogStreams(ctx,
 &cloudwatchlogs.DescribeLogStreamsInput{
  Descending:   aws.Bool(true),
  Limit:        aws.Int32(1),
  LogGroupName: aws.String(logGroupName),
  OrderBy:      types.OrderByLastEventTime,
 })
 if err != nil {
  log.Printf("Couldn't get log streams for log group %v. Here's why: %v\n",
 logGroupName, err)
 } else {
  logStream = output.LogStreams[0]
 }
 return logStream, err
}

// GetLogEvents gets the most recent eventCount events from the specified log
 stream.
func (actor CloudWatchLogsActions) GetLogEvents(ctx context.Context, functionName
 string, logStreamName string, eventCount int32) (
 []types.OutputLogEvent, error) {
 var events []types.OutputLogEvent
 logGroupName := fmt.Sprintf("/aws/lambda/%s", functionName)
 output, err := actor.CwlClient.GetLogEvents(ctx, &cloudwatchlogs.GetLogEventsInput{
  LogStreamName: aws.String(logStreamName),
  Limit:         aws.Int32(eventCount),
  LogGroupName:  aws.String(logGroupName),
 })
 if err != nil {
  log.Printf("Couldn't get log event for log stream %v. Here's why: %v\n",
 logStreamName, err)
 } else {
  events = output.Events
 }
 return events, err
```

```
}
```

Erstellen Sie eine Struktur, die Aktionen umschließt. AWS CloudFormation

```
import (
 "context"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/cloudformation"
)

// StackOutputs defines a map of outputs from a specific stack.
type StackOutputs map[string]string

type CloudFormationActions struct {
 CfnClient *cloudformation.Client
}

// GetOutputs gets the outputs from a CloudFormation stack and puts them into a
 structured format.
func (actor CloudFormationActions) GetOutputs(ctx context.Context, stackName string)
 StackOutputs {
 output, err := actor.CfnClient.DescribeStacks(ctx,
 &cloudformation.DescribeStacksInput{
  StackName: aws.String(stackName),
 })
 if err != nil || len(output.Stacks) == 0 {
  log.Panicf("Couldn't find a CloudFormation stack named %v. Here's why: %v\n",
 stackName, err)
 }
 stackOutputs := StackOutputs{}
 for _, out := range output.Stacks[0].Outputs {
  stackOutputs[*out.OutputKey] = *out.OutputValue
 }
 return stackOutputs
}
```

Bereinigen von Ressourcen.

```
import (
 "context"
 "log"
 "user_pools_and_lambda_triggers/actions"

 "github.com/awsdocs/aws-doc-sdk-examples/gov2/demotools"
)

// Resources keeps track of AWS resources created during an example and handles
// cleanup when the example finishes.
type Resources struct {
 userPoolId       string
 userAccessTokens []string
 triggers         []actions.Trigger

 cognitoActor *actions.CognitoActions
 questioner   demotools.IQuestioner
}

func (resources *Resources) init(cognitoActor *actions.CognitoActions, questioner
 demotools.IQuestioner) {
 resources.userAccessTokens = []string{}
 resources.triggers = []actions.Trigger{}
 resources.cognitoActor = cognitoActor
 resources.questioner = questioner
}

// Cleanup deletes all AWS resources created during an example.
func (resources *Resources) Cleanup(ctx context.Context) {
 defer func() {
  if r := recover(); r != nil {
   log.Printf("Something went wrong during cleanup.\n%v\n", r)
   log.Println("Use the AWS Management Console to remove any remaining resources \n"
 +
    "that were created for this scenario.")
  }
 }()

 wantDelete := resources.questioner.AskBool("Do you want to remove all of the AWS
 resources that were created "+
  "during this demo (y/n)?", "y")
```

```
 if wantDelete {
  for _, accessToken := range resources.userAccessTokens {
   err := resources.cognitoActor.DeleteUser(ctx, accessToken)
   if err != nil {
    log.Println("Couldn't delete user during cleanup.")
    panic(err)
   }
   log.Println("Deleted user.")
  }
  triggerList := make([]actions.TriggerInfo, len(resources.triggers))
  for i := 0; i < len(resources.triggers); i++ {
   triggerList[i] = actions.TriggerInfo{Trigger: resources.triggers[i], HandlerArn:
 nil}
  }
  err := resources.cognitoActor.UpdateTriggers(ctx, resources.userPoolId,
 triggerList...)
  if err != nil {
   log.Println("Couldn't update Cognito triggers during cleanup.")
   panic(err)
  }
  log.Println("Removed Cognito triggers from user pool.")
 } else {
  log.Println("Be sure to remove resources when you're done with them to avoid
 unexpected charges!")
 }
}
```

- API-Details finden Sie in den folgenden Themen der AWS SDK für Go -API-Referenz.
  - DeleteUser
  - InitiateAuth
  - SignUp
  - UpdateUserPool

Migrieren Sie automatisch bekannte Benutzer mit einer Lambda-Funktion

Das folgende Codebeispiel zeigt, wie bekannte Amazon Cognito-Benutzer mit einer Lambda-Funktion automatisch migriert werden können.

- Konfigurieren Sie einen Benutzerpool, um eine Lambda-Funktion für den `MigrateUser`-Trigger aufzurufen.
- Melden Sie sich bei Amazon Cognito mit einem Benutzernamen und einer E-Mail-Adresse an, die sich nicht im Benutzerpool befinden.
- Die Lambda-Funktion durchsucht eine DynamoDB-Tabelle und migriert bekannte Benutzer automatisch in den Benutzerpool.
- Führen Sie den Ablauf „Passwort vergessen“ aus, um das Passwort für den migrierten Benutzer zurückzusetzen.
- Melden Sie sich als neuer Benutzer an und bereinigen Sie dann die Ressourcen.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Führen Sie ein interaktives Szenario an einem Prompt aus.

```
import (
 "context"
 "errors"
 "fmt"
 "log"
 "strings"
 "user_pools_and_lambda_triggers/actions"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/cognitoidentityprovider"
 "github.com/aws/aws-sdk-go-v2/service/cognitoidentityprovider/types"
 "github.com/awsdocs/aws-doc-sdk-examples/gov2/demotools"
)

// MigrateUser separates the steps of this scenario into individual functions so
 that
// they are simpler to read and understand.
type MigrateUser struct {
```

```
 helper       IScenarioHelper
 questioner   demotools.IQuestioner
 resources    Resources
 cognitoActor *actions.CognitoActions
}

// NewMigrateUser constructs a new migrate user runner.
func NewMigrateUser(sdkConfig aws.Config, questioner demotools.IQuestioner, helper
 IScenarioHelper) MigrateUser {
 scenario := MigrateUser{
  helper:       helper,
  questioner:   questioner,
  resources:    Resources{},
  cognitoActor: &actions.CognitoActions{CognitoClient:
 cognitoidentityprovider.NewFromConfig(sdkConfig)},
 }
 scenario.resources.init(scenario.cognitoActor, questioner)
 return scenario
}

// AddMigrateUserTrigger adds a Lambda handler as an invocation target for the
 MigrateUser trigger.
func (runner *MigrateUser) AddMigrateUserTrigger(ctx context.Context, userPoolId
 string, functionArn string) {
 log.Printf("Let's add a Lambda function to handle the MigrateUser trigger from
 Cognito.\n" +
  "This trigger happens when an unknown user signs in, and lets your function take
 action before Cognito\n" +
  "rejects the user.\n\n")
 err := runner.cognitoActor.UpdateTriggers(
  ctx, userPoolId,
  actions.TriggerInfo{Trigger: actions.UserMigration, HandlerArn:
 aws.String(functionArn)})
 if err != nil {
  panic(err)
 }
 log.Printf("Lambda function %v added to user pool %v to handle the MigrateUser
 trigger.\n",
  functionArn, userPoolId)

 log.Println(strings.Repeat("-", 88))
}
```

```
// SignInUser adds a new user to the known users table and signs that user in to
 Amazon Cognito.
func (runner *MigrateUser) SignInUser(ctx context.Context, usersTable string,
 clientId string) (bool, actions.User) {
 log.Println("Let's sign in a user to your Cognito user pool. When the username and
 email matches an entry in the\n" +
  "DynamoDB known users table, the email is automatically verified and the user is
 migrated to the Cognito user pool.")

 user := actions.User{}
 user.UserName = runner.questioner.Ask("\nEnter a username:")
 user.UserEmail = runner.questioner.Ask("\nEnter an email that you own. This email
 will be used to confirm user migration\n" +
  "during this example:")

 runner.helper.AddKnownUser(ctx, usersTable, user)

 var err error
 var resetRequired *types.PasswordResetRequiredException
 var authResult *types.AuthenticationResultType
 signedIn := false
 for !signedIn && resetRequired == nil {
  log.Printf("Signing in to Cognito as user '%v'. The expected result is a
 PasswordResetRequiredException.\n\n", user.UserName)
  authResult, err = runner.cognitoActor.SignIn(ctx, clientId, user.UserName, "_")
  if err != nil {
   if errors.As(err, &resetRequired) {
    log.Printf("\nUser '%v' is not in the Cognito user pool but was found in the
 DynamoDB known users table.\n"+
     "User migration is started and a password reset is required.", user.UserName)
   } else {
    panic(err)
   }
  } else {
   log.Printf("User '%v' successfully signed in. This is unexpected and probably
 means you have not\n"+
    "cleaned up a previous run of this scenario, so the user exist in the Cognito
 user pool.\n"+
    "You can continue this example and select to clean up resources, or manually
 remove\n"+
    "the user from your user pool and try again.", user.UserName)
   runner.resources.userAccessTokens = append(runner.resources.userAccessTokens,
 *authResult.AccessToken)
   signedIn = true
```

```
  }
 }

 log.Println(strings.Repeat("-", 88))
 return resetRequired != nil, user
}

// ResetPassword starts a password recovery flow.
func (runner *MigrateUser) ResetPassword(ctx context.Context, clientId string, user
 actions.User) {
 wantCode := runner.questioner.AskBool(fmt.Sprintf("In order to migrate the user to
 Cognito, you must be able to receive a confirmation\n"+
  "code by email at %v. Do you want to send a code (y/n)?", user.UserEmail), "y")
 if !wantCode {
  log.Println("To complete this example and successfully migrate a user to Cognito,
 you must enter an email\n" +
   "you own that can receive a confirmation code.")
  return
 }
 codeDelivery, err := runner.cognitoActor.ForgotPassword(ctx, clientId,
 user.UserName)
 if err != nil {
  panic(err)
 }
 log.Printf("\nA confirmation code has been sent to %v.", *codeDelivery.Destination)
 code := runner.questioner.Ask("Check your email and enter it here:")

 confirmed := false
 password := runner.questioner.AskPassword("\nEnter a password that has at least
 eight characters, uppercase, lowercase, numbers and symbols.\n"+
  "(the password will not display as you type):", 8)
 for !confirmed {
  log.Printf("\nConfirming password reset for user '%v'.\n", user.UserName)
  err = runner.cognitoActor.ConfirmForgotPassword(ctx, clientId, code,
 user.UserName, password)
  if err != nil {
   var invalidPassword *types.InvalidPasswordException
   if errors.As(err, &invalidPassword) {
    password = runner.questioner.AskPassword("\nEnter another password:", 8)
   } else {
    panic(err)
   }
  } else {
   confirmed = true
```

```
  }
 }
 log.Printf("User '%v' successfully confirmed and migrated.\n", user.UserName)
 log.Println("Signing in with your username and password...")
 authResult, err := runner.cognitoActor.SignIn(ctx, clientId, user.UserName,
 password)
 if err != nil {
  panic(err)
 }
 log.Printf("Successfully signed in. Your access token starts with: %v...\n",
 (*authResult.AccessToken)[:10])
 runner.resources.userAccessTokens = append(runner.resources.userAccessTokens,
 *authResult.AccessToken)

 log.Println(strings.Repeat("-", 88))
}

// Run runs the scenario.
func (runner *MigrateUser) Run(ctx context.Context, stackName string) {
 defer func() {
  if r := recover(); r != nil {
   log.Println("Something went wrong with the demo.")
   runner.resources.Cleanup(ctx)
  }
 }()

 log.Println(strings.Repeat("-", 88))
 log.Printf("Welcome\n")

 log.Println(strings.Repeat("-", 88))

 stackOutputs, err := runner.helper.GetStackOutputs(ctx, stackName)
 if err != nil {
  panic(err)
 }
 runner.resources.userPoolId = stackOutputs["UserPoolId"]

 runner.AddMigrateUserTrigger(ctx, stackOutputs["UserPoolId"],
 stackOutputs["MigrateUserFunctionArn"])
 runner.resources.triggers = append(runner.resources.triggers,
 actions.UserMigration)
 resetNeeded, user := runner.SignInUser(ctx, stackOutputs["TableName"],
 stackOutputs["UserPoolClientId"])
 if resetNeeded {
```

```
  runner.helper.ListRecentLogEvents(ctx, stackOutputs["MigrateUserFunction"])
  runner.ResetPassword(ctx, stackOutputs["UserPoolClientId"], user)
 }

 runner.resources.Cleanup(ctx)

 log.Println(strings.Repeat("-", 88))
 log.Println("Thanks for watching!")
 log.Println(strings.Repeat("-", 88))
}
```

Behandeln Sie den `MigrateUser`-Trigger mit einer Lambda-Funktion.

```
import (
 "context"
 "log"
 "os"

 "github.com/aws/aws-lambda-go/events"
 "github.com/aws/aws-lambda-go/lambda"
 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/config"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/expression"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
)

const TABLE_NAME = "TABLE_NAME"

// UserInfo defines structured user data that can be marshalled to a DynamoDB
 format.
type UserInfo struct {
 UserName  string `dynamodbav:"UserName"`
 UserEmail string `dynamodbav:"UserEmail"`
}

type handler struct {
 dynamoClient *dynamodb.Client
}
```

```
// HandleRequest handles the MigrateUser event by looking up a user in an Amazon
 DynamoDB table and
// specifying whether they should be migrated to the user pool.
func (h *handler) HandleRequest(ctx context.Context, event
 events.CognitoEventUserPoolsMigrateUser) (events.CognitoEventUserPoolsMigrateUser,
 error) {
 log.Printf("Received migrate trigger from %v for user '%v'", event.TriggerSource,
 event.UserName)
 if event.TriggerSource != "UserMigration_Authentication" {
  return event, nil
 }
 tableName := os.Getenv(TABLE_NAME)
 user := UserInfo{
  UserName: event.UserName,
 }
 log.Printf("Looking up user '%v' in table %v.\n", user.UserName, tableName)
 filterEx := expression.Name("UserName").Equal(expression.Value(user.UserName))
 expr, err := expression.NewBuilder().WithFilter(filterEx).Build()
 if err != nil {
  log.Printf("Error building expression to query for user '%v'.\n", user.UserName)
  return event, err
 }
 output, err := h.dynamoClient.Scan(ctx, &dynamodb.ScanInput{
  TableName:                 aws.String(tableName),
  FilterExpression:          expr.Filter(),
  ExpressionAttributeNames:  expr.Names(),
  ExpressionAttributeValues: expr.Values(),
 })
 if err != nil {
  log.Printf("Error looking up user '%v'.\n", user.UserName)
  return event, err
 }
 if len(output.Items) == 0 {
  log.Printf("User '%v' not found, not migrating user.\n", user.UserName)
  return event, err
 }

 var users []UserInfo
 err = attributevalue.UnmarshalListOfMaps(output.Items, &users)
 if err != nil {
  log.Printf("Couldn't unmarshal DynamoDB items. Here's why: %v\n", err)
  return event, err
 }
```

```
 user = users[0]
 log.Printf("UserName '%v' found with email %v. User is migrated and must reset
 password.\n", user.UserName, user.UserEmail)
 event.CognitoEventUserPoolsMigrateUserResponse.UserAttributes = map[string]string{
  "email":          user.UserEmail,
  "email_verified": "true", // email_verified is required for the forgot password
 flow.
 }
 event.CognitoEventUserPoolsMigrateUserResponse.FinalUserStatus = "RESET_REQUIRED"
 event.CognitoEventUserPoolsMigrateUserResponse.MessageAction = "SUPPRESS"

 return event, err
}

func main() {
 ctx := context.Background()
 sdkConfig, err := config.LoadDefaultConfig(ctx)
 if err != nil {
  log.Panicln(err)
 }
 h := handler{
  dynamoClient: dynamodb.NewFromConfig(sdkConfig),
 }
 lambda.Start(h.HandleRequest)
}
```

Erstellen Sie eine Struktur, die allgemeine Aufgaben ausführt.

```
import (
 "context"
 "log"
 "strings"
 "time"
 "user_pools_and_lambda_triggers/actions"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/cloudformation"
 "github.com/aws/aws-sdk-go-v2/service/cloudwatchlogs"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/awsdocs/aws-doc-sdk-examples/gov2/demotools"
```

```
)

// IScenarioHelper defines common functions used by the workflows in this example.
type IScenarioHelper interface {
 Pause(secs int)
 GetStackOutputs(ctx context.Context, stackName string) (actions.StackOutputs,
 error)
 PopulateUserTable(ctx context.Context, tableName string)
 GetKnownUsers(ctx context.Context, tableName string) (actions.UserList, error)
 AddKnownUser(ctx context.Context, tableName string, user actions.User)
 ListRecentLogEvents(ctx context.Context, functionName string)
}

// ScenarioHelper contains AWS wrapper structs used by the workflows in this
 example.
type ScenarioHelper struct {
 questioner  demotools.IQuestioner
 dynamoActor *actions.DynamoActions
 cfnActor    *actions.CloudFormationActions
 cwlActor    *actions.CloudWatchLogsActions
 isTestRun   bool
}

// NewScenarioHelper constructs a new scenario helper.
func NewScenarioHelper(sdkConfig aws.Config, questioner demotools.IQuestioner)
 ScenarioHelper {
 scenario := ScenarioHelper{
  questioner:  questioner,
  dynamoActor: &actions.DynamoActions{DynamoClient:
 dynamodb.NewFromConfig(sdkConfig)},
  cfnActor:    &actions.CloudFormationActions{CfnClient:
 cloudformation.NewFromConfig(sdkConfig)},
  cwlActor:    &actions.CloudWatchLogsActions{CwlClient:
 cloudwatchlogs.NewFromConfig(sdkConfig)},
 }
 return scenario
}

// Pause waits for the specified number of seconds.
func (helper ScenarioHelper) Pause(secs int) {
 if !helper.isTestRun {
  time.Sleep(time.Duration(secs) * time.Second)
 }
}
```

```
// GetStackOutputs gets the outputs from the specified CloudFormation stack in a
 structured format.
func (helper ScenarioHelper) GetStackOutputs(ctx context.Context, stackName string)
 (actions.StackOutputs, error) {
 return helper.cfnActor.GetOutputs(ctx, stackName), nil
}

// PopulateUserTable fills the known user table with example data.
func (helper ScenarioHelper) PopulateUserTable(ctx context.Context, tableName
 string) {
 log.Printf("First, let's add some users to the DynamoDB %v table we'll use for this
 example.\n", tableName)
 err := helper.dynamoActor.PopulateTable(ctx, tableName)
 if err != nil {
  panic(err)
 }
}

// GetKnownUsers gets the users from the known users table in a structured format.
func (helper ScenarioHelper) GetKnownUsers(ctx context.Context, tableName string)
 (actions.UserList, error) {
 knownUsers, err := helper.dynamoActor.Scan(ctx, tableName)
 if err != nil {
  log.Printf("Couldn't get known users from table %v. Here's why: %v\n", tableName,
 err)
 }
 return knownUsers, err
}

// AddKnownUser adds a user to the known users table.
func (helper ScenarioHelper) AddKnownUser(ctx context.Context, tableName string,
 user actions.User) {
 log.Printf("Adding user '%v' with email '%v' to the DynamoDB known users table...
\n",
  user.UserName, user.UserEmail)
 err := helper.dynamoActor.AddUser(ctx, tableName, user)
 if err != nil {
  panic(err)
 }
}

// ListRecentLogEvents gets the most recent log stream and events for the specified
 Lambda function and displays them.
```

```
func (helper ScenarioHelper) ListRecentLogEvents(ctx context.Context, functionName
 string) {
 log.Println("Waiting a few seconds to let Lambda write to CloudWatch Logs...")
 helper.Pause(10)
 log.Println("Okay, let's check the logs to find what's happened recently with your
 Lambda function.")
 logStream, err := helper.cwlActor.GetLatestLogStream(ctx, functionName)
 if err != nil {
  panic(err)
 }
 log.Printf("Getting some recent events from log stream %v\n",
 *logStream.LogStreamName)
 events, err := helper.cwlActor.GetLogEvents(ctx, functionName,
 *logStream.LogStreamName, 10)
 if err != nil {
  panic(err)
 }
 for _, event := range events {
  log.Printf("\t%v", *event.Message)
 }
 log.Println(strings.Repeat("-", 88))
}
```

Erstellen Sie eine Struktur, die Amazon Cognito-Aktionen umschließt.

```
import (
 "context"
 "errors"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/cognitoidentityprovider"
 "github.com/aws/aws-sdk-go-v2/service/cognitoidentityprovider/types"
)

type CognitoActions struct {
 CognitoClient *cognitoidentityprovider.Client
}
```

```
// Trigger and TriggerInfo define typed data for updating an Amazon Cognito trigger.
type Trigger int

const (
 PreSignUp Trigger = iota
 UserMigration
 PostAuthentication
)

type TriggerInfo struct {
 Trigger    Trigger
 HandlerArn *string
}

// UpdateTriggers adds or removes Lambda triggers for a user pool. When a trigger is
 specified with a `nil` value,
// it is removed from the user pool.
func (actor CognitoActions) UpdateTriggers(ctx context.Context, userPoolId string,
 triggers ...TriggerInfo) error {
 output, err := actor.CognitoClient.DescribeUserPool(ctx,
 &cognitoidentityprovider.DescribeUserPoolInput{
  UserPoolId: aws.String(userPoolId),
 })
 if err != nil {
  log.Printf("Couldn't get info about user pool %v. Here's why: %v\n", userPoolId,
 err)
  return err
 }
 lambdaConfig := output.UserPool.LambdaConfig
 for _, trigger := range triggers {
  switch trigger.Trigger {
  case PreSignUp:
   lambdaConfig.PreSignUp = trigger.HandlerArn
  case UserMigration:
   lambdaConfig.UserMigration = trigger.HandlerArn
  case PostAuthentication:
   lambdaConfig.PostAuthentication = trigger.HandlerArn
  }
 }
 _, err = actor.CognitoClient.UpdateUserPool(ctx,
 &cognitoidentityprovider.UpdateUserPoolInput{
  UserPoolId:   aws.String(userPoolId),
  LambdaConfig: lambdaConfig,
```

```
 })
 if err != nil {
  log.Printf("Couldn't update user pool %v. Here's why: %v\n", userPoolId, err)
 }
 return err
}



// SignUp signs up a user with Amazon Cognito.
func (actor CognitoActions) SignUp(ctx context.Context, clientId string, userName
 string, password string, userEmail string) (bool, error) {
 confirmed := false
 output, err := actor.CognitoClient.SignUp(ctx,
 &cognitoidentityprovider.SignUpInput{
  ClientId: aws.String(clientId),
  Password: aws.String(password),
  Username: aws.String(userName),
  UserAttributes: []types.AttributeType{
   {Name: aws.String("email"), Value: aws.String(userEmail)},
  },
 })
 if err != nil {
  var invalidPassword *types.InvalidPasswordException
  if errors.As(err, &invalidPassword) {
   log.Println(*invalidPassword.Message)
  } else {
   log.Printf("Couldn't sign up user %v. Here's why: %v\n", userName, err)
  }
 } else {
  confirmed = output.UserConfirmed
 }
 return confirmed, err
}



// SignIn signs in a user to Amazon Cognito using a username and password
 authentication flow.
func (actor CognitoActions) SignIn(ctx context.Context, clientId string, userName
 string, password string) (*types.AuthenticationResultType, error) {
 var authResult *types.AuthenticationResultType
 output, err := actor.CognitoClient.InitiateAuth(ctx,
 &cognitoidentityprovider.InitiateAuthInput{
```

```
  AuthFlow:       "USER_PASSWORD_AUTH",
  ClientId:       aws.String(clientId),
  AuthParameters: map[string]string{"USERNAME": userName, "PASSWORD": password},
 })
 if err != nil {
  var resetRequired *types.PasswordResetRequiredException
  if errors.As(err, &resetRequired) {
   log.Println(*resetRequired.Message)
  } else {
   log.Printf("Couldn't sign in user %v. Here's why: %v\n", userName, err)
  }
 } else {
  authResult = output.AuthenticationResult
 }
 return authResult, err
}



// ForgotPassword starts a password recovery flow for a user. This flow typically
 sends a confirmation code
// to the user's configured notification destination, such as email.
func (actor CognitoActions) ForgotPassword(ctx context.Context, clientId string,
 userName string) (*types.CodeDeliveryDetailsType, error) {
 output, err := actor.CognitoClient.ForgotPassword(ctx,
 &cognitoidentityprovider.ForgotPasswordInput{
  ClientId: aws.String(clientId),
  Username: aws.String(userName),
 })
 if err != nil {
  log.Printf("Couldn't start password reset for user '%v'. Here;s why: %v\n",
 userName, err)
 }
 return output.CodeDeliveryDetails, err
}



// ConfirmForgotPassword confirms a user with a confirmation code and a new
 password.
func (actor CognitoActions) ConfirmForgotPassword(ctx context.Context, clientId
 string, code string, userName string, password string) error {
 _, err := actor.CognitoClient.ConfirmForgotPassword(ctx,
 &cognitoidentityprovider.ConfirmForgotPasswordInput{
```

```
  ClientId:         aws.String(clientId),
  ConfirmationCode: aws.String(code),
  Password:         aws.String(password),
  Username:         aws.String(userName),
 })
 if err != nil {
  var invalidPassword *types.InvalidPasswordException
  if errors.As(err, &invalidPassword) {
   log.Println(*invalidPassword.Message)
  } else {
   log.Printf("Couldn't confirm user %v. Here's why: %v", userName, err)
  }
 }
 return err
}



// DeleteUser removes a user from the user pool.
func (actor CognitoActions) DeleteUser(ctx context.Context, userAccessToken string)
 error {
 _, err := actor.CognitoClient.DeleteUser(ctx,
 &cognitoidentityprovider.DeleteUserInput{
  AccessToken: aws.String(userAccessToken),
 })
 if err != nil {
  log.Printf("Couldn't delete user. Here's why: %v\n", err)
 }
 return err
}



// AdminCreateUser uses administrator credentials to add a user to a user pool. This
 method leaves the user
// in a state that requires they enter a new password next time they sign in.
func (actor CognitoActions) AdminCreateUser(ctx context.Context, userPoolId string,
 userName string, userEmail string) error {
 _, err := actor.CognitoClient.AdminCreateUser(ctx,
 &cognitoidentityprovider.AdminCreateUserInput{
  UserPoolId:     aws.String(userPoolId),
  Username:       aws.String(userName),
  MessageAction:  types.MessageActionTypeSuppress,
```

```
  UserAttributes: []types.AttributeType{{Name: aws.String("email"), Value:
 aws.String(userEmail)}},
 })
 if err != nil {
  var userExists *types.UsernameExistsException
  if errors.As(err, &userExists) {
   log.Printf("User %v already exists in the user pool.", userName)
   err = nil
  } else {
   log.Printf("Couldn't create user %v. Here's why: %v\n", userName, err)
  }
 }
 return err
}



// AdminSetUserPassword uses administrator credentials to set a password for a user
 without requiring a
// temporary password.
func (actor CognitoActions) AdminSetUserPassword(ctx context.Context, userPoolId
 string, userName string, password string) error {
 _, err := actor.CognitoClient.AdminSetUserPassword(ctx,
 &cognitoidentityprovider.AdminSetUserPasswordInput{
  Password:   aws.String(password),
  UserPoolId: aws.String(userPoolId),
  Username:   aws.String(userName),
  Permanent:  true,
 })
 if err != nil {
  var invalidPassword *types.InvalidPasswordException
  if errors.As(err, &invalidPassword) {
   log.Println(*invalidPassword.Message)
  } else {
   log.Printf("Couldn't set password for user %v. Here's why: %v\n", userName, err)
  }
 }
 return err
}
```

Erstellen Sie eine Struktur, die DynamoDB-Aktionen umhüllt.

```
import (
 "context"
 "fmt"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// DynamoActions encapsulates the Amazon Simple Notification Service (Amazon SNS)
 actions
// used in the examples.
type DynamoActions struct {
 DynamoClient *dynamodb.Client
}

// User defines structured user data.
type User struct {
 UserName  string
 UserEmail string
 LastLogin *LoginInfo `dynamodbav:",omitempty"`
}

// LoginInfo defines structured custom login data.
type LoginInfo struct {
 UserPoolId string
 ClientId   string
 Time       string
}

// UserList defines a list of users.
type UserList struct {
 Users []User
}

// UserNameList returns the usernames contained in a UserList as a list of strings.
func (users *UserList) UserNameList() []string {
 names := make([]string, len(users.Users))
 for i := 0; i < len(users.Users); i++ {
  names[i] = users.Users[i].UserName
```

```
 }
 return names
}

// PopulateTable adds a set of test users to the table.
func (actor DynamoActions) PopulateTable(ctx context.Context, tableName string)
 error {
 var err error
 var item map[string]types.AttributeValue
 var writeReqs []types.WriteRequest
 for i := 1; i < 4; i++ {
  item, err = attributevalue.MarshalMap(User{UserName: fmt.Sprintf("test_user_%v",
 i), UserEmail: fmt.Sprintf("test_email_%v@example.com", i)})
  if err != nil {
   log.Printf("Couldn't marshall user into DynamoDB format. Here's why: %v\n", err)
   return err
  }
  writeReqs = append(writeReqs, types.WriteRequest{PutRequest:
 &types.PutRequest{Item: item}})
 }
 _, err = actor.DynamoClient.BatchWriteItem(ctx, &dynamodb.BatchWriteItemInput{
  RequestItems: map[string][]types.WriteRequest{tableName: writeReqs},
 })
 if err != nil {
  log.Printf("Couldn't populate table %v with users. Here's why: %v\n", tableName,
 err)
 }
 return err
}

// Scan scans the table for all items.
func (actor DynamoActions) Scan(ctx context.Context, tableName string) (UserList,
 error) {
 var userList UserList
 output, err := actor.DynamoClient.Scan(ctx, &dynamodb.ScanInput{
  TableName: aws.String(tableName),
 })
 if err != nil {
  log.Printf("Couldn't scan table %v for items. Here's why: %v\n", tableName, err)
 } else {
  err = attributevalue.UnmarshalListOfMaps(output.Items, &userList.Users)
  if err != nil {
   log.Printf("Couldn't unmarshal items into users. Here's why: %v\n", err)
  }
```

```
 }
 return userList, err
}

// AddUser adds a user item to a table.
func (actor DynamoActions) AddUser(ctx context.Context, tableName string, user User)
 error {
 userItem, err := attributevalue.MarshalMap(user)
 if err != nil {
  log.Printf("Couldn't marshall user to item. Here's why: %v\n", err)
 }
 _, err = actor.DynamoClient.PutItem(ctx, &dynamodb.PutItemInput{
  Item:      userItem,
  TableName: aws.String(tableName),
 })
 if err != nil {
  log.Printf("Couldn't put item in table %v. Here's why: %v", tableName, err)
 }
 return err
}
```

Erstellen Sie eine Struktur, die CloudWatch Logs-Aktionen umschließt.

```
import (
 "context"
 "fmt"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/cloudwatchlogs"
 "github.com/aws/aws-sdk-go-v2/service/cloudwatchlogs/types"
)

type CloudWatchLogsActions struct {
 CwlClient *cloudwatchlogs.Client
}

// GetLatestLogStream gets the most recent log stream for a Lambda function.
func (actor CloudWatchLogsActions) GetLatestLogStream(ctx context.Context,
 functionName string) (types.LogStream, error) {
```

```
 var logStream types.LogStream
 logGroupName := fmt.Sprintf("/aws/lambda/%s", functionName)
 output, err := actor.CwlClient.DescribeLogStreams(ctx,
 &cloudwatchlogs.DescribeLogStreamsInput{
  Descending:   aws.Bool(true),
  Limit:        aws.Int32(1),
  LogGroupName: aws.String(logGroupName),
  OrderBy:      types.OrderByLastEventTime,
 })
 if err != nil {
  log.Printf("Couldn't get log streams for log group %v. Here's why: %v\n",
 logGroupName, err)
 } else {
  logStream = output.LogStreams[0]
 }
 return logStream, err
}

// GetLogEvents gets the most recent eventCount events from the specified log
 stream.
func (actor CloudWatchLogsActions) GetLogEvents(ctx context.Context, functionName
 string, logStreamName string, eventCount int32) (
 []types.OutputLogEvent, error) {
 var events []types.OutputLogEvent
 logGroupName := fmt.Sprintf("/aws/lambda/%s", functionName)
 output, err := actor.CwlClient.GetLogEvents(ctx, &cloudwatchlogs.GetLogEventsInput{
  LogStreamName: aws.String(logStreamName),
  Limit:         aws.Int32(eventCount),
  LogGroupName:  aws.String(logGroupName),
 })
 if err != nil {
  log.Printf("Couldn't get log event for log stream %v. Here's why: %v\n",
 logStreamName, err)
 } else {
  events = output.Events
 }
 return events, err
}
```

Erstellen Sie eine Struktur, die Aktionen umschließt. AWS CloudFormation

```
import (
 "context"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/cloudformation"
)

// StackOutputs defines a map of outputs from a specific stack.
type StackOutputs map[string]string

type CloudFormationActions struct {
 CfnClient *cloudformation.Client
}

// GetOutputs gets the outputs from a CloudFormation stack and puts them into a
 structured format.
func (actor CloudFormationActions) GetOutputs(ctx context.Context, stackName string)
 StackOutputs {
 output, err := actor.CfnClient.DescribeStacks(ctx,
 &cloudformation.DescribeStacksInput{
  StackName: aws.String(stackName),
 })
 if err != nil || len(output.Stacks) == 0 {
  log.Panicf("Couldn't find a CloudFormation stack named %v. Here's why: %v\n",
 stackName, err)
 }
 stackOutputs := StackOutputs{}
 for _, out := range output.Stacks[0].Outputs {
  stackOutputs[*out.OutputKey] = *out.OutputValue
 }
 return stackOutputs
}
```

Bereinigen von Ressourcen.

```
import (
 "context"
 "log"
```

```
 "user_pools_and_lambda_triggers/actions"

 "github.com/awsdocs/aws-doc-sdk-examples/gov2/demotools"
)

// Resources keeps track of AWS resources created during an example and handles
// cleanup when the example finishes.
type Resources struct {
 userPoolId       string
 userAccessTokens []string
 triggers         []actions.Trigger

 cognitoActor *actions.CognitoActions
 questioner   demotools.IQuestioner
}

func (resources *Resources) init(cognitoActor *actions.CognitoActions, questioner
 demotools.IQuestioner) {
 resources.userAccessTokens = []string{}
 resources.triggers = []actions.Trigger{}
 resources.cognitoActor = cognitoActor
 resources.questioner = questioner
}

// Cleanup deletes all AWS resources created during an example.
func (resources *Resources) Cleanup(ctx context.Context) {
 defer func() {
  if r := recover(); r != nil {
   log.Printf("Something went wrong during cleanup.\n%v\n", r)
   log.Println("Use the AWS Management Console to remove any remaining resources \n"
 +
    "that were created for this scenario.")
  }
 }()

 wantDelete := resources.questioner.AskBool("Do you want to remove all of the AWS
 resources that were created "+
  "during this demo (y/n)?", "y")
 if wantDelete {
  for _, accessToken := range resources.userAccessTokens {
   err := resources.cognitoActor.DeleteUser(ctx, accessToken)
   if err != nil {
    log.Println("Couldn't delete user during cleanup.")
    panic(err)
```

```
   }
   log.Println("Deleted user.")
  }
  triggerList := make([]actions.TriggerInfo, len(resources.triggers))
  for i := 0; i < len(resources.triggers); i++ {
   triggerList[i] = actions.TriggerInfo{Trigger: resources.triggers[i], HandlerArn:
 nil}
  }
  err := resources.cognitoActor.UpdateTriggers(ctx, resources.userPoolId,
 triggerList...)
  if err != nil {
   log.Println("Couldn't update Cognito triggers during cleanup.")
   panic(err)
  }
  log.Println("Removed Cognito triggers from user pool.")
 } else {
  log.Println("Be sure to remove resources when you're done with them to avoid
 unexpected charges!")
 }
}
```

- API-Details finden Sie in den folgenden Themen der AWS SDK für Go -API-Referenz.
  - ConfirmForgotPassword
  - DeleteUser
  - ForgotPassword
  - InitiateAuth
  - SignUp
  - UpdateUserPool

Schreiben von benutzerdefinierten Aktivitätsdaten mit einer Lambda-Funktion nach der Amazon Cognito-Benutzerauthentifizierung

Das folgende Codebeispiel veranschaulicht, wie benutzerdefinierte Aktivitätsdaten mit einer Lambda-Funktion nach der Amazon-Cognito-Benutzerauthentifizierung geschrieben werden.

- Verwenden Sie Administratorfunktionen, um einen Benutzer zu einem Benutzerpool hinzuzufügen.

- Konfigurieren Sie einen Benutzerpool, um eine Lambda-Funktion für den `PostAuthentication`-Trigger aufzurufen.
- Melden Sie den neuen Benutzer bei Amazon Cognito an.
- Die Lambda-Funktion schreibt benutzerdefinierte Informationen in CloudWatch Logs und in eine DynamoDB-Tabelle.
- Ermitteln Sie benutzerdefinierte Daten aus der DynamoDB-Tabelle, zeigen Sie diese an und bereinigen Sie dann die Ressourcen.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Führen Sie ein interaktives Szenario an einem Prompt aus.

```
import (
 "context"
 "errors"
 "log"
 "strings"
 "user_pools_and_lambda_triggers/actions"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/cognitoidentityprovider"
 "github.com/aws/aws-sdk-go-v2/service/cognitoidentityprovider/types"
 "github.com/awsdocs/aws-doc-sdk-examples/gov2/demotools"
)

// ActivityLog separates the steps of this scenario into individual functions so
 that
// they are simpler to read and understand.
type ActivityLog struct {
 helper       IScenarioHelper
 questioner   demotools.IQuestioner
 resources    Resources
 cognitoActor *actions.CognitoActions
```

```
}

// NewActivityLog constructs a new activity log runner.
func NewActivityLog(sdkConfig aws.Config, questioner demotools.IQuestioner, helper
 IScenarioHelper) ActivityLog {
 scenario := ActivityLog{
  helper:       helper,
  questioner:   questioner,
  resources:    Resources{},
  cognitoActor: &actions.CognitoActions{CognitoClient:
 cognitoidentityprovider.NewFromConfig(sdkConfig)},
 }
 scenario.resources.init(scenario.cognitoActor, questioner)
 return scenario
}

// AddUserToPool selects a user from the known users table and uses administrator
 credentials to add the user to the user pool.
func (runner *ActivityLog) AddUserToPool(ctx context.Context, userPoolId string,
 tableName string) (string, string) {
 log.Println("To facilitate this example, let's add a user to the user pool using
 administrator privileges.")
 users, err := runner.helper.GetKnownUsers(ctx, tableName)
 if err != nil {
  panic(err)
 }
 user := users.Users[0]
 log.Printf("Adding known user %v to the user pool.\n", user.UserName)
 err = runner.cognitoActor.AdminCreateUser(ctx, userPoolId, user.UserName,
 user.UserEmail)
 if err != nil {
  panic(err)
 }
 pwSet := false
 password := runner.questioner.AskPassword("\nEnter a password that has at least
 eight characters, uppercase, lowercase, numbers and symbols.\n"+
  "(the password will not display as you type):", 8)
 for !pwSet {
  log.Printf("\nSetting password for user '%v'.\n", user.UserName)
  err = runner.cognitoActor.AdminSetUserPassword(ctx, userPoolId, user.UserName,
 password)
  if err != nil {
   var invalidPassword *types.InvalidPasswordException
   if errors.As(err, &invalidPassword) {
```

```
    password = runner.questioner.AskPassword("\nEnter another password:", 8)
   } else {
    panic(err)
   }
  } else {
   pwSet = true
  }
 }

 log.Println(strings.Repeat("-", 88))

 return user.UserName, password
}

// AddActivityLogTrigger adds a Lambda handler as an invocation target for the
 PostAuthentication trigger.
func (runner *ActivityLog) AddActivityLogTrigger(ctx context.Context, userPoolId
 string, activityLogArn string) {
 log.Println("Let's add a Lambda function to handle the PostAuthentication trigger
 from Cognito.\n" +
  "This trigger happens after a user is authenticated, and lets your function take
 action, such as logging\n" +
  "the outcome.")
 err := runner.cognitoActor.UpdateTriggers(
  ctx, userPoolId,
  actions.TriggerInfo{Trigger: actions.PostAuthentication, HandlerArn:
 aws.String(activityLogArn)})
 if err != nil {
  panic(err)
 }
 runner.resources.triggers = append(runner.resources.triggers,
 actions.PostAuthentication)
 log.Printf("Lambda function %v added to user pool %v to handle PostAuthentication
 Cognito trigger.\n",
  activityLogArn, userPoolId)

 log.Println(strings.Repeat("-", 88))
}

// SignInUser signs in as the specified user.
func (runner *ActivityLog) SignInUser(ctx context.Context, clientId string, userName
 string, password string) {
 log.Printf("Now we'll sign in user %v and check the results in the logs and the
 DynamoDB table.", userName)
```

```
 runner.questioner.Ask("Press Enter when you're ready.")
 authResult, err := runner.cognitoActor.SignIn(ctx, clientId, userName, password)
 if err != nil {
  panic(err)
 }
 log.Println("Sign in successful.",
  "The PostAuthentication Lambda handler writes custom information to CloudWatch
 Logs.")

 runner.resources.userAccessTokens = append(runner.resources.userAccessTokens,
 *authResult.AccessToken)
}

// GetKnownUserLastLogin gets the login info for a user from the Amazon DynamoDB
 table and displays it.
func (runner *ActivityLog) GetKnownUserLastLogin(ctx context.Context, tableName
 string, userName string) {
 log.Println("The PostAuthentication handler also writes login data to the DynamoDB
 table.")
 runner.questioner.Ask("Press Enter when you're ready to continue.")
 users, err := runner.helper.GetKnownUsers(ctx, tableName)
 if err != nil {
  panic(err)
 }
 for _, user := range users.Users {
  if user.UserName == userName {
   log.Println("The last login info for the user in the known users table is:")
   log.Printf("\t%+v", *user.LastLogin)
  }
 }
 log.Println(strings.Repeat("-", 88))
}

// Run runs the scenario.
func (runner *ActivityLog) Run(ctx context.Context, stackName string) {
 defer func() {
  if r := recover(); r != nil {
   log.Println("Something went wrong with the demo.")
   runner.resources.Cleanup(ctx)
  }
 }()

 log.Println(strings.Repeat("-", 88))
 log.Printf("Welcome\n")
```

```
 log.Println(strings.Repeat("-", 88))

 stackOutputs, err := runner.helper.GetStackOutputs(ctx, stackName)
 if err != nil {
  panic(err)
 }
 runner.resources.userPoolId = stackOutputs["UserPoolId"]
 runner.helper.PopulateUserTable(ctx, stackOutputs["TableName"])
 userName, password := runner.AddUserToPool(ctx, stackOutputs["UserPoolId"],
 stackOutputs["TableName"])

 runner.AddActivityLogTrigger(ctx, stackOutputs["UserPoolId"],
 stackOutputs["ActivityLogFunctionArn"])
 runner.SignInUser(ctx, stackOutputs["UserPoolClientId"], userName, password)
 runner.helper.ListRecentLogEvents(ctx, stackOutputs["ActivityLogFunction"])
 runner.GetKnownUserLastLogin(ctx, stackOutputs["TableName"], userName)

 runner.resources.Cleanup(ctx)

 log.Println(strings.Repeat("-", 88))
 log.Println("Thanks for watching!")
 log.Println(strings.Repeat("-", 88))
}
```

Behandeln Sie den `PostAuthentication`-Trigger mit einer Lambda-Funktion.

```
import (
 "context"
 "fmt"
 "log"
 "os"
 "time"

 "github.com/aws/aws-lambda-go/events"
 "github.com/aws/aws-lambda-go/lambda"
 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/config"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
```

```
 dynamodbtypes "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

const TABLE_NAME = "TABLE_NAME"

// LoginInfo defines structured login data that can be marshalled to a DynamoDB
 format.
type LoginInfo struct {
 UserPoolId string `dynamodbav:"UserPoolId"`
 ClientId   string `dynamodbav:"ClientId"`
 Time       string `dynamodbav:"Time"`
}

// UserInfo defines structured user data that can be marshalled to a DynamoDB
 format.
type UserInfo struct {
 UserName  string    `dynamodbav:"UserName"`
 UserEmail string    `dynamodbav:"UserEmail"`
 LastLogin LoginInfo `dynamodbav:"LastLogin"`
}

// GetKey marshals the user email value to a DynamoDB key format.
func (user UserInfo) GetKey() map[string]dynamodbtypes.AttributeValue {
 userEmail, err := attributevalue.Marshal(user.UserEmail)
 if err != nil {
  panic(err)
 }
 return map[string]dynamodbtypes.AttributeValue{"UserEmail": userEmail}
}

type handler struct {
 dynamoClient *dynamodb.Client
}

// HandleRequest handles the PostAuthentication event by writing custom data to the
 logs and
// to an Amazon DynamoDB table.
func (h *handler) HandleRequest(ctx context.Context,
 event events.CognitoEventUserPoolsPostAuthentication)
 (events.CognitoEventUserPoolsPostAuthentication, error) {
 log.Printf("Received post authentication trigger from %v for user '%v'",
 event.TriggerSource, event.UserName)
 tableName := os.Getenv(TABLE_NAME)
 user := UserInfo{
```

```
  UserName:  event.UserName,
  UserEmail: event.Request.UserAttributes["email"],
  LastLogin: LoginInfo{
   UserPoolId: event.UserPoolID,
   ClientId:   event.CallerContext.ClientID,
   Time:       time.Now().Format(time.UnixDate),
  },
 }
 // Write to CloudWatch Logs.
 fmt.Printf("%#v", user)

 // Also write to an external system. This examples uses DynamoDB to demonstrate.
 userMap, err := attributevalue.MarshalMap(user)
 if err != nil {
  log.Printf("Couldn't marshal to DynamoDB map. Here's why: %v\n", err)
 } else if len(userMap) == 0 {
  log.Printf("User info marshaled to an empty map.")
 } else {
  _, err := h.dynamoClient.PutItem(ctx, &dynamodb.PutItemInput{
   Item:      userMap,
   TableName: aws.String(tableName),
  })
  if err != nil {
   log.Printf("Couldn't write to DynamoDB. Here's why: %v\n", err)
  } else {
   log.Printf("Wrote user info to DynamoDB table %v.\n", tableName)
  }
 }

 return event, nil
}

func main() {
 ctx := context.Background()
 sdkConfig, err := config.LoadDefaultConfig(ctx)
 if err != nil {
  log.Panicln(err)
 }
 h := handler{
  dynamoClient: dynamodb.NewFromConfig(sdkConfig),
 }
 lambda.Start(h.HandleRequest)
}
```

Erstellen Sie eine Struktur, die allgemeine Aufgaben ausführt.

```
import (
 "context"
 "log"
 "strings"
 "time"
 "user_pools_and_lambda_triggers/actions"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/cloudformation"
 "github.com/aws/aws-sdk-go-v2/service/cloudwatchlogs"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/awsdocs/aws-doc-sdk-examples/gov2/demotools"
)

// IScenarioHelper defines common functions used by the workflows in this example.
type IScenarioHelper interface {
 Pause(secs int)
 GetStackOutputs(ctx context.Context, stackName string) (actions.StackOutputs,
 error)
 PopulateUserTable(ctx context.Context, tableName string)
 GetKnownUsers(ctx context.Context, tableName string) (actions.UserList, error)
 AddKnownUser(ctx context.Context, tableName string, user actions.User)
 ListRecentLogEvents(ctx context.Context, functionName string)
}

// ScenarioHelper contains AWS wrapper structs used by the workflows in this
 example.
type ScenarioHelper struct {
 questioner  demotools.IQuestioner
 dynamoActor *actions.DynamoActions
 cfnActor    *actions.CloudFormationActions
 cwlActor    *actions.CloudWatchLogsActions
 isTestRun   bool
}

// NewScenarioHelper constructs a new scenario helper.
func NewScenarioHelper(sdkConfig aws.Config, questioner demotools.IQuestioner)
 ScenarioHelper {
```

```
 scenario := ScenarioHelper{
  questioner:  questioner,
  dynamoActor: &actions.DynamoActions{DynamoClient:
 dynamodb.NewFromConfig(sdkConfig)},
  cfnActor:    &actions.CloudFormationActions{CfnClient:
 cloudformation.NewFromConfig(sdkConfig)},
  cwlActor:    &actions.CloudWatchLogsActions{CwlClient:
 cloudwatchlogs.NewFromConfig(sdkConfig)},
 }
 return scenario
}

// Pause waits for the specified number of seconds.
func (helper ScenarioHelper) Pause(secs int) {
 if !helper.isTestRun {
  time.Sleep(time.Duration(secs) * time.Second)
 }
}

// GetStackOutputs gets the outputs from the specified CloudFormation stack in a
 structured format.
func (helper ScenarioHelper) GetStackOutputs(ctx context.Context, stackName string)
 (actions.StackOutputs, error) {
 return helper.cfnActor.GetOutputs(ctx, stackName), nil
}

// PopulateUserTable fills the known user table with example data.
func (helper ScenarioHelper) PopulateUserTable(ctx context.Context, tableName
 string) {
 log.Printf("First, let's add some users to the DynamoDB %v table we'll use for this
 example.\n", tableName)
 err := helper.dynamoActor.PopulateTable(ctx, tableName)
 if err != nil {
  panic(err)
 }
}

// GetKnownUsers gets the users from the known users table in a structured format.
func (helper ScenarioHelper) GetKnownUsers(ctx context.Context, tableName string)
 (actions.UserList, error) {
 knownUsers, err := helper.dynamoActor.Scan(ctx, tableName)
 if err != nil {
  log.Printf("Couldn't get known users from table %v. Here's why: %v\n", tableName,
 err)
```

```
 }
 return knownUsers, err
}

// AddKnownUser adds a user to the known users table.
func (helper ScenarioHelper) AddKnownUser(ctx context.Context, tableName string,
 user actions.User) {
 log.Printf("Adding user '%v' with email '%v' to the DynamoDB known users table...
\n",
  user.UserName, user.UserEmail)
 err := helper.dynamoActor.AddUser(ctx, tableName, user)
 if err != nil {
  panic(err)
 }
}

// ListRecentLogEvents gets the most recent log stream and events for the specified
 Lambda function and displays them.
func (helper ScenarioHelper) ListRecentLogEvents(ctx context.Context, functionName
 string) {
 log.Println("Waiting a few seconds to let Lambda write to CloudWatch Logs...")
 helper.Pause(10)
 log.Println("Okay, let's check the logs to find what's happened recently with your
 Lambda function.")
 logStream, err := helper.cwlActor.GetLatestLogStream(ctx, functionName)
 if err != nil {
  panic(err)
 }
 log.Printf("Getting some recent events from log stream %v\n",
 *logStream.LogStreamName)
 events, err := helper.cwlActor.GetLogEvents(ctx, functionName,
 *logStream.LogStreamName, 10)
 if err != nil {
  panic(err)
 }
 for _, event := range events {
  log.Printf("\t%v", *event.Message)
 }
 log.Println(strings.Repeat("-", 88))
}
```

Erstellen Sie eine Struktur, die Amazon Cognito-Aktionen umschließt.

```
import (
 "context"
 "errors"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/cognitoidentityprovider"
 "github.com/aws/aws-sdk-go-v2/service/cognitoidentityprovider/types"
)

type CognitoActions struct {
 CognitoClient *cognitoidentityprovider.Client
}



// Trigger and TriggerInfo define typed data for updating an Amazon Cognito trigger.
type Trigger int

const (
 PreSignUp Trigger = iota
 UserMigration
 PostAuthentication
)

type TriggerInfo struct {
 Trigger    Trigger
 HandlerArn *string
}

// UpdateTriggers adds or removes Lambda triggers for a user pool. When a trigger is
 specified with a `nil` value,
// it is removed from the user pool.
func (actor CognitoActions) UpdateTriggers(ctx context.Context, userPoolId string,
 triggers ...TriggerInfo) error {
 output, err := actor.CognitoClient.DescribeUserPool(ctx,
 &cognitoidentityprovider.DescribeUserPoolInput{
  UserPoolId: aws.String(userPoolId),
 })
 if err != nil {
```

```
  log.Printf("Couldn't get info about user pool %v. Here's why: %v\n", userPoolId,
 err)
  return err
 }
 lambdaConfig := output.UserPool.LambdaConfig
 for _, trigger := range triggers {
  switch trigger.Trigger {
  case PreSignUp:
   lambdaConfig.PreSignUp = trigger.HandlerArn
  case UserMigration:
   lambdaConfig.UserMigration = trigger.HandlerArn
  case PostAuthentication:
   lambdaConfig.PostAuthentication = trigger.HandlerArn
  }
 }
 _, err = actor.CognitoClient.UpdateUserPool(ctx,
 &cognitoidentityprovider.UpdateUserPoolInput{
  UserPoolId:   aws.String(userPoolId),
  LambdaConfig: lambdaConfig,
 })
 if err != nil {
  log.Printf("Couldn't update user pool %v. Here's why: %v\n", userPoolId, err)
 }
 return err
}



// SignUp signs up a user with Amazon Cognito.
func (actor CognitoActions) SignUp(ctx context.Context, clientId string, userName
 string, password string, userEmail string) (bool, error) {
 confirmed := false
 output, err := actor.CognitoClient.SignUp(ctx,
 &cognitoidentityprovider.SignUpInput{
  ClientId: aws.String(clientId),
  Password: aws.String(password),
  Username: aws.String(userName),
  UserAttributes: []types.AttributeType{
   {Name: aws.String("email"), Value: aws.String(userEmail)},
  },
 })
 if err != nil {
  var invalidPassword *types.InvalidPasswordException
  if errors.As(err, &invalidPassword) {
```

```
   log.Println(*invalidPassword.Message)
  } else {
   log.Printf("Couldn't sign up user %v. Here's why: %v\n", userName, err)
  }
 } else {
  confirmed = output.UserConfirmed
 }
 return confirmed, err
}



// SignIn signs in a user to Amazon Cognito using a username and password
 authentication flow.
func (actor CognitoActions) SignIn(ctx context.Context, clientId string, userName
 string, password string) (*types.AuthenticationResultType, error) {
 var authResult *types.AuthenticationResultType
 output, err := actor.CognitoClient.InitiateAuth(ctx,
 &cognitoidentityprovider.InitiateAuthInput{
  AuthFlow:       "USER_PASSWORD_AUTH",
  ClientId:       aws.String(clientId),
  AuthParameters: map[string]string{"USERNAME": userName, "PASSWORD": password},
 })
 if err != nil {
  var resetRequired *types.PasswordResetRequiredException
  if errors.As(err, &resetRequired) {
   log.Println(*resetRequired.Message)
  } else {
   log.Printf("Couldn't sign in user %v. Here's why: %v\n", userName, err)
  }
 } else {
  authResult = output.AuthenticationResult
 }
 return authResult, err
}



// ForgotPassword starts a password recovery flow for a user. This flow typically
 sends a confirmation code
// to the user's configured notification destination, such as email.
func (actor CognitoActions) ForgotPassword(ctx context.Context, clientId string,
 userName string) (*types.CodeDeliveryDetailsType, error) {
```

```
 output, err := actor.CognitoClient.ForgotPassword(ctx,
 &cognitoidentityprovider.ForgotPasswordInput{
  ClientId: aws.String(clientId),
  Username: aws.String(userName),
 })
 if err != nil {
  log.Printf("Couldn't start password reset for user '%v'. Here;s why: %v\n",
 userName, err)
 }
 return output.CodeDeliveryDetails, err
}



// ConfirmForgotPassword confirms a user with a confirmation code and a new
 password.
func (actor CognitoActions) ConfirmForgotPassword(ctx context.Context, clientId
 string, code string, userName string, password string) error {
 _, err := actor.CognitoClient.ConfirmForgotPassword(ctx,
 &cognitoidentityprovider.ConfirmForgotPasswordInput{
  ClientId:         aws.String(clientId),
  ConfirmationCode: aws.String(code),
  Password:         aws.String(password),
  Username:         aws.String(userName),
 })
 if err != nil {
  var invalidPassword *types.InvalidPasswordException
  if errors.As(err, &invalidPassword) {
   log.Println(*invalidPassword.Message)
  } else {
   log.Printf("Couldn't confirm user %v. Here's why: %v", userName, err)
  }
 }
 return err
}



// DeleteUser removes a user from the user pool.
func (actor CognitoActions) DeleteUser(ctx context.Context, userAccessToken string)
 error {
 _, err := actor.CognitoClient.DeleteUser(ctx,
 &cognitoidentityprovider.DeleteUserInput{
  AccessToken: aws.String(userAccessToken),
```

```
 })
 if err != nil {
  log.Printf("Couldn't delete user. Here's why: %v\n", err)
 }
 return err
}



// AdminCreateUser uses administrator credentials to add a user to a user pool. This
 method leaves the user
// in a state that requires they enter a new password next time they sign in.
func (actor CognitoActions) AdminCreateUser(ctx context.Context, userPoolId string,
 userName string, userEmail string) error {
 _, err := actor.CognitoClient.AdminCreateUser(ctx,
 &cognitoidentityprovider.AdminCreateUserInput{
  UserPoolId:     aws.String(userPoolId),
  Username:       aws.String(userName),
  MessageAction:  types.MessageActionTypeSuppress,
  UserAttributes: []types.AttributeType{{Name: aws.String("email"), Value:
 aws.String(userEmail)}},
 })
 if err != nil {
  var userExists *types.UsernameExistsException
  if errors.As(err, &userExists) {
   log.Printf("User %v already exists in the user pool.", userName)
   err = nil
  } else {
   log.Printf("Couldn't create user %v. Here's why: %v\n", userName, err)
  }
 }
 return err
}



// AdminSetUserPassword uses administrator credentials to set a password for a user
 without requiring a
// temporary password.
func (actor CognitoActions) AdminSetUserPassword(ctx context.Context, userPoolId
 string, userName string, password string) error {
 _, err := actor.CognitoClient.AdminSetUserPassword(ctx,
 &cognitoidentityprovider.AdminSetUserPasswordInput{
  Password:   aws.String(password),
```

```
  UserPoolId: aws.String(userPoolId),
  Username:   aws.String(userName),
  Permanent:  true,
 })
 if err != nil {
  var invalidPassword *types.InvalidPasswordException
  if errors.As(err, &invalidPassword) {
   log.Println(*invalidPassword.Message)
  } else {
   log.Printf("Couldn't set password for user %v. Here's why: %v\n", userName, err)
  }
 }
 return err
}
```

Erstellen Sie eine Struktur, die DynamoDB-Aktionen umhüllt.

```
import (
 "context"
 "fmt"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// DynamoActions encapsulates the Amazon Simple Notification Service (Amazon SNS)
 actions
// used in the examples.
type DynamoActions struct {
 DynamoClient *dynamodb.Client
}

// User defines structured user data.
type User struct {
 UserName  string
 UserEmail string
 LastLogin *LoginInfo `dynamodbav:",omitempty"`
```

```
}

// LoginInfo defines structured custom login data.
type LoginInfo struct {
 UserPoolId string
 ClientId   string
 Time       string
}

// UserList defines a list of users.
type UserList struct {
 Users []User
}

// UserNameList returns the usernames contained in a UserList as a list of strings.
func (users *UserList) UserNameList() []string {
 names := make([]string, len(users.Users))
 for i := 0; i < len(users.Users); i++ {
  names[i] = users.Users[i].UserName
 }
 return names
}

// PopulateTable adds a set of test users to the table.
func (actor DynamoActions) PopulateTable(ctx context.Context, tableName string)
 error {
 var err error
 var item map[string]types.AttributeValue
 var writeReqs []types.WriteRequest
 for i := 1; i < 4; i++ {
  item, err = attributevalue.MarshalMap(User{UserName: fmt.Sprintf("test_user_%v",
 i), UserEmail: fmt.Sprintf("test_email_%v@example.com", i)})
  if err != nil {
   log.Printf("Couldn't marshall user into DynamoDB format. Here's why: %v\n", err)
   return err
  }
  writeReqs = append(writeReqs, types.WriteRequest{PutRequest:
 &types.PutRequest{Item: item}})
 }
 _, err = actor.DynamoClient.BatchWriteItem(ctx, &dynamodb.BatchWriteItemInput{
  RequestItems: map[string][]types.WriteRequest{tableName: writeReqs},
 })
 if err != nil {
```

```
  log.Printf("Couldn't populate table %v with users. Here's why: %v\n", tableName,
 err)
 }
 return err
}

// Scan scans the table for all items.
func (actor DynamoActions) Scan(ctx context.Context, tableName string) (UserList,
 error) {
 var userList UserList
 output, err := actor.DynamoClient.Scan(ctx, &dynamodb.ScanInput{
  TableName: aws.String(tableName),
 })
 if err != nil {
  log.Printf("Couldn't scan table %v for items. Here's why: %v\n", tableName, err)
 } else {
  err = attributevalue.UnmarshalListOfMaps(output.Items, &userList.Users)
  if err != nil {
   log.Printf("Couldn't unmarshal items into users. Here's why: %v\n", err)
  }
 }
 return userList, err
}

// AddUser adds a user item to a table.
func (actor DynamoActions) AddUser(ctx context.Context, tableName string, user User)
 error {
 userItem, err := attributevalue.MarshalMap(user)
 if err != nil {
  log.Printf("Couldn't marshall user to item. Here's why: %v\n", err)
 }
 _, err = actor.DynamoClient.PutItem(ctx, &dynamodb.PutItemInput{
  Item:      userItem,
  TableName: aws.String(tableName),
 })
 if err != nil {
  log.Printf("Couldn't put item in table %v. Here's why: %v", tableName, err)
 }
 return err
}
```

Erstellen Sie eine Struktur, die CloudWatch Logs-Aktionen umschließt.

```
import (
 "context"
 "fmt"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/cloudwatchlogs"
 "github.com/aws/aws-sdk-go-v2/service/cloudwatchlogs/types"
)

type CloudWatchLogsActions struct {
 CwlClient *cloudwatchlogs.Client
}

// GetLatestLogStream gets the most recent log stream for a Lambda function.
func (actor CloudWatchLogsActions) GetLatestLogStream(ctx context.Context,
 functionName string) (types.LogStream, error) {
 var logStream types.LogStream
 logGroupName := fmt.Sprintf("/aws/lambda/%s", functionName)
 output, err := actor.CwlClient.DescribeLogStreams(ctx,
 &cloudwatchlogs.DescribeLogStreamsInput{
  Descending:   aws.Bool(true),
  Limit:        aws.Int32(1),
  LogGroupName: aws.String(logGroupName),
  OrderBy:      types.OrderByLastEventTime,
 })
 if err != nil {
  log.Printf("Couldn't get log streams for log group %v. Here's why: %v\n",
 logGroupName, err)
 } else {
  logStream = output.LogStreams[0]
 }
 return logStream, err
}

// GetLogEvents gets the most recent eventCount events from the specified log
 stream.
func (actor CloudWatchLogsActions) GetLogEvents(ctx context.Context, functionName
 string, logStreamName string, eventCount int32) (
 []types.OutputLogEvent, error) {
 var events []types.OutputLogEvent
```

```
 logGroupName := fmt.Sprintf("/aws/lambda/%s", functionName)
 output, err := actor.CwlClient.GetLogEvents(ctx, &cloudwatchlogs.GetLogEventsInput{
  LogStreamName: aws.String(logStreamName),
  Limit:         aws.Int32(eventCount),
  LogGroupName:  aws.String(logGroupName),
 })
 if err != nil {
  log.Printf("Couldn't get log event for log stream %v. Here's why: %v\n",
 logStreamName, err)
 } else {
  events = output.Events
 }
 return events, err
}
```

Erstellen Sie eine Struktur, die Aktionen umschließt. AWS CloudFormation

```
import (
 "context"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/cloudformation"
)

// StackOutputs defines a map of outputs from a specific stack.
type StackOutputs map[string]string

type CloudFormationActions struct {
 CfnClient *cloudformation.Client
}

// GetOutputs gets the outputs from a CloudFormation stack and puts them into a
 structured format.
func (actor CloudFormationActions) GetOutputs(ctx context.Context, stackName string)
 StackOutputs {
 output, err := actor.CfnClient.DescribeStacks(ctx,
 &cloudformation.DescribeStacksInput{
  StackName: aws.String(stackName),
 })
```

```
 if err != nil || len(output.Stacks) == 0 {
  log.Panicf("Couldn't find a CloudFormation stack named %v. Here's why: %v\n",
 stackName, err)
 }
 stackOutputs := StackOutputs{}
 for _, out := range output.Stacks[0].Outputs {
  stackOutputs[*out.OutputKey] = *out.OutputValue
 }
 return stackOutputs
}
```

Bereinigen von Ressourcen.

```
import (
 "context"
 "log"
 "user_pools_and_lambda_triggers/actions"

 "github.com/awsdocs/aws-doc-sdk-examples/gov2/demotools"
)

// Resources keeps track of AWS resources created during an example and handles
// cleanup when the example finishes.
type Resources struct {
 userPoolId       string
 userAccessTokens []string
 triggers         []actions.Trigger

 cognitoActor *actions.CognitoActions
 questioner   demotools.IQuestioner
}

func (resources *Resources) init(cognitoActor *actions.CognitoActions, questioner
 demotools.IQuestioner) {
 resources.userAccessTokens = []string{}
 resources.triggers = []actions.Trigger{}
 resources.cognitoActor = cognitoActor
 resources.questioner = questioner
}
```

```
// Cleanup deletes all AWS resources created during an example.
func (resources *Resources) Cleanup(ctx context.Context) {
 defer func() {
  if r := recover(); r != nil {
   log.Printf("Something went wrong during cleanup.\n%v\n", r)
   log.Println("Use the AWS Management Console to remove any remaining resources \n"
 +
    "that were created for this scenario.")
  }
 }()

 wantDelete := resources.questioner.AskBool("Do you want to remove all of the AWS
 resources that were created "+
  "during this demo (y/n)?", "y")
 if wantDelete {
  for _, accessToken := range resources.userAccessTokens {
   err := resources.cognitoActor.DeleteUser(ctx, accessToken)
   if err != nil {
    log.Println("Couldn't delete user during cleanup.")
    panic(err)
   }
   log.Println("Deleted user.")
  }
  triggerList := make([]actions.TriggerInfo, len(resources.triggers))
  for i := 0; i < len(resources.triggers); i++ {
   triggerList[i] = actions.TriggerInfo{Trigger: resources.triggers[i], HandlerArn:
 nil}
  }
  err := resources.cognitoActor.UpdateTriggers(ctx, resources.userPoolId,
 triggerList...)
  if err != nil {
   log.Println("Couldn't update Cognito triggers during cleanup.")
   panic(err)
  }
  log.Println("Removed Cognito triggers from user pool.")
 } else {
  log.Println("Be sure to remove resources when you're done with them to avoid
 unexpected charges!")
 }
}
```

- API-Details finden Sie in den folgenden Themen der AWS SDK für Go -API-Referenz.

- AdminCreateUser
- AdminSetUserPassword
- DeleteUser
- InitiateAuth
- UpdateUserPool

Serverless-Beispiele

Herstellen einer Verbindung mit einer Amazon-RDS-Datenbank in einer Lambda-Funktion

Das folgende Codebeispiel zeigt, wie eine Lambda-Funktion implementiert wird, die eine Verbindung zu einer RDS-Datenbank herstellt. Die Funktion stellt eine einfache Datenbankanfrage und gibt das Ergebnis zurück.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu. GitHub Das vollständige Beispiel sowie eine Anleitung zum Einrichten und Ausführen finden Sie im Repository mit Serverless-Beispielen.

Herstellen einer Verbindung zu einer Amazon RDS-Datenbank in einer Lambda-Funktion mit Go.

```
/*
Golang v2 code here.
*/

package main

import (
 "context"
 "database/sql"
 "encoding/json"
 "fmt"
 "os"

 "github.com/aws/aws-lambda-go/lambda"
 "github.com/aws/aws-sdk-go-v2/config"
```

```
 "github.com/aws/aws-sdk-go-v2/feature/rds/auth"
 _ "github.com/go-sql-driver/mysql"
)

type MyEvent struct {
 Name string `json:"name"`
}

func HandleRequest(event *MyEvent) (map[string]interface{}, error) {

 var dbName string = os.Getenv("DatabaseName")
 var dbUser string = os.Getenv("DatabaseUser")
 var dbHost string = os.Getenv("DBHost") // Add hostname without https
 var dbPort int = os.Getenv("Port")      // Add port number
 var dbEndpoint string = fmt.Sprintf("%s:%d", dbHost, dbPort)
 var region string = os.Getenv("AWS_REGION")

 cfg, err := config.LoadDefaultConfig(context.TODO())
 if err != nil {
  panic("configuration error: " + err.Error())
 }

 authenticationToken, err := auth.BuildAuthToken(
  context.TODO(), dbEndpoint, region, dbUser, cfg.Credentials)
 if err != nil {
  panic("failed to create authentication token: " + err.Error())
 }

 dsn := fmt.Sprintf("%s:%s@tcp(%s)/%s?tls=true&allowCleartextPasswords=true",
  dbUser, authenticationToken, dbEndpoint, dbName,
 )

 db, err := sql.Open("mysql", dsn)
 if err != nil {
  panic(err)
 }

 defer db.Close()

 var sum int
 err = db.QueryRow("SELECT ?+? AS sum", 3, 2).Scan(&sum)
 if err != nil {
  panic(err)
 }
```

```
 s := fmt.Sprint(sum)
 message := fmt.Sprintf("The selected sum is: %s", s)

 messageBytes, err := json.Marshal(message)
 if err != nil {
  return nil, err
 }

 messageString := string(messageBytes)
 return map[string]interface{}{
  "statusCode": 200,
  "headers":    map[string]string{"Content-Type": "application/json"},
  "body":       messageString,
 }, nil
}

func main() {
 lambda.Start(HandleRequest)
}
```

Aufrufen einer Lambda-Funktion über einen Kinesis-Auslöser

Das folgende Codebeispiel zeigt, wie eine Lambda-Funktion implementiert wird, die ein Ereignis empfängt, das durch den Empfang von Datensätzen aus einem Kinesis-Stream ausgelöst wird. Die Funktion ruft die Kinesis-Nutzlast ab, dekodiert von Base64 und protokolliert den Datensatzinhalt.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu. GitHub Das vollständige Beispiel sowie eine Anleitung zum Einrichten und Ausführen finden Sie im Repository mit Serverless-Beispielen.

Nutzen eines Kinesis-Ereignisses mit Lambda unter Verwendung von Go.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
package main
```

```
import (
 "context"
 "log"

 "github.com/aws/aws-lambda-go/events"
 "github.com/aws/aws-lambda-go/lambda"
)

func handler(ctx context.Context, kinesisEvent events.KinesisEvent) error {
 if len(kinesisEvent.Records) == 0 {
  log.Printf("empty Kinesis event received")
  return nil
 }

 for _, record := range kinesisEvent.Records {
  log.Printf("processed Kinesis event with EventId: %v", record.EventID)
  recordDataBytes := record.Kinesis.Data
  recordDataText := string(recordDataBytes)
  log.Printf("record data: %v", recordDataText)
  // TODO: Do interesting work based on the new data
 }
 log.Printf("successfully processed %v records", len(kinesisEvent.Records))
 return nil
}

func main() {
 lambda.Start(handler)
}
```

Aufrufen einer Lambda-Funktion über einen DynamoDB-Auslöser

Das folgende Codebeispiel zeigt, wie eine Lambda-Funktion implementiert wird, die ein Ereignis empfängt, das durch den Empfang von Datensätzen aus einem DynamoDB-Stream ausgelöst wird. Die Funktion ruft die DynamoDB-Nutzdaten ab und protokolliert den Inhalt des Datensatzes.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu. GitHub Das vollständige Beispiel sowie eine Anleitung zum Einrichten und Ausführen finden Sie im Repository mit Serverless-Beispielen.

Nutzen eines DynamoDB-Ereignisses mit Lambda unter Verwendung von Go.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
package main

import (
 "context"
 "github.com/aws/aws-lambda-go/lambda"
 "github.com/aws/aws-lambda-go/events"
 "fmt"
)

func HandleRequest(ctx context.Context, event events.DynamoDBEvent) (*string, error)
 {
 if len(event.Records) == 0 {
  return nil, fmt.Errorf("received empty event")
 }

 for _, record := range event.Records {
   LogDynamoDBRecord(record)
 }

 message := fmt.Sprintf("Records processed: %d", len(event.Records))
 return &message, nil
}

func main() {
 lambda.Start(HandleRequest)
}

func LogDynamoDBRecord(record events.DynamoDBEventRecord){
 fmt.Println(record.EventID)
 fmt.Println(record.EventName)
 fmt.Printf("%+v\n", record.Change)
```

```
}
```

Aufrufen einer Lambda-Funktion über einen Amazon DocumentDB-Auslöser

Das folgende Codebeispiel zeigt, wie eine Lambda-Funktion implementiert wird, die ein Ereignis empfängt, das durch den Empfang von Datensätzen aus einem DocumentDB-Änderungsstream ausgelöst wird. Die Funktion ruft die DocumentDB-Nutzdaten ab und protokolliert den Inhalt des Datensatzes.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu. GitHub Das vollständige Beispiel sowie eine Anleitung zum Einrichten und Ausführen finden Sie im Repository mit Serverless-Beispielen.

Nutzen eines Amazon DocumentDB-Ereignisses mit Lambda unter Verwendung von Go.

```
package main

import (
 "context"
 "encoding/json"
 "fmt"

 "github.com/aws/aws-lambda-go/lambda"
)

type Event struct {
 Events []Record `json:"events"`
}

type Record struct {
 Event struct {
  OperationType string `json:"operationType"`
  NS            struct {
   DB   string `json:"db"`
   Coll string `json:"coll"`
  } `json:"ns"`
```

```
  FullDocument interface{} `json:"fullDocument"`
 } `json:"event"`
}

func main() {
 lambda.Start(handler)
}

func handler(ctx context.Context, event Event) (string, error) {
 fmt.Println("Loading function")
 for _, record := range event.Events {
  logDocumentDBEvent(record)
 }

 return "OK", nil
}

func logDocumentDBEvent(record Record) {
 fmt.Printf("Operation type: %s\n", record.Event.OperationType)
 fmt.Printf("db: %s\n", record.Event.NS.DB)
 fmt.Printf("collection: %s\n", record.Event.NS.Coll)
 docBytes, _ := json.MarshalIndent(record.Event.FullDocument, "", "  ")
 fmt.Printf("Full document: %s\n", string(docBytes))
}
```

Aufrufen einer Lambda-Funktion über einen Amazon-MSK-Auslöser

Das folgende Codebeispiel zeigt, wie eine Lambda-Funktion implementiert wird, die ein Ereignis empfängt, das durch den Empfang von Datensätzen aus einem Amazon MSK-Cluster ausgelöst wird. Die Funktion ruft die MSK-Nutzdaten ab und protokolliert den Inhalt des Datensatzes.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu. GitHub Das vollständige Beispiel sowie eine Anleitung zum Einrichten und Ausführen finden Sie im Repository mit Serverless-Beispielen.

Nutzen eines Amazon MSK-Ereignisses mit Lambda unter Verwendung von Go.

```
package main

import (
 "encoding/base64"
 "fmt"

 "github.com/aws/aws-lambda-go/events"
 "github.com/aws/aws-lambda-go/lambda"
)

func handler(event events.KafkaEvent) {
 for key, records := range event.Records {
  fmt.Println("Key:", key)

  for _, record := range records {
   fmt.Println("Record:", record)

   decodedValue, _ := base64.StdEncoding.DecodeString(record.Value)
   message := string(decodedValue)
   fmt.Println("Message:", message)
  }
 }
}

func main() {
 lambda.Start(handler)
}
```

Aufrufen einer Lambda-Funktion über einen Amazon-S3-Auslöser

Das folgende Codebeispiel zeigt, wie eine Lambda-Funktion implementiert wird, die ein Ereignis empfängt, das durch das Hochladen eines Objekts in einen S3-Bucket ausgelöst wird. Die Funktion ruft den Namen des S3-Buckets sowie den Objektschlüssel aus dem Ereignisparameter ab und ruft die Amazon-S3-API auf, um den Inhaltstyp des Objekts abzurufen und zu protokollieren.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu. GitHub Das vollständige Beispiel sowie eine Anleitung zum Einrichten und Ausführen finden Sie im Repository mit Serverless-Beispielen.

Nutzen eines S3-Ereignisses mit Lambda unter Verwendung von Go

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
package main

import (
 "context"
 "log"

 "github.com/aws/aws-lambda-go/events"
 "github.com/aws/aws-lambda-go/lambda"
 "github.com/aws/aws-sdk-go-v2/config"
 "github.com/aws/aws-sdk-go-v2/service/s3"
)

func handler(ctx context.Context, s3Event events.S3Event) error {
 sdkConfig, err := config.LoadDefaultConfig(ctx)
 if err != nil {
  log.Printf("failed to load default config: %s", err)
  return err
 }
 s3Client := s3.NewFromConfig(sdkConfig)

 for _, record := range s3Event.Records {
  bucket := record.S3.Bucket.Name
  key := record.S3.Object.URLDecodedKey
  headOutput, err := s3Client.HeadObject(ctx, &s3.HeadObjectInput{
   Bucket: &bucket,
   Key:    &key,
  })
  if err != nil {
   log.Printf("error getting head of object %s/%s: %s", bucket, key, err)
   return err
  }
```

```
  log.Printf("successfully retrieved %s/%s of type %s", bucket, key,
 *headOutput.ContentType)
 }

 return nil
}

func main() {
 lambda.Start(handler)
}
```

Eine Lambda-Funktion über einen Amazon-SNS-Trigger aufrufen

Das folgende Codebeispiel zeigt, wie eine Lambda-Funktion implementiert wird, die ein Ereignis empfängt, das durch den Empfang von Nachrichten von einem SNS-Thema ausgelöst wird. Die Funktion ruft die Nachrichten aus dem Ereignisparameter ab und protokolliert den Inhalt jeder Nachricht.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu. GitHub Das vollständige Beispiel sowie eine Anleitung zum Einrichten und Ausführen finden Sie im Repository mit Serverless-Beispielen.

Nutzen eines SNS-Ereignisses mit Lambda unter Verwendung von Go.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
package main

import (
 "context"
 "fmt"

 "github.com/aws/aws-lambda-go/events"
 "github.com/aws/aws-lambda-go/lambda"
)
```

```
func handler(ctx context.Context, snsEvent events.SNSEvent) {
 for _, record := range snsEvent.Records {
  processMessage(record)
 }
 fmt.Println("done")
}

func processMessage(record events.SNSEventRecord) {
 message := record.SNS.Message
 fmt.Printf("Processed message: %s\n", message)
 // TODO: Process your record here
}

func main() {
 lambda.Start(handler)
}
```

Aufrufen einer Lambda-Funktion über einen Amazon-SQS-Auslöser

Das folgende Codebeispiel zeigt, wie eine Lambda-Funktion implementiert wird, die ein Ereignis empfängt, das durch den Empfang von Nachrichten aus einer SQS-Warteschlange ausgelöst wird. Die Funktion ruft die Nachrichten aus dem Ereignisparameter ab und protokolliert den Inhalt jeder Nachricht.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu. GitHub Das vollständige Beispiel sowie eine Anleitung zum Einrichten und Ausführen finden Sie im Repository mit Serverless-Beispielen.

Nutzen eines SQS-Ereignisses mit Lambda unter Verwendung von Go.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
package integration_sqs_to_lambda

import (
 "fmt"
```

```
 "github.com/aws/aws-lambda-go/events"
 "github.com/aws/aws-lambda-go/lambda"
)

func handler(event events.SQSEvent) error {
 for _, record := range event.Records {
  err := processMessage(record)
  if err != nil {
   return err
  }
 }
 fmt.Println("done")
 return nil
}

func processMessage(record events.SQSMessage) error {
 fmt.Printf("Processed message %s\n", record.Body)
 // TODO: Do interesting work based on the new message
 return nil
}

func main() {
 lambda.Start(handler)
}
```

Melden von Batch-Elementfehlern für Lambda-Funktionen mit einem Kinesis-Auslöser

Das folgende Codebeispiel zeigt, wie eine partielle Batch-Antwort für Lambda-Funktionen implementiert wird, die Ereignisse aus einem Kinesis-Stream empfangen. Die Funktion meldet die Batch-Elementfehler in der Antwort und signalisiert Lambda, diese Nachrichten später erneut zu versuchen.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu. GitHub Das vollständige Beispiel sowie eine Anleitung zum Einrichten und Ausführen finden Sie im Repository mit Serverless-Beispielen.

Melden von Fehlern bei Kinesis-Batchelementen mit Lambda unter Verwendung von Go.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
package main

import (
 "context"
 "fmt"
 "github.com/aws/aws-lambda-go/events"
 "github.com/aws/aws-lambda-go/lambda"
)

func handler(ctx context.Context, kinesisEvent events.KinesisEvent)
 (map[string]interface{}, error) {
 batchItemFailures := []map[string]interface{}{}

 for _, record := range kinesisEvent.Records {
  curRecordSequenceNumber := ""

  // Process your record
  if /* Your record processing condition here */ {
   curRecordSequenceNumber = record.Kinesis.SequenceNumber
  }

  // Add a condition to check if the record processing failed
  if curRecordSequenceNumber != "" {
   batchItemFailures = append(batchItemFailures, map[string]interface{}
{"itemIdentifier": curRecordSequenceNumber})
  }
 }

 kinesisBatchResponse := map[string]interface{}{
  "batchItemFailures": batchItemFailures,
 }
 return kinesisBatchResponse, nil
}

func main() {
 lambda.Start(handler)
}
```

Melden von Batch-Elementfehlern für Lambda-Funktionen mit einem DynamoDB-Auslöser

Das folgende Codebeispiel zeigt, wie eine partielle Batch-Antwort für Lambda-Funktionen implementiert wird, die Ereignisse aus einem DynamoDB-Stream empfangen. Die Funktion meldet die Batch-Elementfehler in der Antwort und signalisiert Lambda, diese Nachrichten später erneut zu versuchen.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu. GitHub Das vollständige Beispiel sowie eine Anleitung zum Einrichten und Ausführen finden Sie im Repository mit Serverless-Beispielen.

Melden von DynamoDB-Batchelementfehlern mit Lambda unter Verwendung von Go.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
package main

import (
 "context"
 "github.com/aws/aws-lambda-go/events"
 "github.com/aws/aws-lambda-go/lambda"
)

type BatchItemFailure struct {
 ItemIdentifier string `json:"ItemIdentifier"`
}

type BatchResult struct {
 BatchItemFailures []BatchItemFailure `json:"BatchItemFailures"`
}

func HandleRequest(ctx context.Context, event events.DynamoDBEvent) (*BatchResult,
 error) {
 var batchItemFailures []BatchItemFailure
 curRecordSequenceNumber := ""

 for _, record := range event.Records {
  // Process your record
```

```
  curRecordSequenceNumber = record.Change.SequenceNumber
 }

 if curRecordSequenceNumber != "" {
  batchItemFailures = append(batchItemFailures, BatchItemFailure{ItemIdentifier:
 curRecordSequenceNumber})
 }

 batchResult := BatchResult{
  BatchItemFailures: batchItemFailures,
 }

 return &batchResult, nil
}

func main() {
 lambda.Start(HandleRequest)
}
```

Melden von Batch-Elementfehlern für Lambda-Funktionen mit einem Amazon-SQS-Auslöser

Das folgende Codebeispiel zeigt, wie eine partielle Batch-Antwort für Lambda-Funktionen implementiert wird, die Ereignisse aus einer SQS-Warteschlange empfangen. Die Funktion meldet die Batch-Elementfehler in der Antwort und signalisiert Lambda, diese Nachrichten später erneut zu versuchen.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu. GitHub Das vollständige Beispiel sowie eine Anleitung zum Einrichten und Ausführen finden Sie im Repository mit Serverless-Beispielen.

Melden von SQS-Batchelementfehlern mit Lambda unter Verwendung von Go.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
package main
```

```
import (
 "context"
 "encoding/json"
 "fmt"
 "github.com/aws/aws-lambda-go/events"
 "github.com/aws/aws-lambda-go/lambda"
)

func handler(ctx context.Context, sqsEvent events.SQSEvent) (map[string]interface{},
 error) {
 batchItemFailures := []map[string]interface{}{}

 for _, message := range sqsEvent.Records {

  if /* Your message processing condition here */ {
   batchItemFailures = append(batchItemFailures, map[string]interface{}
{"itemIdentifier": message.MessageId})
  }
 }

 sqsBatchResponse := map[string]interface{}{
  "batchItemFailures": batchItemFailures,
 }
 return sqsBatchResponse, nil
}

func main() {
 lambda.Start(handler)
}
```

AWS Beiträge der Gemeinschaft

Erstellen und testen Sie eine serverlose Anwendung

Das folgende Codebeispiel zeigt, wie eine serverlose Anwendung mithilfe von API Gateway mit Lambda und DynamoDB erstellt und getestet wird.

SDK für Go V2

> Zeigt, wie eine serverlose Anwendung, die aus einem API Gateway mit Lambda und DynamoDB besteht, mithilfe des Go SDK erstellt und getestet wird.

Den vollständigen Quellcode und Anweisungen zur Einrichtung und Ausführung finden Sie im vollständigen Beispiel unter. GitHub

In diesem Beispiel verwendete Dienste

- API Gateway
- DynamoDB
- Lambda

Amazon MSK-Beispiele mit SDK for Go V2

Die folgenden Codebeispiele zeigen Ihnen, wie Sie mithilfe von AWS SDK für Go V2 mit Amazon MSK Aktionen ausführen und allgemeine Szenarien implementieren.

Jedes Beispiel enthält einen Link zum vollständigen Quellcode, in dem Sie Anweisungen zur Einrichtung und Ausführung des Codes im Kontext finden.

Themen

- Serverless-Beispiele

Serverless-Beispiele

Aufrufen einer Lambda-Funktion über einen Amazon-MSK-Auslöser

Das folgende Codebeispiel zeigt, wie eine Lambda-Funktion implementiert wird, die ein Ereignis empfängt, das durch den Empfang von Datensätzen aus einem Amazon MSK-Cluster ausgelöst wird. Die Funktion ruft die MSK-Nutzdaten ab und protokolliert den Inhalt des Datensatzes.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu. GitHub Das vollständige Beispiel sowie eine Anleitung zum Einrichten und Ausführen finden Sie im Repository mit Serverless-Beispielen.

Nutzen eines Amazon MSK-Ereignisses mit Lambda unter Verwendung von Go.

```
package main

import (
 "encoding/base64"
 "fmt"

 "github.com/aws/aws-lambda-go/events"
 "github.com/aws/aws-lambda-go/lambda"
)

func handler(event events.KafkaEvent) {
 for key, records := range event.Records {
  fmt.Println("Key:", key)

  for _, record := range records {
   fmt.Println("Record:", record)

   decodedValue, _ := base64.StdEncoding.DecodeString(record.Value)
   message := string(decodedValue)
   fmt.Println("Message:", message)
  }
 }
}

func main() {
 lambda.Start(handler)
}
```

Partner Central-Beispiele mit SDK for Go V2

Die folgenden Codebeispiele zeigen Ihnen, wie Sie mithilfe von AWS SDK für Go V2 mit Partner Central Aktionen ausführen und allgemeine Szenarien implementieren.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Service-Funktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarios anzeigen.

Jedes Beispiel enthält einen Link zum vollständigen Quellcode, in dem Sie Anweisungen zum Einrichten und Ausführen des Codes im Kontext finden.

Themen

- Aktionen

Aktionen

`GetOpportunity`

Das folgende Codebeispiel zeigt die Verwendung`GetOpportunity`.

SDK für Go V2

Holen Sie sich eine Gelegenheit.

```
package main

import (
 "context"
 "encoding/json"
 "fmt"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/config"
 "github.com/aws/aws-sdk-go-v2/service/partnercentralselling"
)

func main() {
 config, err := config.LoadDefaultConfig(context.TODO())

 if err != nil {
  log.Fatal(err)
 }

 config.Region = "us-east-1"

 client := partnercentralselling.NewFromConfig(config)

 output, err := client.GetOpportunity(context.TODO(),
 &partnercentralselling.GetOpportunityInput{
  Identifier: aws.String("O1111111"),
  Catalog:    aws.String("AWS"),
 })
```

```
 if err != nil {
  log.Fatal(err)
 }
 log.Println("printing opportuniy...\n")

 jsonOutput, err := json.MarshalIndent(output, "", "    ")

 fmt.Println(string(jsonOutput))
}
```

- Einzelheiten zur API finden Sie GetOpportunityin der AWS SDK für Go API-Referenz.

`ListOpportunities`

Das folgende Codebeispiel zeigt die Verwendung`ListOpportunities`.

SDK für Go V2

Möglichkeiten auflisten.

```
package main

import (
 "context"
 "encoding/json"
 "fmt"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/config"
 "github.com/aws/aws-sdk-go-v2/service/partnercentralselling"
)

func main() {
 config, err := config.LoadDefaultConfig(context.TODO())

 if err != nil {
  log.Fatal(err)
 }

 config.Region = "us-east-1"
```

```
 client := partnercentralselling.NewFromConfig(config)

 output, err := client.ListOpportunities(context.TODO(),
 &partnercentralselling.ListOpportunitiesInput{
  MaxResults: aws.Int32(2),
  Catalog:    aws.String("AWS"),
 })

 if err != nil {
  log.Fatal(err)
 }

 jsonOutput, err := json.MarshalIndent(output, "", "    ")
 fmt.Println(string(jsonOutput))
}
```

- Einzelheiten zur API finden Sie ListOpportunitiesin der AWS SDK für Go API-Referenz.

Amazon RDS-Beispiele mit SDK for Go V2

Die folgenden Codebeispiele zeigen Ihnen, wie Sie mithilfe von AWS SDK für Go V2 mit Amazon RDS Aktionen ausführen und allgemeine Szenarien implementieren.

Bei Grundlagen handelt es sich um Code-Beispiele, die Ihnen zeigen, wie Sie die wesentlichen Vorgänge innerhalb eines Services ausführen.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Service-Funktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarios anzeigen.

Jedes Beispiel enthält einen Link zum vollständigen Quellcode, in dem Sie Anweisungen zur Einrichtung und Ausführung des Codes im Kontext finden.

Erste Schritte

Hello Amazon RDS

Die folgenden Codebeispiele veranschaulichen die ersten Schritte mit Amazon RDS.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
package main

import (
 "context"
 "fmt"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/config"
 "github.com/aws/aws-sdk-go-v2/service/rds"
)

// main uses the AWS SDK for Go V2 to create an Amazon Relational Database Service
 (Amazon RDS)
// client and list up to 20 DB instances in your account.
// This example uses the default settings specified in your shared credentials
// and config files.
func main() {
 ctx := context.Background()
 sdkConfig, err := config.LoadDefaultConfig(ctx)
 if err != nil {
  fmt.Println("Couldn't load default configuration. Have you set up your AWS
 account?")
  fmt.Println(err)
  return
 }
 rdsClient := rds.NewFromConfig(sdkConfig)
 const maxInstances = 20
 fmt.Printf("Let's list up to %v DB instances.\n", maxInstances)
 output, err := rdsClient.DescribeDBInstances(ctx,
  &rds.DescribeDBInstancesInput{MaxRecords: aws.Int32(maxInstances)})
 if err != nil {
  fmt.Printf("Couldn't list DB instances: %v\n", err)
  return
```

```
 }
 if len(output.DBInstances) == 0 {
  fmt.Println("No DB instances found.")
 } else {
  for _, instance := range output.DBInstances {
   fmt.Printf("DB instance %v has database %v.\n", *instance.DBInstanceIdentifier,
    *instance.DBName)
  }
 }
}
```

- Einzelheiten zur API finden Sie unter Describe DBInstances in der AWS SDK für Go API-Referenz.

Themen

- Grundlagen
- Aktionen
- Serverless-Beispiele

Grundlagen

Erlernen der Grundlagen

Wie das aussehen kann, sehen Sie am nachfolgenden Beispielcode:

- Erstellen Sie eine benutzerdefinierte DB-Parametergruppe und legen Sie Parameterwerte fest.
- Erstellen Sie eine DB-Instance, die zur Verwendung der Parametergruppe konfiguriert ist. Die DB-Instance enthält auch eine Datenbank.
- Erstellen Sie einen Snapshot der Instance.
- Löschen Sie die Instance und die Parametergruppe.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Führen Sie ein interaktives Szenario an einem Prompt aus.

```
import (
 "context"
 "fmt"
 "log"
 "sort"
 "strconv"
 "strings"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/rds"
 "github.com/aws/aws-sdk-go-v2/service/rds/types"
 "github.com/awsdocs/aws-doc-sdk-examples/gov2/demotools"
 "github.com/awsdocs/aws-doc-sdk-examples/gov2/rds/actions"
 "github.com/google/uuid"
)

// GetStartedInstances is an interactive example that shows you how to use the AWS
 SDK for Go
// with Amazon Relation Database Service (Amazon RDS) to do the following:
//
//  1. Create a custom DB parameter group and set parameter values.
//  2. Create a DB instance that is configured to use the parameter group. The DB
 instance
//     also contains a database.
//  3. Take a snapshot of the DB instance.
//  4. Delete the DB instance and parameter group.
type GetStartedInstances struct {
 sdkConfig  aws.Config
 instances  actions.DbInstances
 questioner demotools.IQuestioner
 helper     IScenarioHelper
```

```
 isTestRun  bool
}

// NewGetStartedInstances constructs a GetStartedInstances instance from a
 configuration.
// It uses the specified config to get an Amazon RDS
// client and create wrappers for the actions used in the scenario.
func NewGetStartedInstances(sdkConfig aws.Config, questioner demotools.IQuestioner,
 helper IScenarioHelper) GetStartedInstances {
 rdsClient := rds.NewFromConfig(sdkConfig)
 return GetStartedInstances{
  sdkConfig:  sdkConfig,
  instances:  actions.DbInstances{RdsClient: rdsClient},
  questioner: questioner,
  helper:     helper,
 }
}

// Run runs the interactive scenario.
func (scenario GetStartedInstances) Run(ctx context.Context, dbEngine string,
 parameterGroupName string,
 instanceName string, dbName string) {
 defer func() {
  if r := recover(); r != nil {
   log.Println("Something went wrong with the demo.")
  }
 }()

 log.Println(strings.Repeat("-", 88))
 log.Println("Welcome to the Amazon Relational Database Service (Amazon RDS) DB
 Instance demo.")
 log.Println(strings.Repeat("-", 88))

 parameterGroup := scenario.CreateParameterGroup(ctx, dbEngine, parameterGroupName)
 scenario.SetUserParameters(ctx, parameterGroupName)
 instance := scenario.CreateInstance(ctx, instanceName, dbEngine, dbName,
 parameterGroup)
 scenario.DisplayConnection(instance)
 scenario.CreateSnapshot(ctx, instance)
 scenario.Cleanup(ctx, instance, parameterGroup)

 log.Println(strings.Repeat("-", 88))
 log.Println("Thanks for watching!")
 log.Println(strings.Repeat("-", 88))
```

```
}

// CreateParameterGroup shows how to get available engine versions for a specified
// database engine and create a DB parameter group that is compatible with a
// selected engine family.
func (scenario GetStartedInstances) CreateParameterGroup(ctx context.Context,
 dbEngine string,
 parameterGroupName string) *types.DBParameterGroup {

 log.Printf("Checking for an existing DB parameter group named %v.\n",
  parameterGroupName)
 parameterGroup, err := scenario.instances.GetParameterGroup(ctx,
 parameterGroupName)
 if err != nil {
  panic(err)
 }
 if parameterGroup == nil {
  log.Printf("Getting available database engine versions for %v.\n", dbEngine)
  engineVersions, err := scenario.instances.GetEngineVersions(ctx, dbEngine, "")
  if err != nil {
   panic(err)
  }

  familySet := map[string]struct{}{}
  for _, family := range engineVersions {
   familySet[*family.DBParameterGroupFamily] = struct{}{}
  }
  var families []string
  for family := range familySet {
   families = append(families, family)
  }
  sort.Strings(families)
  familyIndex := scenario.questioner.AskChoice("Which family do you want to use?\n",
 families)
  log.Println("Creating a DB parameter group.")
  _, err = scenario.instances.CreateParameterGroup(
   ctx, parameterGroupName, families[familyIndex], "Example parameter group.")
  if err != nil {
   panic(err)
  }
  parameterGroup, err = scenario.instances.GetParameterGroup(ctx,
 parameterGroupName)
  if err != nil {
   panic(err)
```

```
  }
 }
 log.Printf("Parameter group %v:\n", *parameterGroup.DBParameterGroupFamily)
 log.Printf("\tName: %v\n", *parameterGroup.DBParameterGroupName)
 log.Printf("\tARN: %v\n", *parameterGroup.DBParameterGroupArn)
 log.Printf("\tFamily: %v\n", *parameterGroup.DBParameterGroupFamily)
 log.Printf("\tDescription: %v\n", *parameterGroup.Description)
 log.Println(strings.Repeat("-", 88))
 return parameterGroup
}

// SetUserParameters shows how to get the parameters contained in a custom parameter
// group and update some of the parameter values in the group.
func (scenario GetStartedInstances) SetUserParameters(ctx context.Context,
 parameterGroupName string) {
 log.Println("Let's set some parameter values in your parameter group.")
 dbParameters, err := scenario.instances.GetParameters(ctx, parameterGroupName, "")
 if err != nil {
  panic(err)
 }
 var updateParams []types.Parameter
 for _, dbParam := range dbParameters {
  if strings.HasPrefix(*dbParam.ParameterName, "auto_increment") &&
   *dbParam.IsModifiable && *dbParam.DataType == "integer" {
   log.Printf("The %v parameter is described as:\n\t%v",
    *dbParam.ParameterName, *dbParam.Description)
   rangeSplit := strings.Split(*dbParam.AllowedValues, "-")
   lower, _ := strconv.Atoi(rangeSplit[0])
   upper, _ := strconv.Atoi(rangeSplit[1])
   newValue := scenario.questioner.AskInt(
    fmt.Sprintf("Enter a value between %v and %v:", lower, upper),
    demotools.InIntRange{Lower: lower, Upper: upper})
   dbParam.ParameterValue = aws.String(strconv.Itoa(newValue))
   updateParams = append(updateParams, dbParam)
  }
 }
 err = scenario.instances.UpdateParameters(ctx, parameterGroupName, updateParams)
 if err != nil {
  panic(err)
 }
 log.Println("To get a list of parameters that you set previously, specify a source
 of 'user'.")
 userParameters, err := scenario.instances.GetParameters(ctx, parameterGroupName,
 "user")
```

```
 if err != nil {
  panic(err)
 }
 log.Println("Here are the parameters you set:")
 for _, param := range userParameters {
  log.Printf("\t%v: %v\n", *param.ParameterName, *param.ParameterValue)
 }
 log.Println(strings.Repeat("-", 88))
}

// CreateInstance shows how to create a DB instance that contains a database of a
// specified type. The database is also configured to use a custom DB parameter
 group.
func (scenario GetStartedInstances) CreateInstance(ctx context.Context, instanceName
 string, dbEngine string,
 dbName string, parameterGroup *types.DBParameterGroup) *types.DBInstance {

 log.Println("Checking for an existing DB instance.")
 instance, err := scenario.instances.GetInstance(ctx, instanceName)
 if err != nil {
  panic(err)
 }
 if instance == nil {
  adminUsername := scenario.questioner.Ask(
   "Enter an administrator username for the database: ", demotools.NotEmpty{})
  adminPassword := scenario.questioner.AskPassword(
   "Enter a password for the administrator (at least 8 characters): ", 7)
  engineVersions, err := scenario.instances.GetEngineVersions(ctx, dbEngine,
   *parameterGroup.DBParameterGroupFamily)
  if err != nil {
   panic(err)
  }
  var engineChoices []string
  for _, engine := range engineVersions {
   engineChoices = append(engineChoices, *engine.EngineVersion)
  }
  engineIndex := scenario.questioner.AskChoice(
   "The available engines for your parameter group are:\n", engineChoices)
  engineSelection := engineVersions[engineIndex]
  instOpts, err := scenario.instances.GetOrderableInstances(ctx,
 *engineSelection.Engine,
   *engineSelection.EngineVersion)
  if err != nil {
   panic(err)
```

```
 }
 optSet := map[string]struct{}{}
 for _, opt := range instOpts {
  if strings.Contains(*opt.DBInstanceClass, "micro") {
   optSet[*opt.DBInstanceClass] = struct{}{}
  }
 }
 var optChoices []string
 for opt := range optSet {
  optChoices = append(optChoices, opt)
 }
 sort.Strings(optChoices)
 optIndex := scenario.questioner.AskChoice(
  "The available micro DB instance classes for your database engine are:\n",
optChoices)
 storageType := "standard"
 allocatedStorage := int32(5)
 log.Printf("Creating a DB instance named %v and database %v.\n"+
  "The DB instance is configured to use your custom parameter group %v,\n"+
  "selected engine %v,\n"+
  "selected DB instance class %v,"+
  "and %v GiB of %v storage.\n"+
  "This typically takes several minutes.",
  instanceName, dbName, *parameterGroup.DBParameterGroupName,
*engineSelection.EngineVersion,
  optChoices[optIndex], allocatedStorage, storageType)
 instance, err = scenario.instances.CreateInstance(
  ctx, instanceName, dbName, *engineSelection.Engine,
*engineSelection.EngineVersion,
  *parameterGroup.DBParameterGroupName, optChoices[optIndex], storageType,
  allocatedStorage, adminUsername, adminPassword)
 if err != nil {
  panic(err)
 }
 for *instance.DBInstanceStatus != "available" {
  scenario.helper.Pause(30)
  instance, err = scenario.instances.GetInstance(ctx, instanceName)
  if err != nil {
   panic(err)
  }
 }
 log.Println("Instance created and available.")
}
log.Println("Instance data:")
```

```
 log.Printf("\tDBInstanceIdentifier: %v\n", *instance.DBInstanceIdentifier)
 log.Printf("\tARN: %v\n", *instance.DBInstanceArn)
 log.Printf("\tStatus: %v\n", *instance.DBInstanceStatus)
 log.Printf("\tEngine: %v\n", *instance.Engine)
 log.Printf("\tEngine version: %v\n", *instance.EngineVersion)
 log.Println(strings.Repeat("-", 88))
 return instance
}

// DisplayConnection displays connection information about a DB instance and tips
// on how to connect to it.
func (scenario GetStartedInstances) DisplayConnection(instance *types.DBInstance) {
 log.Println(
  "You can now connect to your database by using your favorite MySQL client.\n" +
   "One way to connect is by using the 'mysql' shell on an Amazon EC2 instance\n" +
   "that is running in the same VPC as your DB instance. Pass the endpoint,\n" +
   "port, and administrator username to 'mysql'. Then, enter your password\n" +
   "when prompted:")
 log.Printf("\n\tmysql -h %v -P %v -u %v -p\n",
  *instance.Endpoint.Address, instance.Endpoint.Port, *instance.MasterUsername)
 log.Println("For more information, see the User Guide for RDS:\n" +
  "\thttps://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/
CHAP_GettingStarted.CreatingConnecting.MySQL.html#CHAP_GettingStarted.Connecting.MySQL")
 log.Println(strings.Repeat("-", 88))
}

// CreateSnapshot shows how to create a DB instance snapshot and wait until it's
 available.
func (scenario GetStartedInstances) CreateSnapshot(ctx context.Context, instance
 *types.DBInstance) {
 if scenario.questioner.AskBool(
  "Do you want to create a snapshot of your DB instance (y/n)? ", "y") {
  snapshotId := fmt.Sprintf("%v-%v", *instance.DBInstanceIdentifier,
 scenario.helper.UniqueId())
  log.Printf("Creating a snapshot named %v. This typically takes a few minutes.\n",
 snapshotId)
  snapshot, err := scenario.instances.CreateSnapshot(ctx,
 *instance.DBInstanceIdentifier, snapshotId)
  if err != nil {
   panic(err)
  }
  for *snapshot.Status != "available" {
   scenario.helper.Pause(30)
   snapshot, err = scenario.instances.GetSnapshot(ctx, snapshotId)
```

```
   if err != nil {
    panic(err)
   }
  }
  log.Println("Snapshot data:")
  log.Printf("\tDBSnapshotIdentifier: %v\n", *snapshot.DBSnapshotIdentifier)
  log.Printf("\tARN: %v\n", *snapshot.DBSnapshotArn)
  log.Printf("\tStatus: %v\n", *snapshot.Status)
  log.Printf("\tEngine: %v\n", *snapshot.Engine)
  log.Printf("\tEngine version: %v\n", *snapshot.EngineVersion)
  log.Printf("\tDBInstanceIdentifier: %v\n", *snapshot.DBInstanceIdentifier)
  log.Printf("\tSnapshotCreateTime: %v\n", *snapshot.SnapshotCreateTime)
  log.Println(strings.Repeat("-", 88))
 }
}

// Cleanup shows how to clean up a DB instance and DB parameter group.
// Before the DB parameter group can be deleted, all associated DB instances must
 first be deleted.
func (scenario GetStartedInstances) Cleanup(
 ctx context.Context, instance *types.DBInstance, parameterGroup
 *types.DBParameterGroup) {

 if scenario.questioner.AskBool(
  "\nDo you want to delete the database instance and parameter group (y/n)? ", "y")
 {
  log.Printf("Deleting database instance %v.\n", *instance.DBInstanceIdentifier)
  err := scenario.instances.DeleteInstance(ctx, *instance.DBInstanceIdentifier)
  if err != nil {
   panic(err)
  }
  log.Println(
   "Waiting for the DB instance to delete. This typically takes several minutes.")
  for instance != nil {
   scenario.helper.Pause(30)
   instance, err = scenario.instances.GetInstance(ctx,
 *instance.DBInstanceIdentifier)
   if err != nil {
    panic(err)
   }
  }
  log.Printf("Deleting parameter group %v.", *parameterGroup.DBParameterGroupName)
  err = scenario.instances.DeleteParameterGroup(ctx,
 *parameterGroup.DBParameterGroupName)
```

```
  if err != nil {
   panic(err)
  }
 }
}

// IScenarioHelper abstracts the function from a scenario so that it
// can be mocked for unit testing.
type IScenarioHelper interface {
 Pause(secs int)
 UniqueId() string
}
type ScenarioHelper struct{}

// Pause waits for the specified number of seconds.
func (helper ScenarioHelper) Pause(secs int) {
 time.Sleep(time.Duration(secs) * time.Second)
}

// UniqueId returns a new UUID.
func (helper ScenarioHelper) UniqueId() string {
 return uuid.New().String()
}
```

Definieren Sie Funktionen, die vom Szenario aufgerufen werden, um Amazon-RDS-Aktionen zu verwalten.

```
import (
 "context"
 "errors"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/rds"
 "github.com/aws/aws-sdk-go-v2/service/rds/types"
)

type DbInstances struct {
 RdsClient *rds.Client
}
```

```
// GetParameterGroup gets a DB parameter group by name.
func (instances *DbInstances) GetParameterGroup(ctx context.Context,
 parameterGroupName string) (
 *types.DBParameterGroup, error) {
 output, err := instances.RdsClient.DescribeDBParameterGroups(
  ctx, &rds.DescribeDBParameterGroupsInput{
   DBParameterGroupName: aws.String(parameterGroupName),
  })
 if err != nil {
  var notFoundError *types.DBParameterGroupNotFoundFault
  if errors.As(err, &notFoundError) {
   log.Printf("Parameter group %v does not exist.\n", parameterGroupName)
   err = nil
  } else {
   log.Printf("Error getting parameter group %v: %v\n", parameterGroupName, err)
  }
  return nil, err
 } else {
  return &output.DBParameterGroups[0], err
 }
}



// CreateParameterGroup creates a DB parameter group that is based on the specified
// parameter group family.
func (instances *DbInstances) CreateParameterGroup(
 ctx context.Context, parameterGroupName string, parameterGroupFamily string,
 description string) (
 *types.DBParameterGroup, error) {

 output, err := instances.RdsClient.CreateDBParameterGroup(ctx,
  &rds.CreateDBParameterGroupInput{
   DBParameterGroupName:   aws.String(parameterGroupName),
   DBParameterGroupFamily: aws.String(parameterGroupFamily),
   Description:            aws.String(description),
  })
 if err != nil {
  log.Printf("Couldn't create parameter group %v: %v\n", parameterGroupName, err)
  return nil, err
 } else {
  return output.DBParameterGroup, err
```

```
 }
}



// DeleteParameterGroup deletes the named DB parameter group.
func (instances *DbInstances) DeleteParameterGroup(ctx context.Context,
 parameterGroupName string) error {
 _, err := instances.RdsClient.DeleteDBParameterGroup(ctx,
  &rds.DeleteDBParameterGroupInput{
   DBParameterGroupName: aws.String(parameterGroupName),
  })
 if err != nil {
  log.Printf("Couldn't delete parameter group %v: %v\n", parameterGroupName, err)
  return err
 } else {
  return nil
 }
}



// GetParameters gets the parameters that are contained in a DB parameter group.
func (instances *DbInstances) GetParameters(ctx context.Context, parameterGroupName
 string, source string) (
 []types.Parameter, error) {

 var output *rds.DescribeDBParametersOutput
 var params []types.Parameter
 var err error
 parameterPaginator := rds.NewDescribeDBParametersPaginator(instances.RdsClient,
  &rds.DescribeDBParametersInput{
   DBParameterGroupName: aws.String(parameterGroupName),
   Source:               aws.String(source),
  })
 for parameterPaginator.HasMorePages() {
  output, err = parameterPaginator.NextPage(ctx)
  if err != nil {
   log.Printf("Couldn't get parameters for %v: %v\n", parameterGroupName, err)
   break
  } else {
   params = append(params, output.Parameters...)
  }
 }
```

```
 return params, err
}



// UpdateParameters updates parameters in a named DB parameter group.
func (instances *DbInstances) UpdateParameters(ctx context.Context,
 parameterGroupName string, params []types.Parameter) error {
 _, err := instances.RdsClient.ModifyDBParameterGroup(ctx,
  &rds.ModifyDBParameterGroupInput{
   DBParameterGroupName: aws.String(parameterGroupName),
   Parameters:           params,
  })
 if err != nil {
  log.Printf("Couldn't update parameters in %v: %v\n", parameterGroupName, err)
  return err
 } else {
  return nil
 }
}



// CreateSnapshot creates a snapshot of a DB instance.
func (instances *DbInstances) CreateSnapshot(ctx context.Context, instanceName
 string, snapshotName string) (
 *types.DBSnapshot, error) {
 output, err := instances.RdsClient.CreateDBSnapshot(ctx,
 &rds.CreateDBSnapshotInput{
  DBInstanceIdentifier: aws.String(instanceName),
  DBSnapshotIdentifier: aws.String(snapshotName),
 })
 if err != nil {
  log.Printf("Couldn't create snapshot %v: %v\n", snapshotName, err)
  return nil, err
 } else {
  return output.DBSnapshot, nil
 }
}



// GetSnapshot gets a DB instance snapshot.
```

```
func (instances *DbInstances) GetSnapshot(ctx context.Context, snapshotName string)
 (*types.DBSnapshot, error) {
 output, err := instances.RdsClient.DescribeDBSnapshots(ctx,
  &rds.DescribeDBSnapshotsInput{
   DBSnapshotIdentifier: aws.String(snapshotName),
  })
 if err != nil {
  log.Printf("Couldn't get snapshot %v: %v\n", snapshotName, err)
  return nil, err
 } else {
  return &output.DBSnapshots[0], nil
 }
}



// CreateInstance creates a DB instance.
func (instances *DbInstances) CreateInstance(ctx context.Context, instanceName
 string, dbName string,
 dbEngine string, dbEngineVersion string, parameterGroupName string, dbInstanceClass
 string,
 storageType string, allocatedStorage int32, adminName string, adminPassword string)
 (
 *types.DBInstance, error) {
 output, err := instances.RdsClient.CreateDBInstance(ctx,
 &rds.CreateDBInstanceInput{
  DBInstanceIdentifier: aws.String(instanceName),
  DBName:               aws.String(dbName),
  DBParameterGroupName: aws.String(parameterGroupName),
  Engine:               aws.String(dbEngine),
  EngineVersion:        aws.String(dbEngineVersion),
  DBInstanceClass:      aws.String(dbInstanceClass),
  StorageType:          aws.String(storageType),
  AllocatedStorage:     aws.Int32(allocatedStorage),
  MasterUsername:       aws.String(adminName),
  MasterUserPassword:   aws.String(adminPassword),
 })
 if err != nil {
  log.Printf("Couldn't create instance %v: %v\n", instanceName, err)
  return nil, err
 } else {
  return output.DBInstance, nil
 }
}
```

```
// GetInstance gets data about a DB instance.
func (instances *DbInstances) GetInstance(ctx context.Context, instanceName string)
 (
 *types.DBInstance, error) {
 output, err := instances.RdsClient.DescribeDBInstances(ctx,
  &rds.DescribeDBInstancesInput{
   DBInstanceIdentifier: aws.String(instanceName),
  })
 if err != nil {
  var notFoundError *types.DBInstanceNotFoundFault
  if errors.As(err, &notFoundError) {
   log.Printf("DB instance %v does not exist.\n", instanceName)
   err = nil
  } else {
   log.Printf("Couldn't get instance %v: %v\n", instanceName, err)
  }
  return nil, err
 } else {
  return &output.DBInstances[0], nil
 }
}



// DeleteInstance deletes a DB instance.
func (instances *DbInstances) DeleteInstance(ctx context.Context, instanceName
 string) error {
 _, err := instances.RdsClient.DeleteDBInstance(ctx, &rds.DeleteDBInstanceInput{
  DBInstanceIdentifier:   aws.String(instanceName),
  SkipFinalSnapshot:      aws.Bool(true),
  DeleteAutomatedBackups: aws.Bool(true),
 })
 if err != nil {
  log.Printf("Couldn't delete instance %v: %v\n", instanceName, err)
  return err
 } else {
  return nil
 }
}
```

```
// GetEngineVersions gets database engine versions that are available for the
 specified engine
// and parameter group family.
func (instances *DbInstances) GetEngineVersions(ctx context.Context, engine string,
 parameterGroupFamily string) (
 []types.DBEngineVersion, error) {
 output, err := instances.RdsClient.DescribeDBEngineVersions(ctx,
  &rds.DescribeDBEngineVersionsInput{
   Engine:                 aws.String(engine),
   DBParameterGroupFamily: aws.String(parameterGroupFamily),
  })
 if err != nil {
  log.Printf("Couldn't get engine versions for %v: %v\n", engine, err)
  return nil, err
 } else {
  return output.DBEngineVersions, nil
 }
}



// GetOrderableInstances uses a paginator to get DB instance options that can be
 used to create DB instances that are
// compatible with a set of specifications.
func (instances *DbInstances) GetOrderableInstances(ctx context.Context, engine
 string, engineVersion string) (
 []types.OrderableDBInstanceOption, error) {

 var output *rds.DescribeOrderableDBInstanceOptionsOutput
 var instanceOptions []types.OrderableDBInstanceOption
 var err error
 orderablePaginator :=
 rds.NewDescribeOrderableDBInstanceOptionsPaginator(instances.RdsClient,
  &rds.DescribeOrderableDBInstanceOptionsInput{
   Engine:        aws.String(engine),
   EngineVersion: aws.String(engineVersion),
  })
 for orderablePaginator.HasMorePages() {
  output, err = orderablePaginator.NextPage(ctx)
  if err != nil {
   log.Printf("Couldn't get orderable DB instance options: %v\n", err)
   break
  } else {
```

```
    instanceOptions = append(instanceOptions, output.OrderableDBInstanceOptions...)
   }
  }
  return instanceOptions, err
}
```

- API-Details finden Sie in den folgenden Themen der AWS SDK für Go -API-Referenz.
  - CreateDBInstance
  - DBParameterGruppe erstellen
  - CreateDBSnapshot
  - LöschenDBInstance
  - DBParameterGruppe löschen
  - DBEngineVersionen beschreiben
  - Beschreiben DBInstances
  - Beschreiben Sie DBParameter Gruppen
  - Beschreiben DBParameters
  - Beschreiben DBSnapshots
  - DescribeOrderableDBInstanceOptionen
  - DBParameterGruppe ändern

Aktionen

`CreateDBInstance`

Das folgende Codebeispiel zeigt die Verwendung`CreateDBInstance`.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import (
 "context"
 "errors"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/rds"
 "github.com/aws/aws-sdk-go-v2/service/rds/types"
)

type DbInstances struct {
 RdsClient *rds.Client
}



// CreateInstance creates a DB instance.
func (instances *DbInstances) CreateInstance(ctx context.Context, instanceName
 string, dbName string,
 dbEngine string, dbEngineVersion string, parameterGroupName string, dbInstanceClass
 string,
 storageType string, allocatedStorage int32, adminName string, adminPassword string)
 (
 *types.DBInstance, error) {
 output, err := instances.RdsClient.CreateDBInstance(ctx,
 &rds.CreateDBInstanceInput{
  DBInstanceIdentifier: aws.String(instanceName),
  DBName:               aws.String(dbName),
  DBParameterGroupName: aws.String(parameterGroupName),
  Engine:               aws.String(dbEngine),
  EngineVersion:        aws.String(dbEngineVersion),
  DBInstanceClass:      aws.String(dbInstanceClass),
  StorageType:          aws.String(storageType),
  AllocatedStorage:     aws.Int32(allocatedStorage),
  MasterUsername:       aws.String(adminName),
  MasterUserPassword:   aws.String(adminPassword),
 })
 if err != nil {
  log.Printf("Couldn't create instance %v: %v\n", instanceName, err)
  return nil, err
 } else {
  return output.DBInstance, nil
```

```
 }
}
```

- Einzelheiten zur API finden Sie unter DBInstance In der AWS SDK für Go API-Referenz erstellen.

`CreateDBParameterGroup`

Das folgende Codebeispiel zeigt die Verwendung`CreateDBParameterGroup`.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import (
 "context"
 "errors"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/rds"
 "github.com/aws/aws-sdk-go-v2/service/rds/types"
)

type DbInstances struct {
 RdsClient *rds.Client
}



// CreateParameterGroup creates a DB parameter group that is based on the specified
// parameter group family.
func (instances *DbInstances) CreateParameterGroup(
```

```
 ctx context.Context, parameterGroupName string, parameterGroupFamily string,
 description string) (
 *types.DBParameterGroup, error) {

 output, err := instances.RdsClient.CreateDBParameterGroup(ctx,
  &rds.CreateDBParameterGroupInput{
   DBParameterGroupName:   aws.String(parameterGroupName),
   DBParameterGroupFamily: aws.String(parameterGroupFamily),
   Description:            aws.String(description),
  })
 if err != nil {
  log.Printf("Couldn't create parameter group %v: %v\n", parameterGroupName, err)
  return nil, err
 } else {
  return output.DBParameterGroup, err
 }
}
```

- Einzelheiten zur API finden Sie unter DBParameterGruppe erstellen in der AWS SDK für Go API-Referenz.

`CreateDBSnapshot`

Das folgende Codebeispiel zeigt die Verwendung`CreateDBSnapshot`.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import (
 "context"
 "errors"
 "log"
```

```
 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/rds"
 "github.com/aws/aws-sdk-go-v2/service/rds/types"
)

type DbInstances struct {
 RdsClient *rds.Client
}



// CreateSnapshot creates a snapshot of a DB instance.
func (instances *DbInstances) CreateSnapshot(ctx context.Context, instanceName
 string, snapshotName string) (
 *types.DBSnapshot, error) {
 output, err := instances.RdsClient.CreateDBSnapshot(ctx,
 &rds.CreateDBSnapshotInput{
  DBInstanceIdentifier: aws.String(instanceName),
  DBSnapshotIdentifier: aws.String(snapshotName),
 })
 if err != nil {
  log.Printf("Couldn't create snapshot %v: %v\n", snapshotName, err)
  return nil, err
 } else {
  return output.DBSnapshot, nil
 }
}
```

- Einzelheiten zur API finden Sie unter DBSnapshot In der AWS SDK für Go API-Referenz erstellen.

`DeleteDBInstance`

Das folgende Codebeispiel zeigt die Verwendung`DeleteDBInstance`.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import (
 "context"
 "errors"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/rds"
 "github.com/aws/aws-sdk-go-v2/service/rds/types"
)

type DbInstances struct {
 RdsClient *rds.Client
}



// DeleteInstance deletes a DB instance.
func (instances *DbInstances) DeleteInstance(ctx context.Context, instanceName
 string) error {
 _, err := instances.RdsClient.DeleteDBInstance(ctx, &rds.DeleteDBInstanceInput{
  DBInstanceIdentifier:   aws.String(instanceName),
  SkipFinalSnapshot:      aws.Bool(true),
  DeleteAutomatedBackups: aws.Bool(true),
 })
 if err != nil {
  log.Printf("Couldn't delete instance %v: %v\n", instanceName, err)
  return err
 } else {
  return nil
 }
}
```

- Einzelheiten zur API finden Sie unter Löschen DBInstance in der AWS SDK für Go API-Referenz.

`DeleteDBParameterGroup`

Das folgende Codebeispiel zeigt die Verwendung`DeleteDBParameterGroup`.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import (
 "context"
 "errors"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/rds"
 "github.com/aws/aws-sdk-go-v2/service/rds/types"
)

type DbInstances struct {
 RdsClient *rds.Client
}



// DeleteParameterGroup deletes the named DB parameter group.
func (instances *DbInstances) DeleteParameterGroup(ctx context.Context,
 parameterGroupName string) error {
 _, err := instances.RdsClient.DeleteDBParameterGroup(ctx,
  &rds.DeleteDBParameterGroupInput{
   DBParameterGroupName: aws.String(parameterGroupName),
  })
 if err != nil {
  log.Printf("Couldn't delete parameter group %v: %v\n", parameterGroupName, err)
  return err
```

```
 } else {
  return nil
 }
}
```

- Einzelheiten zur API finden Sie unter DBParameterGruppe löschen in der AWS SDK für Go API-Referenz.

`DescribeDBEngineVersions`

Das folgende Codebeispiel zeigt die Verwendung`DescribeDBEngineVersions`.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import (
 "context"
 "errors"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/rds"
 "github.com/aws/aws-sdk-go-v2/service/rds/types"
)

type DbInstances struct {
 RdsClient *rds.Client
}



// GetEngineVersions gets database engine versions that are available for the
 specified engine
```

```
// and parameter group family.
func (instances *DbInstances) GetEngineVersions(ctx context.Context, engine string,
 parameterGroupFamily string) (
 []types.DBEngineVersion, error) {
 output, err := instances.RdsClient.DescribeDBEngineVersions(ctx,
  &rds.DescribeDBEngineVersionsInput{
   Engine:                 aws.String(engine),
   DBParameterGroupFamily: aws.String(parameterGroupFamily),
  })
 if err != nil {
  log.Printf("Couldn't get engine versions for %v: %v\n", engine, err)
  return nil, err
 } else {
  return output.DBEngineVersions, nil
 }
}
```

- Einzelheiten zur API finden Sie unter DBEngineVersionen beschreiben in der AWS SDK für Go API-Referenz.

`DescribeDBInstances`

Das folgende Codebeispiel zeigt die Verwendung`DescribeDBInstances`.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import (
 "context"
 "errors"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
```

```
 "github.com/aws/aws-sdk-go-v2/service/rds"
 "github.com/aws/aws-sdk-go-v2/service/rds/types"
)

type DbInstances struct {
 RdsClient *rds.Client
}



// GetInstance gets data about a DB instance.
func (instances *DbInstances) GetInstance(ctx context.Context, instanceName string)
 (
 *types.DBInstance, error) {
 output, err := instances.RdsClient.DescribeDBInstances(ctx,
  &rds.DescribeDBInstancesInput{
   DBInstanceIdentifier: aws.String(instanceName),
  })
 if err != nil {
  var notFoundError *types.DBInstanceNotFoundFault
  if errors.As(err, &notFoundError) {
   log.Printf("DB instance %v does not exist.\n", instanceName)
   err = nil
  } else {
   log.Printf("Couldn't get instance %v: %v\n", instanceName, err)
  }
  return nil, err
 } else {
  return &output.DBInstances[0], nil
 }
}
```

- Einzelheiten zur API finden Sie unter Describe DBInstances in der AWS SDK für Go API-Referenz.

`DescribeDBParameterGroups`

Das folgende Codebeispiel zeigt die Verwendung`DescribeDBParameterGroups`.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import (
 "context"
 "errors"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/rds"
 "github.com/aws/aws-sdk-go-v2/service/rds/types"
)

type DbInstances struct {
 RdsClient *rds.Client
}



// GetParameterGroup gets a DB parameter group by name.
func (instances *DbInstances) GetParameterGroup(ctx context.Context,
 parameterGroupName string) (
 *types.DBParameterGroup, error) {
 output, err := instances.RdsClient.DescribeDBParameterGroups(
  ctx, &rds.DescribeDBParameterGroupsInput{
   DBParameterGroupName: aws.String(parameterGroupName),
  })
 if err != nil {
  var notFoundError *types.DBParameterGroupNotFoundFault
  if errors.As(err, &notFoundError) {
   log.Printf("Parameter group %v does not exist.\n", parameterGroupName)
   err = nil
  } else {
   log.Printf("Error getting parameter group %v: %v\n", parameterGroupName, err)
  }
  return nil, err
```

```
 } else {
  return &output.DBParameterGroups[0], err
 }
}
```

- Einzelheiten zur API finden Sie unter DBParameterGruppen beschreiben in der AWS SDK für Go API-Referenz.

`DescribeDBParameters`

Das folgende Codebeispiel zeigt die Verwendung`DescribeDBParameters`.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import (
 "context"
 "errors"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/rds"
 "github.com/aws/aws-sdk-go-v2/service/rds/types"
)

type DbInstances struct {
 RdsClient *rds.Client
}



// GetParameters gets the parameters that are contained in a DB parameter group.
```

```
func (instances *DbInstances) GetParameters(ctx context.Context, parameterGroupName
 string, source string) (
 []types.Parameter, error) {

 var output *rds.DescribeDBParametersOutput
 var params []types.Parameter
 var err error
 parameterPaginator := rds.NewDescribeDBParametersPaginator(instances.RdsClient,
  &rds.DescribeDBParametersInput{
   DBParameterGroupName: aws.String(parameterGroupName),
   Source:               aws.String(source),
  })
 for parameterPaginator.HasMorePages() {
  output, err = parameterPaginator.NextPage(ctx)
  if err != nil {
   log.Printf("Couldn't get parameters for %v: %v\n", parameterGroupName, err)
   break
  } else {
   params = append(params, output.Parameters...)
  }
 }
 return params, err
}
```

- Einzelheiten zur API finden Sie unter Describe DBParameters in der AWS SDK für Go API-Referenz.

`DescribeDBSnapshots`

Das folgende Codebeispiel zeigt die Verwendung`DescribeDBSnapshots`.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import (
 "context"
 "errors"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/rds"
 "github.com/aws/aws-sdk-go-v2/service/rds/types"
)

type DbInstances struct {
 RdsClient *rds.Client
}



// GetSnapshot gets a DB instance snapshot.
func (instances *DbInstances) GetSnapshot(ctx context.Context, snapshotName string)
 (*types.DBSnapshot, error) {
 output, err := instances.RdsClient.DescribeDBSnapshots(ctx,
  &rds.DescribeDBSnapshotsInput{
   DBSnapshotIdentifier: aws.String(snapshotName),
  })
 if err != nil {
  log.Printf("Couldn't get snapshot %v: %v\n", snapshotName, err)
  return nil, err
 } else {
  return &output.DBSnapshots[0], nil
 }
}
```

- Einzelheiten zur API finden Sie unter Describe DBSnapshots in der AWS SDK für Go API-Referenz.

`DescribeOrderableDBInstanceOptions`

Das folgende Codebeispiel zeigt die Verwendung`DescribeOrderableDBInstanceOptions`.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import (
 "context"
 "errors"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/rds"
 "github.com/aws/aws-sdk-go-v2/service/rds/types"
)

type DbInstances struct {
 RdsClient *rds.Client
}



// GetOrderableInstances uses a paginator to get DB instance options that can be
 used to create DB instances that are
// compatible with a set of specifications.
func (instances *DbInstances) GetOrderableInstances(ctx context.Context, engine
 string, engineVersion string) (
 []types.OrderableDBInstanceOption, error) {

 var output *rds.DescribeOrderableDBInstanceOptionsOutput
 var instanceOptions []types.OrderableDBInstanceOption
 var err error
 orderablePaginator :=
 rds.NewDescribeOrderableDBInstanceOptionsPaginator(instances.RdsClient,
  &rds.DescribeOrderableDBInstanceOptionsInput{
   Engine:        aws.String(engine),
   EngineVersion: aws.String(engineVersion),
  })
 for orderablePaginator.HasMorePages() {
```

```
  output, err = orderablePaginator.NextPage(ctx)
  if err != nil {
   log.Printf("Couldn't get orderable DB instance options: %v\n", err)
   break
  } else {
   instanceOptions = append(instanceOptions, output.OrderableDBInstanceOptions...)
  }
 }
 return instanceOptions, err
}
```

- Einzelheiten zur API finden Sie unter DescribeOrderableDBInstanceOptionen in der AWS SDK für Go API-Referenz.

`ModifyDBParameterGroup`

Das folgende Codebeispiel zeigt die Verwendung`ModifyDBParameterGroup`.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import (
 "context"
 "errors"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/rds"
 "github.com/aws/aws-sdk-go-v2/service/rds/types"
)

type DbInstances struct {
 RdsClient *rds.Client
```

```
}



// UpdateParameters updates parameters in a named DB parameter group.
func (instances *DbInstances) UpdateParameters(ctx context.Context,
 parameterGroupName string, params []types.Parameter) error {
 _, err := instances.RdsClient.ModifyDBParameterGroup(ctx,
  &rds.ModifyDBParameterGroupInput{
   DBParameterGroupName: aws.String(parameterGroupName),
   Parameters:           params,
  })
 if err != nil {
  log.Printf("Couldn't update parameters in %v: %v\n", parameterGroupName, err)
  return err
 } else {
  return nil
 }
}
```

- Einzelheiten zur API finden Sie unter DBParameterGruppe ändern in der AWS SDK für Go API-Referenz.

Serverless-Beispiele

Herstellen einer Verbindung mit einer Amazon-RDS-Datenbank in einer Lambda-Funktion

Das folgende Codebeispiel zeigt, wie eine Lambda-Funktion implementiert wird, die eine Verbindung zu einer RDS-Datenbank herstellt. Die Funktion stellt eine einfache Datenbankanfrage und gibt das Ergebnis zurück.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu. GitHub Das vollständige Beispiel sowie eine Anleitung zum Einrichten und Ausführen finden Sie im Repository mit Serverless-Beispielen.

Herstellen einer Verbindung zu einer Amazon RDS-Datenbank in einer Lambda-Funktion mit Go.

```
/*
Golang v2 code here.
*/

package main

import (
 "context"
 "database/sql"
 "encoding/json"
 "fmt"
 "os"

 "github.com/aws/aws-lambda-go/lambda"
 "github.com/aws/aws-sdk-go-v2/config"
 "github.com/aws/aws-sdk-go-v2/feature/rds/auth"
 _ "github.com/go-sql-driver/mysql"
)

type MyEvent struct {
 Name string `json:"name"`
}

func HandleRequest(event *MyEvent) (map[string]interface{}, error) {

 var dbName string = os.Getenv("DatabaseName")
 var dbUser string = os.Getenv("DatabaseUser")
 var dbHost string = os.Getenv("DBHost") // Add hostname without https
 var dbPort int = os.Getenv("Port")      // Add port number
 var dbEndpoint string = fmt.Sprintf("%s:%d", dbHost, dbPort)
 var region string = os.Getenv("AWS_REGION")

 cfg, err := config.LoadDefaultConfig(context.TODO())
 if err != nil {
  panic("configuration error: " + err.Error())
 }

 authenticationToken, err := auth.BuildAuthToken(
  context.TODO(), dbEndpoint, region, dbUser, cfg.Credentials)
 if err != nil {
  panic("failed to create authentication token: " + err.Error())
 }
```

```
 dsn := fmt.Sprintf("%s:%s@tcp(%s)/%s?tls=true&allowCleartextPasswords=true",
  dbUser, authenticationToken, dbEndpoint, dbName,
 )

 db, err := sql.Open("mysql", dsn)
 if err != nil {
  panic(err)
 }

 defer db.Close()

 var sum int
 err = db.QueryRow("SELECT ?+? AS sum", 3, 2).Scan(&sum)
 if err != nil {
  panic(err)
 }
 s := fmt.Sprint(sum)
 message := fmt.Sprintf("The selected sum is: %s", s)

 messageBytes, err := json.Marshal(message)
 if err != nil {
  return nil, err
 }

 messageString := string(messageBytes)
 return map[string]interface{}{
  "statusCode": 200,
  "headers":    map[string]string{"Content-Type": "application/json"},
  "body":       messageString,
 }, nil
}

func main() {
 lambda.Start(HandleRequest)
}
```

Amazon Redshift Redshift-Beispiele mit SDK for Go V2

Die folgenden Codebeispiele zeigen Ihnen, wie Sie mithilfe von AWS SDK für Go V2 mit Amazon Redshift Aktionen ausführen und allgemeine Szenarien implementieren.

Bei Grundlagen handelt es sich um Code-Beispiele, die Ihnen zeigen, wie Sie die wesentlichen Vorgänge innerhalb eines Services ausführen.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Service-Funktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarios anzeigen.

Jedes Beispiel enthält einen Link zum vollständigen Quellcode, in dem Sie Anweisungen zur Einrichtung und Ausführung des Codes im Kontext finden.

Erste Schritte

Hallo Amazon Redshift

Die folgenden Codebeispiele zeigen, wie Sie mit Amazon Redshift beginnen können.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
package main

import (
 "context"
 "fmt"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/config"
 "github.com/aws/aws-sdk-go-v2/service/redshift"
)

// main uses the AWS SDK for Go V2 to create a Redshift client
// and list up to 10 clusters in your account.
// This example uses the default settings specified in your shared credentials
// and config files.
func main() {
```

```
 ctx := context.Background()
 sdkConfig, err := config.LoadDefaultConfig(ctx)
 if err != nil {
  fmt.Println("Couldn't load default configuration. Have you set up your AWS
 account?")
  fmt.Println(err)
  return
 }
 redshiftClient := redshift.NewFromConfig(sdkConfig)
 count := 20
 fmt.Printf("Let's list up to %v clusters for your account.\n", count)
 result, err := redshiftClient.DescribeClusters(ctx,
 &redshift.DescribeClustersInput{
  MaxRecords: aws.Int32(int32(count)),
 })
 if err != nil {
  fmt.Printf("Couldn't list clusters for your account. Here's why: %v\n", err)
  return
 }
 if len(result.Clusters) == 0 {
  fmt.Println("You don't have any clusters!")
  return
 }
 for _, cluster := range result.Clusters {
  fmt.Printf("\t%v : %v\n", *cluster.ClusterIdentifier, *cluster.ClusterStatus)
 }
}
```

- Einzelheiten zur API finden Sie DescribeClustersin der AWS SDK für Go API-Referenz.

Themen

- Grundlagen
- Aktionen

Grundlagen

Erlernen der Grundlagen

Wie das aussehen kann, sehen Sie am nachfolgenden Beispielcode:

- Erstellen Sie einen Redshift-Cluster.
- Listet die Datenbanken im Cluster auf.
- Erstellen Sie eine Tabelle mit dem Namen Movies.
- Füllen Sie die Tabelle Filme aus.
- Fragen Sie die Tabelle Filme nach Jahr ab.
- Ändern Sie den Redshift-Cluster.
- Löschen Sie den Amazon Redshift Redshift-Cluster.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
package scenarios

import (
 "context"
 "encoding/json"
 "errors"
 "fmt"
 "log"
 "math/rand"
 "strings"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 redshift_types "github.com/aws/aws-sdk-go-v2/service/redshift/types"
 redshiftdata_types "github.com/aws/aws-sdk-go-v2/service/redshiftdata/types"
 "github.com/aws/aws-sdk-go-v2/service/secretsmanager"
 "github.com/awsdocs/aws-doc-sdk-examples/gov2/demotools"
 "github.com/awsdocs/aws-doc-sdk-examples/gov2/redshift/actions"

 "github.com/aws/aws-sdk-go-v2/service/redshift"
 "github.com/aws/aws-sdk-go-v2/service/redshiftdata"
)
```

```
// IScenarioHelper abstracts input and wait functions from a scenario so that they
// can be mocked for unit testing.
type IScenarioHelper interface {
 GetName() string
}

const rMax = 100000

type ScenarioHelper struct {
 Prefix string
 Random *rand.Rand
}

// GetName returns a unique name formed of a prefix and a random number.
func (helper ScenarioHelper) GetName() string {
 return fmt.Sprintf("%v%v", helper.Prefix, helper.Random.Intn(rMax))
}

// RedshiftBasicsScenario separates the steps of this scenario into individual
 functions so that
// they are simpler to read and understand.
type RedshiftBasicsScenario struct {
 sdkConfig         aws.Config
 helper            IScenarioHelper
 questioner        demotools.IQuestioner
 pauser            demotools.IPausable
 filesystem        demotools.IFileSystem
 redshiftActor     *actions.RedshiftActions
 redshiftDataActor *actions.RedshiftDataActions
 secretsmanager    *SecretsManager
}

// SecretsManager is used to retrieve username and password information from a
 secure service.
type SecretsManager struct {
 SecretsManagerClient *secretsmanager.Client
}

// RedshiftBasics constructs a new Redshift Basics runner.
func RedshiftBasics(sdkConfig aws.Config, questioner demotools.IQuestioner, pauser
 demotools.IPausable, filesystem demotools.IFileSystem, helper IScenarioHelper)
 RedshiftBasicsScenario {
 scenario := RedshiftBasicsScenario{
```

```
  sdkConfig:         sdkConfig,
  helper:            helper,
  questioner:        questioner,
  pauser:            pauser,
  filesystem:        filesystem,
  secretsmanager:    &SecretsManager{SecretsManagerClient:
 secretsmanager.NewFromConfig(sdkConfig)},
  redshiftActor:     &actions.RedshiftActions{RedshiftClient:
 redshift.NewFromConfig(sdkConfig)},
  redshiftDataActor: &actions.RedshiftDataActions{RedshiftDataClient:
 redshiftdata.NewFromConfig(sdkConfig)},
 }
 return scenario
}


// Movie makes it easier to use Movie objects given in json format.
type Movie struct {
 ID    int    `json:"id"`
 Title string `json:"title"`
 Year  int    `json:"year"`
}


// User makes it easier to get the User data back from SecretsManager and use it
 later.
type User struct {
 Username string `json:"userName"`
 Password string `json:"userPassword"`
}

// Run runs the RedshiftBasics interactive example that shows you how to use Amazon
// Redshift and how to interact with its common endpoints.
//
// 0. Retrieve username and password information to access Redshift.
// 1. Create a cluster.
// 2. Wait for the cluster to become available.
// 3. List the available databases in the region.
// 4. Create a table named "Movies" in the "dev" database.
// 5. Populate the movies table from the "movies.json" file.
// 6. Query the movies table by year.
// 7. Modify the cluster's maintenance window.
// 8. Optionally clean up all resources created during this demo.
//
```

```
// This example creates an Amazon Redshift service client from the specified
 sdkConfig so that
// you can replace it with a mocked or stubbed config for unit testing.
//
// It uses a questioner from the `demotools` package to get input during the
 example.
// This package can be found in the ..\..\demotools folder of this repo.
func (runner *RedshiftBasicsScenario) Run(ctx context.Context) {

 user := User{}
 secretId := "s3express/basics/secrets"
 clusterId := "demo-cluster-1"
 maintenanceWindow := "wed:07:30-wed:08:00"
 databaseName := "dev"
 tableName := "Movies"
 fileName := "Movies.json"
 nodeType := "ra3.xlplus"
 clusterType := "single-node"

 defer func() {
  if r := recover(); r != nil {
   log.Println("Something went wrong with the demo.")
   _, isMock := runner.questioner.(*demotools.MockQuestioner)
   if isMock || runner.questioner.AskBool("Do you want to see the full error message
 (y/n)?", "y") {
    log.Println(r)
   }
   runner.cleanUpResources(ctx, clusterId, databaseName, tableName, user.Username,
 runner.questioner)
  }
 }()

 // Retrieve the userName and userPassword from SecretsManager
 output, err := runner.secretsmanager.SecretsManagerClient.GetSecretValue(ctx,
 &secretsmanager.GetSecretValueInput{
  SecretId: aws.String(secretId),
 })
 if err != nil {
  log.Printf("There was a problem getting the secret value: %s", err)
  log.Printf("Please make sure to create a secret named 's3express/basics/secrets'
 with keys of 'userName' and 'userPassword'.")
  panic(err)
 }
```

```
err = json.Unmarshal([]byte(*output.SecretString), &user)
if err != nil {
 log.Printf("There was a problem parsing the secret value from JSON: %s", err)
 panic(err)
}

// Create the Redshift cluster
_, err = runner.redshiftActor.CreateCluster(ctx, clusterId, user.Username,
user.Password, nodeType, clusterType, true)
if err != nil {
 var clusterAlreadyExistsFault *redshift_types.ClusterAlreadyExistsFault
 if errors.As(err, &clusterAlreadyExistsFault) {
  log.Println("Cluster already exists. Continuing.")
 } else {
  log.Println("Error creating cluster.")
  panic(err)
 }
}

// Wait for the cluster to become available
waiter := redshift.NewClusterAvailableWaiter(runner.redshiftActor.RedshiftClient)
err = waiter.Wait(ctx, &redshift.DescribeClustersInput{
 ClusterIdentifier: aws.String(clusterId),
}, 5*time.Minute)
if err != nil {
 log.Println("An error occurred waiting for the cluster.")
 panic(err)
}

// Get some info about the cluster
describeOutput, err := runner.redshiftActor.DescribeClusters(ctx, clusterId)
if err != nil {
 log.Println("Something went wrong trying to get information about the cluster.")
 panic(err)
}
log.Println("Here's some information about the cluster.")
log.Printf("The cluster's status is %s", *describeOutput.Clusters[0].ClusterStatus)
log.Printf("The cluster was created at %s",
*describeOutput.Clusters[0].ClusterCreateTime)

// List databases
log.Println("List databases in", clusterId)
runner.questioner.Ask("Press Enter to continue...")
```

```
err = runner.redshiftDataActor.ListDatabases(ctx, clusterId, databaseName,
user.Username)
if err != nil {
 log.Printf("Failed to list databases: %v\n", err)
 panic(err)
}

// Create the "Movies" table
log.Println("Now you will create a table named " + tableName + ".")
runner.questioner.Ask("Press Enter to continue...")
err = nil
result, err := runner.redshiftDataActor.CreateTable(ctx, clusterId, databaseName,
tableName, user.Username, runner.pauser, []string{"title VARCHAR(256)", "year
INT"})
if err != nil {
 log.Printf("Failed to create table: %v\n", err)
 panic(err)
}

describeInput := redshiftdata.DescribeStatementInput{
 Id: result.Id,
}
query := actions.RedshiftQuery{
 Context: ctx,
 Input:   describeInput,
 Result:  result,
}
err = runner.redshiftDataActor.WaitForQueryStatus(query, runner.pauser, true)
if err != nil {
 log.Printf("Failed to execute query: %v\n", err)
 panic(err)
}
log.Printf("Successfully executed query\n")

// Populate the "Movies" table
runner.PopulateMoviesTable(ctx, clusterId, databaseName, tableName, user.Username,
fileName)

// Query the "Movies" table by year
log.Println("Query the Movies table by year.")
year := runner.questioner.AskInt(
 fmt.Sprintf("Enter a value between %v and %v:", 2012, 2014),
 demotools.InIntRange{Lower: 2012, Upper: 2014})
```

```
 runner.QueryMoviesByYear(ctx, clusterId, databaseName, tableName, user.Username,
 year)

 // Modify the cluster's maintenance window
 runner.redshiftActor.ModifyCluster(ctx, clusterId, maintenanceWindow)

 // Delete the Redshift cluster if confirmed
 runner.cleanUpResources(ctx, clusterId, databaseName, tableName, user.Username,
 runner.questioner)

 log.Println("Thanks for watching!")
}

// cleanUpResources asks the user if they would like to delete each resource created
 during the scenario, from most
// impactful to least impactful. If any choice to delete is made, further deletion
 attempts are skipped.
func (runner *RedshiftBasicsScenario) cleanUpResources(ctx context.Context,
 clusterId string, databaseName string, tableName string, userName string,
 questioner demotools.IQuestioner) {
 deleted := false
 var err error = nil
 if questioner.AskBool("Do you want to delete the entire cluster? This will clean up
 all resources. (y/n)", "y") {
  deleted, err = runner.redshiftActor.DeleteCluster(ctx, clusterId)
  if err != nil {
   log.Printf("Error deleting cluster: %v", err)
  }
 }
 if !deleted && questioner.AskBool("Do you want to delete the dev table? This will
 clean up all inserted records but keep your cluster intact. (y/n)", "y") {
  deleted, err = runner.redshiftDataActor.DeleteTable(ctx, clusterId, databaseName,
 tableName, userName)
  if err != nil {
   log.Printf("Error deleting movies table: %v", err)
  }
 }
 if !deleted && questioner.AskBool("Do you want to delete all rows in the Movies
 table? This will clean up all inserted records but keep your cluster and table
 intact. (y/n)", "y") {
  deleted, err = runner.redshiftDataActor.DeleteDataRows(ctx, clusterId,
 databaseName, tableName, userName, runner.pauser)
  if err != nil {
   log.Printf("Error deleting data rows: %v", err)
```

```
  }
 }
 if !deleted {
  log.Print("Please manually delete any unwanted resources.")
 }
}


// loadMoviesFromJSON takes the <fileName> file and populates a slice of Movie
 objects.
func (runner *RedshiftBasicsScenario) loadMoviesFromJSON(fileName string, filesystem
 demotools.IFileSystem) ([]Movie, error) {
 file, err := filesystem.OpenFile("../../resources/sample_files/" + fileName)
 if err != nil {
  return nil, err
 }
 defer filesystem.CloseFile(file)

 var movies []Movie
 err = json.NewDecoder(file).Decode(&movies)
 if err != nil {
  return nil, err
 }

 return movies, nil
}



// PopulateMoviesTable reads data from the <fileName> file and inserts records into
 the "Movies" table.
func (runner *RedshiftBasicsScenario) PopulateMoviesTable(ctx context.Context,
 clusterId string, databaseName string, tableName string, userName string, fileName
 string) {
 log.Println("Populate the " + tableName + " table using the " + fileName + "
 file.")
 numRecords := runner.questioner.AskInt(
  fmt.Sprintf("Enter a value between %v and %v:", 10, 100),
  demotools.InIntRange{Lower: 10, Upper: 100})

 movies, err := runner.loadMoviesFromJSON(fileName, runner.filesystem)
 if err != nil {
  log.Printf("Failed to load movies from JSON: %v\n", err)
  panic(err)
```

```
}

var sqlStatements []string

for i, movie := range movies {
 if i >= numRecords {
  break
 }

 sqlStatement := fmt.Sprintf(`INSERT INTO %s (title, year) VALUES ('%s', %d);`,
  tableName,
  strings.Replace(movie.Title, "'", "''", -1), // Double any single quotes to
escape them
  movie.Year)

 sqlStatements = append(sqlStatements, sqlStatement)
}

input := &redshiftdata.BatchExecuteStatementInput{
 ClusterIdentifier: aws.String(clusterId),
 Database:          aws.String(databaseName),
 DbUser:            aws.String(userName),
 Sqls:              sqlStatements,
}

result, err := runner.redshiftDataActor.ExecuteBatchStatement(ctx, *input)
if err != nil {
 log.Printf("Failed to execute batch statement: %v\n", err)
 panic(err)
}

describeInput := redshiftdata.DescribeStatementInput{
 Id: result.Id,
}

query := actions.RedshiftQuery{
 Context: ctx,
 Result:  result,
 Input:   describeInput,
}
err = runner.redshiftDataActor.WaitForQueryStatus(query, runner.pauser, true)
if err != nil {
 log.Printf("Failed to execute batch insert query: %v\n", err)
 return
```

```
 }
 log.Printf("Successfully executed batch statement\n")

 log.Printf("%d records were added to the Movies table.\n", numRecords)
}



// QueryMoviesByYear retrieves only movies from the "Movies" table which match the
 given year.
func (runner *RedshiftBasicsScenario) QueryMoviesByYear(ctx context.Context,
 clusterId string, databaseName string, tableName string, userName string, year int)
 {

 sqlStatement := fmt.Sprintf(`SELECT title FROM %s WHERE year = %d;`, tableName,
 year)

 input := &redshiftdata.ExecuteStatementInput{
  ClusterIdentifier: aws.String(clusterId),
  Database:          aws.String(databaseName),
  DbUser:            aws.String(userName),
  Sql:               aws.String(sqlStatement),
 }

 result, err := runner.redshiftDataActor.ExecuteStatement(ctx, *input)
 if err != nil {
  log.Printf("Failed to query movies: %v\n", err)
  panic(err)
 }

 log.Println("The identifier of the statement is ", *result.Id)

 describeInput := redshiftdata.DescribeStatementInput{
  Id: result.Id,
 }

 query := actions.RedshiftQuery{
  Context: ctx,
  Input:   describeInput,
  Result:  result,
 }
 err = runner.redshiftDataActor.WaitForQueryStatus(query, runner.pauser, true)
 if err != nil {
  log.Printf("Failed to execute query: %v\n", err)
```

```
  panic(err)
 }
 log.Printf("Successfully executed query\n")

 getResultOutput, err := runner.redshiftDataActor.GetStatementResult(ctx,
 *result.Id)
 if err != nil {
  log.Printf("Failed to query movies: %v\n", err)
  panic(err)
 }
 for _, row := range getResultOutput.Records {
  for _, col := range row {
   title, ok := col.(*redshiftdata_types.FieldMemberStringValue)
   if !ok {
    log.Println("Failed to parse the field")
   } else {
    log.Printf("The Movie title field is %s\n", title.Value)
   }
  }
 }
}
```

- API-Details finden Sie in den folgenden Themen der AWS SDK für Go -API-Referenz.
  - CreateCluster
  - DescribeClusters
  - DescribeStatement
  - ExecuteStatement
  - GetStatementResult
  - ListDatabasesPaginator
  - ModifyCluster

Aktionen

`CreateCluster`

Das folgende Codebeispiel zeigt, wie man es benutzt`CreateCluster`.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import (
 "context"
 "errors"
 "log"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/redshift"
 "github.com/aws/aws-sdk-go-v2/service/redshift/types"
)



// RedshiftActions wraps Redshift service actions.
type RedshiftActions struct {
 RedshiftClient *redshift.Client
}



// CreateCluster sends a request to create a cluster with the given clusterId using
 the provided credentials.
func (actor RedshiftActions) CreateCluster(ctx context.Context, clusterId string,
 userName string, userPassword string, nodeType string, clusterType string,
 publiclyAccessible bool) (*redshift.CreateClusterOutput, error) {
 // Create a new Redshift cluster
 input := &redshift.CreateClusterInput{
  ClusterIdentifier:  aws.String(clusterId),
  MasterUserPassword: aws.String(userPassword),
  MasterUsername:     aws.String(userName),
  NodeType:           aws.String(nodeType),
  ClusterType:        aws.String(clusterType),
  PubliclyAccessible: aws.Bool(publiclyAccessible),
```

```
 }
 var opErr *types.ClusterAlreadyExistsFault
 output, err := actor.RedshiftClient.CreateCluster(ctx, input)
 if err != nil && errors.As(err, &opErr) {
  log.Println("Cluster already exists")
  return nil, nil
 } else if err != nil {
  log.Printf("Failed to create Redshift cluster: %v\n", err)
  return nil, err
 }

 log.Printf("Created cluster %s\n", *output.Cluster.ClusterIdentifier)
 return output, nil
}
```

- Einzelheiten zur API finden Sie CreateClusterin der AWS SDK für Go API-Referenz.

`DeleteCluster`

Das folgende Codebeispiel zeigt die Verwendung`DeleteCluster`.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import (
 "context"
 "errors"
 "log"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/redshift"
 "github.com/aws/aws-sdk-go-v2/service/redshift/types"
```

```
)



// RedshiftActions wraps Redshift service actions.
type RedshiftActions struct {
 RedshiftClient *redshift.Client
}



// DeleteCluster deletes the given cluster.
func (actor RedshiftActions) DeleteCluster(ctx context.Context, clusterId string)
 (bool, error) {
 input := redshift.DeleteClusterInput{
  ClusterIdentifier:        aws.String(clusterId),
  SkipFinalClusterSnapshot: aws.Bool(true),
 }
 _, err := actor.RedshiftClient.DeleteCluster(ctx, &input)
 var opErr *types.ClusterNotFoundFault
 if err != nil && errors.As(err, &opErr) {
  log.Println("Cluster was not found. Where could it be?")
  return false, err
 } else if err != nil {
  log.Printf("Failed to delete Redshift cluster: %v\n", err)
  return false, err
 }
 waiter := redshift.NewClusterDeletedWaiter(actor.RedshiftClient)
 err = waiter.Wait(ctx, &redshift.DescribeClustersInput{
  ClusterIdentifier: aws.String(clusterId),
 }, 5*time.Minute)
 if err != nil {
  log.Printf("Wait time exceeded for deleting cluster, continuing: %v\n", err)
 }
 log.Printf("The cluster %s was deleted\n", clusterId)
 return true, nil
}
```

- Einzelheiten zur API finden Sie DeleteClusterin der AWS SDK für Go API-Referenz.

`DescribeClusters`

Das folgende Codebeispiel zeigt die Verwendung`DescribeClusters`.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import (
 "context"
 "errors"
 "log"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/redshift"
 "github.com/aws/aws-sdk-go-v2/service/redshift/types"
)



// RedshiftActions wraps Redshift service actions.
type RedshiftActions struct {
 RedshiftClient *redshift.Client
}



// DescribeClusters returns information about the given cluster.
func (actor RedshiftActions) DescribeClusters(ctx context.Context, clusterId string)
 (*redshift.DescribeClustersOutput, error) {
 input, err := actor.RedshiftClient.DescribeClusters(ctx,
 &redshift.DescribeClustersInput{
  ClusterIdentifier: aws.String(clusterId),
 })
 var opErr *types.AccessToClusterDeniedFault
 if errors.As(err, &opErr) {
  println("Access to cluster denied.")
```

```
  panic(err)
 } else if err != nil {
  println("Failed to describe Redshift clusters.")
  return nil, err
 }
 return input, nil
}
```

- Einzelheiten zur API finden Sie DescribeClustersin der AWS SDK für Go API-Referenz.

`ModifyCluster`

Das folgende Codebeispiel zeigt die Verwendung`ModifyCluster`.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import (
 "context"
 "errors"
 "log"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/redshift"
 "github.com/aws/aws-sdk-go-v2/service/redshift/types"
)



// RedshiftActions wraps Redshift service actions.
type RedshiftActions struct {
 RedshiftClient *redshift.Client
}
```

```
// ModifyCluster sets the preferred maintenance window for the given cluster.
func (actor RedshiftActions) ModifyCluster(ctx context.Context, clusterId string,
 maintenanceWindow string) *redshift.ModifyClusterOutput {
 // Modify the cluster's maintenance window
 input := &redshift.ModifyClusterInput{
  ClusterIdentifier:          aws.String(clusterId),
  PreferredMaintenanceWindow: aws.String(maintenanceWindow),
 }

 var opErr *types.InvalidClusterStateFault
 output, err := actor.RedshiftClient.ModifyCluster(ctx, input)
 if err != nil && errors.As(err, &opErr) {
  log.Println("Cluster is in an invalid state.")
  panic(err)
 } else if err != nil {
  log.Printf("Failed to modify Redshift cluster: %v\n", err)
  panic(err)
 }

 log.Printf("The cluster was successfully modified and now has %s as the maintenance
 window\n", *output.Cluster.PreferredMaintenanceWindow)
 return output
}
```

- Einzelheiten zur API finden Sie ModifyClusterin der AWS SDK für Go API-Referenz.

Amazon S3 S3-Beispiele mit SDK for Go V2

Die folgenden Codebeispiele zeigen Ihnen, wie Sie mithilfe der AWS SDK für Go V2 mit Amazon S3 Aktionen ausführen und allgemeine Szenarien implementieren.

Bei Grundlagen handelt es sich um Code-Beispiele, die Ihnen zeigen, wie Sie die wesentlichen Vorgänge innerhalb eines Services ausführen.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Service-Funktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarios anzeigen.

Szenarien sind Code-Beispiele, die Ihnen zeigen, wie Sie bestimmte Aufgaben ausführen, indem Sie mehrere Funktionen innerhalb eines Services aufrufen oder mit anderen AWS-Services kombinieren.

Jedes Beispiel enthält einen Link zum vollständigen Quellcode, in dem Sie Anweisungen zur Einrichtung und Ausführung des Codes im Kontext finden.

Erste Schritte

Hello Amazon S3

Die folgenden Codebeispiele veranschaulichen die ersten Schritte mit Amazon S3.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
package main

import (
 "context"
 "errors"
 "fmt"

 "github.com/aws/aws-sdk-go-v2/config"
 "github.com/aws/aws-sdk-go-v2/service/s3"
 "github.com/aws/smithy-go"
)

// main uses the AWS SDK for Go V2 to create an Amazon Simple Storage Service
// (Amazon S3) client and list up to 10 buckets in your account.
// This example uses the default settings specified in your shared credentials
// and config files.
func main() {
 ctx := context.Background()
 sdkConfig, err := config.LoadDefaultConfig(ctx)
 if err != nil {
  fmt.Println("Couldn't load default configuration. Have you set up your AWS
 account?")
```

```
  fmt.Println(err)
  return
 }
 s3Client := s3.NewFromConfig(sdkConfig)
 count := 10
 fmt.Printf("Let's list up to %v buckets for your account.\n", count)
 result, err := s3Client.ListBuckets(ctx, &s3.ListBucketsInput{})
 if err != nil {
  var ae smithy.APIError
  if errors.As(err, &ae) && ae.ErrorCode() == "AccessDenied" {
   fmt.Println("You don't have permission to list buckets for this account.")
  } else {
   fmt.Printf("Couldn't list buckets for your account. Here's why: %v\n", err)
  }
  return
 }
 if len(result.Buckets) == 0 {
  fmt.Println("You don't have any buckets!")
 } else {
  if count > len(result.Buckets) {
   count = len(result.Buckets)
  }
  for _, bucket := range result.Buckets[:count] {
   fmt.Printf("\t%v\n", *bucket.Name)
  }
 }
}
```

- Einzelheiten zur API finden Sie ListBucketsin der AWS SDK für Go API-Referenz.

Themen

- Grundlagen
- Aktionen
- Szenarien
- Serverless-Beispiele

Grundlagen

Erlernen der Grundlagen

Wie das aussehen kann, sehen Sie am nachfolgenden Beispielcode:

- Erstellen Sie einen Bucket und laden Sie eine Datei in ihn hoch.
- Laden Sie ein Objekt aus einem Bucket herunter.
- Kopieren Sie ein Objekt in einen Unterordner eines Buckets.
- Listen Sie die Objekte in einem Bucket auf.
- Löschen Sie die Bucket-Objekte und den Bucket.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Definieren Sie eine Struktur, die die vom Szenario verwendete Bucket- und Objektaktionen umschließt.

```
import (
 "bytes"
 "context"
 "errors"
 "fmt"
 "io"
 "log"
 "os"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/s3/manager"
 "github.com/aws/aws-sdk-go-v2/service/s3"
 "github.com/aws/aws-sdk-go-v2/service/s3/types"
 "github.com/aws/smithy-go"
)
```

```
// BucketBasics encapsulates the Amazon Simple Storage Service (Amazon S3) actions
// used in the examples.
// It contains S3Client, an Amazon S3 service client that is used to perform bucket
// and object actions.
type BucketBasics struct {
 S3Client *s3.Client
}



// ListBuckets lists the buckets in the current account.
func (basics BucketBasics) ListBuckets(ctx context.Context) ([]types.Bucket, error)
 {
 var err error
 var output *s3.ListBucketsOutput
 var buckets []types.Bucket
 bucketPaginator := s3.NewListBucketsPaginator(basics.S3Client,
 &s3.ListBucketsInput{})
 for bucketPaginator.HasMorePages() {
  output, err = bucketPaginator.NextPage(ctx)
  if err != nil {
   var apiErr smithy.APIError
   if errors.As(err, &apiErr) && apiErr.ErrorCode() == "AccessDenied" {
    fmt.Println("You don't have permission to list buckets for this account.")
    err = apiErr
   } else {
    log.Printf("Couldn't list buckets for your account. Here's why: %v\n", err)
   }
   break
  } else {
   buckets = append(buckets, output.Buckets...)
  }
 }
 return buckets, err
}



// BucketExists checks whether a bucket exists in the current account.
func (basics BucketBasics) BucketExists(ctx context.Context, bucketName string)
 (bool, error) {
 _, err := basics.S3Client.HeadBucket(ctx, &s3.HeadBucketInput{
  Bucket: aws.String(bucketName),
```

```
 })
 exists := true
 if err != nil {
  var apiError smithy.APIError
  if errors.As(err, &apiError) {
   switch apiError.(type) {
   case *types.NotFound:
    log.Printf("Bucket %v is available.\n", bucketName)
    exists = false
    err = nil
   default:
    log.Printf("Either you don't have access to bucket %v or another error occurred.
 "+
     "Here's what happened: %v\n", bucketName, err)
   }
  }
 } else {
  log.Printf("Bucket %v exists and you already own it.", bucketName)
 }

 return exists, err
}



// CreateBucket creates a bucket with the specified name in the specified Region.
func (basics BucketBasics) CreateBucket(ctx context.Context, name string, region
 string) error {
 _, err := basics.S3Client.CreateBucket(ctx, &s3.CreateBucketInput{
  Bucket: aws.String(name),
  CreateBucketConfiguration: &types.CreateBucketConfiguration{
   LocationConstraint: types.BucketLocationConstraint(region),
  },
 })
 if err != nil {
  var owned *types.BucketAlreadyOwnedByYou
  var exists *types.BucketAlreadyExists
  if errors.As(err, &owned) {
   log.Printf("You already own bucket %s.\n", name)
   err = owned
  } else if errors.As(err, &exists) {
   log.Printf("Bucket %s already exists.\n", name)
   err = exists
  }
```

```
 } else {
  err = s3.NewBucketExistsWaiter(basics.S3Client).Wait(
   ctx, &s3.HeadBucketInput{Bucket: aws.String(name)}, time.Minute)
  if err != nil {
   log.Printf("Failed attempt to wait for bucket %s to exist.\n", name)
  }
 }
 return err
}



// UploadFile reads from a file and puts the data into an object in a bucket.
func (basics BucketBasics) UploadFile(ctx context.Context, bucketName string,
 objectKey string, fileName string) error {
 file, err := os.Open(fileName)
 if err != nil {
  log.Printf("Couldn't open file %v to upload. Here's why: %v\n", fileName, err)
 } else {
  defer file.Close()
  _, err = basics.S3Client.PutObject(ctx, &s3.PutObjectInput{
   Bucket: aws.String(bucketName),
   Key:    aws.String(objectKey),
   Body:   file,
  })
  if err != nil {
   var apiErr smithy.APIError
   if errors.As(err, &apiErr) && apiErr.ErrorCode() == "EntityTooLarge" {
    log.Printf("Error while uploading object to %s. The object is too large.\n"+
     "To upload objects larger than 5GB, use the S3 console (160GB max)\n"+
     "or the multipart upload API (5TB max).", bucketName)
   } else {
    log.Printf("Couldn't upload file %v to %v:%v. Here's why: %v\n",
     fileName, bucketName, objectKey, err)
   }
  } else {
   err = s3.NewObjectExistsWaiter(basics.S3Client).Wait(
    ctx, &s3.HeadObjectInput{Bucket: aws.String(bucketName), Key:
 aws.String(objectKey)}, time.Minute)
   if err != nil {
    log.Printf("Failed attempt to wait for object %s to exist.\n", objectKey)
   }
  }
 }
```

```
 return err
}



// UploadLargeObject uses an upload manager to upload data to an object in a bucket.
// The upload manager breaks large data into parts and uploads the parts
 concurrently.
func (basics BucketBasics) UploadLargeObject(ctx context.Context, bucketName string,
 objectKey string, largeObject []byte) error {
 largeBuffer := bytes.NewReader(largeObject)
 var partMiBs int64 = 10
 uploader := manager.NewUploader(basics.S3Client, func(u *manager.Uploader) {
  u.PartSize = partMiBs * 1024 * 1024
 })
 _, err := uploader.Upload(ctx, &s3.PutObjectInput{
  Bucket: aws.String(bucketName),
  Key:    aws.String(objectKey),
  Body:   largeBuffer,
 })
 if err != nil {
  var apiErr smithy.APIError
  if errors.As(err, &apiErr) && apiErr.ErrorCode() == "EntityTooLarge" {
   log.Printf("Error while uploading object to %s. The object is too large.\n"+
    "The maximum size for a multipart upload is 5TB.", bucketName)
  } else {
   log.Printf("Couldn't upload large object to %v:%v. Here's why: %v\n",
    bucketName, objectKey, err)
  }
 } else {
  err = s3.NewObjectExistsWaiter(basics.S3Client).Wait(
   ctx, &s3.HeadObjectInput{Bucket: aws.String(bucketName), Key:
 aws.String(objectKey)}, time.Minute)
  if err != nil {
   log.Printf("Failed attempt to wait for object %s to exist.\n", objectKey)
  }
 }

 return err
}



// DownloadFile gets an object from a bucket and stores it in a local file.
```

```
func (basics BucketBasics) DownloadFile(ctx context.Context, bucketName string,
 objectKey string, fileName string) error {
 result, err := basics.S3Client.GetObject(ctx, &s3.GetObjectInput{
  Bucket: aws.String(bucketName),
  Key:    aws.String(objectKey),
 })
 if err != nil {
  var noKey *types.NoSuchKey
  if errors.As(err, &noKey) {
   log.Printf("Can't get object %s from bucket %s. No such key exists.\n",
 objectKey, bucketName)
   err = noKey
  } else {
   log.Printf("Couldn't get object %v:%v. Here's why: %v\n", bucketName, objectKey,
 err)
  }
  return err
 }
 defer result.Body.Close()
 file, err := os.Create(fileName)
 if err != nil {
  log.Printf("Couldn't create file %v. Here's why: %v\n", fileName, err)
  return err
 }
 defer file.Close()
 body, err := io.ReadAll(result.Body)
 if err != nil {
  log.Printf("Couldn't read object body from %v. Here's why: %v\n", objectKey, err)
 }
 _, err = file.Write(body)
 return err
}



// DownloadLargeObject uses a download manager to download an object from a bucket.
// The download manager gets the data in parts and writes them to a buffer until all
 of
// the data has been downloaded.
func (basics BucketBasics) DownloadLargeObject(ctx context.Context, bucketName
 string, objectKey string) ([]byte, error) {
 var partMiBs int64 = 10
 downloader := manager.NewDownloader(basics.S3Client, func(d *manager.Downloader) {
  d.PartSize = partMiBs * 1024 * 1024
```

```
 })
 buffer := manager.NewWriteAtBuffer([]byte{})
 _, err := downloader.Download(ctx, buffer, &s3.GetObjectInput{
  Bucket: aws.String(bucketName),
  Key:    aws.String(objectKey),
 })
 if err != nil {
  log.Printf("Couldn't download large object from %v:%v. Here's why: %v\n",
   bucketName, objectKey, err)
 }
 return buffer.Bytes(), err
}



// CopyToFolder copies an object in a bucket to a subfolder in the same bucket.
func (basics BucketBasics) CopyToFolder(ctx context.Context, bucketName string,
 objectKey string, folderName string) error {
 objectDest := fmt.Sprintf("%v/%v", folderName, objectKey)
 _, err := basics.S3Client.CopyObject(ctx, &s3.CopyObjectInput{
  Bucket:     aws.String(bucketName),
  CopySource: aws.String(fmt.Sprintf("%v/%v", bucketName, objectKey)),
  Key:        aws.String(objectDest),
 })
 if err != nil {
  var notActive *types.ObjectNotInActiveTierError
  if errors.As(err, &notActive) {
   log.Printf("Couldn't copy object %s from %s because the object isn't in the
 active tier.\n",
    objectKey, bucketName)
   err = notActive
  }
 } else {
  err = s3.NewObjectExistsWaiter(basics.S3Client).Wait(
   ctx, &s3.HeadObjectInput{Bucket: aws.String(bucketName), Key:
 aws.String(objectDest)}, time.Minute)
  if err != nil {
   log.Printf("Failed attempt to wait for object %s to exist.\n", objectDest)
  }
 }
 return err
}
```

```
// CopyToBucket copies an object in a bucket to another bucket.
func (basics BucketBasics) CopyToBucket(ctx context.Context, sourceBucket string,
 destinationBucket string, objectKey string) error {
 _, err := basics.S3Client.CopyObject(ctx, &s3.CopyObjectInput{
  Bucket:     aws.String(destinationBucket),
  CopySource: aws.String(fmt.Sprintf("%v/%v", sourceBucket, objectKey)),
  Key:        aws.String(objectKey),
 })
 if err != nil {
  var notActive *types.ObjectNotInActiveTierError
  if errors.As(err, &notActive) {
   log.Printf("Couldn't copy object %s from %s because the object isn't in the
 active tier.\n",
    objectKey, sourceBucket)
   err = notActive
  }
 } else {
  err = s3.NewObjectExistsWaiter(basics.S3Client).Wait(
   ctx, &s3.HeadObjectInput{Bucket: aws.String(destinationBucket), Key:
 aws.String(objectKey)}, time.Minute)
  if err != nil {
   log.Printf("Failed attempt to wait for object %s to exist.\n", objectKey)
  }
 }
 return err
}



// ListObjects lists the objects in a bucket.
func (basics BucketBasics) ListObjects(ctx context.Context, bucketName string)
 ([]types.Object, error) {
 var err error
 var output *s3.ListObjectsV2Output
 input := &s3.ListObjectsV2Input{
  Bucket: aws.String(bucketName),
 }
 var objects []types.Object
 objectPaginator := s3.NewListObjectsV2Paginator(basics.S3Client, input)
 for objectPaginator.HasMorePages() {
  output, err = objectPaginator.NextPage(ctx)
  if err != nil {
   var noBucket *types.NoSuchBucket
```

```
   if errors.As(err, &noBucket) {
    log.Printf("Bucket %s does not exist.\n", bucketName)
    err = noBucket
   }
   break
  } else {
   objects = append(objects, output.Contents...)
  }
 }
 return objects, err
}



// DeleteObjects deletes a list of objects from a bucket.
func (basics BucketBasics) DeleteObjects(ctx context.Context, bucketName string,
 objectKeys []string) error {
 var objectIds []types.ObjectIdentifier
 for _, key := range objectKeys {
  objectIds = append(objectIds, types.ObjectIdentifier{Key: aws.String(key)})
 }
 output, err := basics.S3Client.DeleteObjects(ctx, &s3.DeleteObjectsInput{
  Bucket: aws.String(bucketName),
  Delete: &types.Delete{Objects: objectIds, Quiet: aws.Bool(true)},
 })
 if err != nil || len(output.Errors) > 0 {
  log.Printf("Error deleting objects from bucket %s.\n", bucketName)
  if err != nil {
   var noBucket *types.NoSuchBucket
   if errors.As(err, &noBucket) {
    log.Printf("Bucket %s does not exist.\n", bucketName)
    err = noBucket
   }
  } else if len(output.Errors) > 0 {
   for _, outErr := range output.Errors {
    log.Printf("%s: %s\n", *outErr.Key, *outErr.Message)
   }
   err = fmt.Errorf("%s", *output.Errors[0].Message)
  }
 } else {
  for _, delObjs := range output.Deleted {
   err = s3.NewObjectNotExistsWaiter(basics.S3Client).Wait(
    ctx, &s3.HeadObjectInput{Bucket: aws.String(bucketName), Key: delObjs.Key},
 time.Minute)
```

```
   if err != nil {
    log.Printf("Failed attempt to wait for object %s to be deleted.\n",
 *delObjs.Key)
   } else {
    log.Printf("Deleted %s.\n", *delObjs.Key)
   }
  }
 }
 return err
}



// DeleteBucket deletes a bucket. The bucket must be empty or an error is returned.
func (basics BucketBasics) DeleteBucket(ctx context.Context, bucketName string)
 error {
 _, err := basics.S3Client.DeleteBucket(ctx, &s3.DeleteBucketInput{
  Bucket: aws.String(bucketName)})
 if err != nil {
  var noBucket *types.NoSuchBucket
  if errors.As(err, &noBucket) {
   log.Printf("Bucket %s does not exist.\n", bucketName)
   err = noBucket
  } else {
   log.Printf("Couldn't delete bucket %v. Here's why: %v\n", bucketName, err)
  }
 } else {
  err = s3.NewBucketNotExistsWaiter(basics.S3Client).Wait(
   ctx, &s3.HeadBucketInput{Bucket: aws.String(bucketName)}, time.Minute)
  if err != nil {
   log.Printf("Failed attempt to wait for bucket %s to be deleted.\n", bucketName)
  } else {
   log.Printf("Deleted %s.\n", bucketName)
  }
 }
 return err
}
```

Führen Sie ein interaktives Szenario aus, das Ihnen zeigt, wie Sie mit S3-Buckets und Objekten arbeiten.

```
import (
 "context"
 "fmt"
 "log"
 "os"
 "strings"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/s3"
 "github.com/awsdocs/aws-doc-sdk-examples/gov2/demotools"
 "github.com/awsdocs/aws-doc-sdk-examples/gov2/s3/actions"
)

// RunGetStartedScenario is an interactive example that shows you how to use Amazon
// Simple Storage Service (Amazon S3) to create an S3 bucket and use it to store
 objects.
//
// 1. Create a bucket.
// 2. Upload a local file to the bucket.
// 3. Download an object to a local file.
// 4. Copy an object to a different folder in the bucket.
// 5. List objects in the bucket.
// 6. Delete all objects in the bucket.
// 7. Delete the bucket.
//
// This example creates an Amazon S3 service client from the specified sdkConfig so
 that
// you can replace it with a mocked or stubbed config for unit testing.
//
// It uses a questioner from the `demotools` package to get input during the
 example.
// This package can be found in the ..\..\demotools folder of this repo.
func RunGetStartedScenario(ctx context.Context, sdkConfig aws.Config, questioner
 demotools.IQuestioner) {
 defer func() {
  if r := recover(); r != nil {
   log.Println("Something went wrong with the demo.")
   _, isMock := questioner.(*demotools.MockQuestioner)
   if isMock || questioner.AskBool("Do you want to see the full error message (y/
n)?", "y") {
    log.Println(r)
   }
```

```
 }
}()

log.Println(strings.Repeat("-", 88))
log.Println("Welcome to the Amazon S3 getting started demo.")
log.Println(strings.Repeat("-", 88))

s3Client := s3.NewFromConfig(sdkConfig)
bucketBasics := actions.BucketBasics{S3Client: s3Client}

count := 10
log.Printf("Let's list up to %v buckets for your account:", count)
buckets, err := bucketBasics.ListBuckets(ctx)
if err != nil {
 panic(err)
}
if len(buckets) == 0 {
 log.Println("You don't have any buckets!")
} else {
 if count > len(buckets) {
  count = len(buckets)
 }
 for _, bucket := range buckets[:count] {
  log.Printf("\t%v\n", *bucket.Name)
 }
}

bucketName := questioner.Ask("Let's create a bucket. Enter a name for your
bucket:",
 demotools.NotEmpty{})
bucketExists, err := bucketBasics.BucketExists(ctx, bucketName)
if err != nil {
 panic(err)
}
if !bucketExists {
 err = bucketBasics.CreateBucket(ctx, bucketName, sdkConfig.Region)
 if err != nil {
  panic(err)
 } else {
  log.Println("Bucket created.")
 }
}
log.Println(strings.Repeat("-", 88))
```

```
fmt.Println("Let's upload a file to your bucket.")
smallFile := questioner.Ask("Enter the path to a file you want to upload:",
 demotools.NotEmpty{})
const smallKey = "doc-example-key"
err = bucketBasics.UploadFile(ctx, bucketName, smallKey, smallFile)
if err != nil {
 panic(err)
}
log.Printf("Uploaded %v as %v.\n", smallFile, smallKey)
log.Println(strings.Repeat("-", 88))

log.Printf("Let's download %v to a file.", smallKey)
downloadFileName := questioner.Ask("Enter a name for the downloaded file:",
demotools.NotEmpty{})
err = bucketBasics.DownloadFile(ctx, bucketName, smallKey, downloadFileName)
if err != nil {
 panic(err)
}
log.Printf("File %v downloaded.", downloadFileName)
log.Println(strings.Repeat("-", 88))

log.Printf("Let's copy %v to a folder in the same bucket.", smallKey)
folderName := questioner.Ask("Enter a folder name: ", demotools.NotEmpty{})
err = bucketBasics.CopyToFolder(ctx, bucketName, smallKey, folderName)
if err != nil {
 panic(err)
}
log.Printf("Copied %v to %v/%v.\n", smallKey, folderName, smallKey)
log.Println(strings.Repeat("-", 88))

log.Println("Let's list the objects in your bucket.")
questioner.Ask("Press Enter when you're ready.")
objects, err := bucketBasics.ListObjects(ctx, bucketName)
if err != nil {
 panic(err)
}
log.Printf("Found %v objects.\n", len(objects))
var objKeys []string
for _, object := range objects {
 objKeys = append(objKeys, *object.Key)
 log.Printf("\t%v\n", *object.Key)
}
log.Println(strings.Repeat("-", 88))
```

```
 if questioner.AskBool("Do you want to delete your bucket and all of its "+
  "contents? (y/n)", "y") {
  log.Println("Deleting objects.")
  err = bucketBasics.DeleteObjects(ctx, bucketName, objKeys)
  if err != nil {
   panic(err)
  }
  log.Println("Deleting bucket.")
  err = bucketBasics.DeleteBucket(ctx, bucketName)
  if err != nil {
   panic(err)
  }
  log.Printf("Deleting downloaded file %v.\n", downloadFileName)
  err = os.Remove(downloadFileName)
  if err != nil {
   panic(err)
  }
 } else {
  log.Println("Okay. Don't forget to delete objects from your bucket to avoid
 charges.")
 }
 log.Println(strings.Repeat("-", 88))

 log.Println("Thanks for watching!")
 log.Println(strings.Repeat("-", 88))
}
```

- API-Details finden Sie in den folgenden Themen der AWS SDK für Go -API-Referenz.
  - CopyObject
  - CreateBucket
  - DeleteBucket
  - DeleteObjects
  - GetObject
  - ListObjectsV2
  - PutObject

Aktionen

`CopyObject`

Das folgende Codebeispiel zeigt die Verwendung`CopyObject`.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import (
 "bytes"
 "context"
 "errors"
 "fmt"
 "io"
 "log"
 "os"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/s3/manager"
 "github.com/aws/aws-sdk-go-v2/service/s3"
 "github.com/aws/aws-sdk-go-v2/service/s3/types"
 "github.com/aws/smithy-go"
)

// BucketBasics encapsulates the Amazon Simple Storage Service (Amazon S3) actions
// used in the examples.
// It contains S3Client, an Amazon S3 service client that is used to perform bucket
// and object actions.
type BucketBasics struct {
 S3Client *s3.Client
}
```

```
// CopyToBucket copies an object in a bucket to another bucket.
func (basics BucketBasics) CopyToBucket(ctx context.Context, sourceBucket string,
 destinationBucket string, objectKey string) error {
 _, err := basics.S3Client.CopyObject(ctx, &s3.CopyObjectInput{
  Bucket:     aws.String(destinationBucket),
  CopySource: aws.String(fmt.Sprintf("%v/%v", sourceBucket, objectKey)),
  Key:        aws.String(objectKey),
 })
 if err != nil {
  var notActive *types.ObjectNotInActiveTierError
  if errors.As(err, &notActive) {
   log.Printf("Couldn't copy object %s from %s because the object isn't in the
 active tier.\n",
    objectKey, sourceBucket)
   err = notActive
  }
 } else {
  err = s3.NewObjectExistsWaiter(basics.S3Client).Wait(
   ctx, &s3.HeadObjectInput{Bucket: aws.String(destinationBucket), Key:
 aws.String(objectKey)}, time.Minute)
  if err != nil {
   log.Printf("Failed attempt to wait for object %s to exist.\n", objectKey)
  }
 }
 return err
}
```

- Einzelheiten zur API finden Sie CopyObjectin der AWS SDK für Go API-Referenz.

`CreateBucket`

Das folgende Codebeispiel zeigt die Verwendung`CreateBucket`.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Erstellen Sie einen Bucket mit Standardkonfiguration.

```
import (
 "bytes"
 "context"
 "errors"
 "fmt"
 "io"
 "log"
 "os"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/s3/manager"
 "github.com/aws/aws-sdk-go-v2/service/s3"
 "github.com/aws/aws-sdk-go-v2/service/s3/types"
 "github.com/aws/smithy-go"
)

// BucketBasics encapsulates the Amazon Simple Storage Service (Amazon S3) actions
// used in the examples.
// It contains S3Client, an Amazon S3 service client that is used to perform bucket
// and object actions.
type BucketBasics struct {
 S3Client *s3.Client
}



// CreateBucket creates a bucket with the specified name in the specified Region.
func (basics BucketBasics) CreateBucket(ctx context.Context, name string, region
 string) error {
 _, err := basics.S3Client.CreateBucket(ctx, &s3.CreateBucketInput{
  Bucket: aws.String(name),
  CreateBucketConfiguration: &types.CreateBucketConfiguration{
   LocationConstraint: types.BucketLocationConstraint(region),
  },
 })
 if err != nil {
  var owned *types.BucketAlreadyOwnedByYou
  var exists *types.BucketAlreadyExists
  if errors.As(err, &owned) {
   log.Printf("You already own bucket %s.\n", name)
```

```
   err = owned
  } else if errors.As(err, &exists) {
   log.Printf("Bucket %s already exists.\n", name)
   err = exists
  }
 } else {
  err = s3.NewBucketExistsWaiter(basics.S3Client).Wait(
   ctx, &s3.HeadBucketInput{Bucket: aws.String(name)}, time.Minute)
  if err != nil {
   log.Printf("Failed attempt to wait for bucket %s to exist.\n", name)
  }
 }
 return err
}
```

Erstellen Sie einen Bucket mit Objektsperre und warten Sie, bis er existiert.

```
import (
 "bytes"
 "context"
 "errors"
 "fmt"
 "log"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/s3/manager"
 "github.com/aws/aws-sdk-go-v2/service/s3"
 "github.com/aws/aws-sdk-go-v2/service/s3/types"
 "github.com/aws/smithy-go"
)

// S3Actions wraps S3 service actions.
type S3Actions struct {
 S3Client  *s3.Client
 S3Manager *manager.Uploader
}
```

```
// CreateBucketWithLock creates a new S3 bucket with optional object locking enabled
// and waits for the bucket to exist before returning.
func (actor S3Actions) CreateBucketWithLock(ctx context.Context, bucket string,
 region string, enableObjectLock bool) (string, error) {
 input := &s3.CreateBucketInput{
  Bucket: aws.String(bucket),
  CreateBucketConfiguration: &types.CreateBucketConfiguration{
   LocationConstraint: types.BucketLocationConstraint(region),
  },
 }

 if enableObjectLock {
  input.ObjectLockEnabledForBucket = aws.Bool(true)
 }

 _, err := actor.S3Client.CreateBucket(ctx, input)
 if err != nil {
  var owned *types.BucketAlreadyOwnedByYou
  var exists *types.BucketAlreadyExists
  if errors.As(err, &owned) {
   log.Printf("You already own bucket %s.\n", bucket)
   err = owned
  } else if errors.As(err, &exists) {
   log.Printf("Bucket %s already exists.\n", bucket)
   err = exists
  }
 } else {
  err = s3.NewBucketExistsWaiter(actor.S3Client).Wait(
   ctx, &s3.HeadBucketInput{Bucket: aws.String(bucket)}, time.Minute)
  if err != nil {
   log.Printf("Failed attempt to wait for bucket %s to exist.\n", bucket)
  }
 }

 return bucket, err
}
```

- Einzelheiten zur API finden Sie CreateBucketin der AWS SDK für Go API-Referenz.

`DeleteBucket`

Das folgende Codebeispiel zeigt die Verwendung`DeleteBucket`.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import (
 "bytes"
 "context"
 "errors"
 "fmt"
 "io"
 "log"
 "os"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/s3/manager"
 "github.com/aws/aws-sdk-go-v2/service/s3"
 "github.com/aws/aws-sdk-go-v2/service/s3/types"
 "github.com/aws/smithy-go"
)

// BucketBasics encapsulates the Amazon Simple Storage Service (Amazon S3) actions
// used in the examples.
// It contains S3Client, an Amazon S3 service client that is used to perform bucket
// and object actions.
type BucketBasics struct {
 S3Client *s3.Client
}



// DeleteBucket deletes a bucket. The bucket must be empty or an error is returned.
func (basics BucketBasics) DeleteBucket(ctx context.Context, bucketName string)
 error {
```

```
 _, err := basics.S3Client.DeleteBucket(ctx, &s3.DeleteBucketInput{
  Bucket: aws.String(bucketName)})
 if err != nil {
  var noBucket *types.NoSuchBucket
  if errors.As(err, &noBucket) {
   log.Printf("Bucket %s does not exist.\n", bucketName)
   err = noBucket
  } else {
   log.Printf("Couldn't delete bucket %v. Here's why: %v\n", bucketName, err)
  }
 } else {
  err = s3.NewBucketNotExistsWaiter(basics.S3Client).Wait(
   ctx, &s3.HeadBucketInput{Bucket: aws.String(bucketName)}, time.Minute)
  if err != nil {
   log.Printf("Failed attempt to wait for bucket %s to be deleted.\n", bucketName)
  } else {
   log.Printf("Deleted %s.\n", bucketName)
  }
 }
 return err
}
```

- Einzelheiten zur API finden Sie DeleteBucketin der AWS SDK für Go API-Referenz.

`DeleteObject`

Das folgende Codebeispiel zeigt die Verwendung`DeleteObject`.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import (
 "bytes"
 "context"
```

```
 "errors"
 "fmt"
 "log"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/s3/manager"
 "github.com/aws/aws-sdk-go-v2/service/s3"
 "github.com/aws/aws-sdk-go-v2/service/s3/types"
 "github.com/aws/smithy-go"
)

// S3Actions wraps S3 service actions.
type S3Actions struct {
 S3Client  *s3.Client
 S3Manager *manager.Uploader
}



// DeleteObject deletes an object from a bucket.
func (actor S3Actions) DeleteObject(ctx context.Context, bucket string, key string,
 versionId string, bypassGovernance bool) (bool, error) {
 deleted := false
 input := &s3.DeleteObjectInput{
  Bucket: aws.String(bucket),
  Key:    aws.String(key),
 }
 if versionId != "" {
  input.VersionId = aws.String(versionId)
 }
 if bypassGovernance {
  input.BypassGovernanceRetention = aws.Bool(true)
 }
 _, err := actor.S3Client.DeleteObject(ctx, input)
 if err != nil {
  var noKey *types.NoSuchKey
  var apiErr *smithy.GenericAPIError
  if errors.As(err, &noKey) {
   log.Printf("Object %s does not exist in %s.\n", key, bucket)
   err = noKey
  } else if errors.As(err, &apiErr) {
   switch apiErr.ErrorCode() {
   case "AccessDenied":
```

```
    log.Printf("Access denied: cannot delete object %s from %s.\n", key, bucket)
    err = nil
   case "InvalidArgument":
    if bypassGovernance {
     log.Printf("You cannot specify bypass governance on a bucket without lock
 enabled.")
     err = nil
    }
   }
  }
 } else {
  err = s3.NewObjectNotExistsWaiter(actor.S3Client).Wait(
   ctx, &s3.HeadObjectInput{Bucket: aws.String(bucket), Key: aws.String(key)},
 time.Minute)
  if err != nil {
   log.Printf("Failed attempt to wait for object %s in bucket %s to be deleted.\n",
 key, bucket)
  } else {
   deleted = true
  }
 }
 return deleted, err
}
```

- Einzelheiten zur API finden Sie DeleteObjectin der AWS SDK für Go API-Referenz.

`DeleteObjects`

Das folgende Codebeispiel zeigt die Verwendung`DeleteObjects`.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import (
```

```
 "bytes"
 "context"
 "errors"
 "fmt"
 "log"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/s3/manager"
 "github.com/aws/aws-sdk-go-v2/service/s3"
 "github.com/aws/aws-sdk-go-v2/service/s3/types"
 "github.com/aws/smithy-go"
)

// S3Actions wraps S3 service actions.
type S3Actions struct {
 S3Client  *s3.Client
 S3Manager *manager.Uploader
}



// DeleteObjects deletes a list of objects from a bucket.
func (actor S3Actions) DeleteObjects(ctx context.Context, bucket string, objects
 []types.ObjectIdentifier, bypassGovernance bool) error {
 if len(objects) == 0 {
  return nil
 }

 input := s3.DeleteObjectsInput{
  Bucket: aws.String(bucket),
  Delete: &types.Delete{
   Objects: objects,
   Quiet:   aws.Bool(true),
  },
 }
 if bypassGovernance {
  input.BypassGovernanceRetention = aws.Bool(true)
 }
 delOut, err := actor.S3Client.DeleteObjects(ctx, &input)
 if err != nil || len(delOut.Errors) > 0 {
  log.Printf("Error deleting objects from bucket %s.\n", bucket)
  if err != nil {
   var noBucket *types.NoSuchBucket
```

```
    if errors.As(err, &noBucket) {
     log.Printf("Bucket %s does not exist.\n", bucket)
     err = noBucket
    }
   } else if len(delOut.Errors) > 0 {
    for _, outErr := range delOut.Errors {
     log.Printf("%s: %s\n", *outErr.Key, *outErr.Message)
    }
    err = fmt.Errorf("%s", *delOut.Errors[0].Message)
   }
  } else {
   for _, delObjs := range delOut.Deleted {
    err = s3.NewObjectNotExistsWaiter(actor.S3Client).Wait(
     ctx, &s3.HeadObjectInput{Bucket: aws.String(bucket), Key: delObjs.Key},
  time.Minute)
    if err != nil {
     log.Printf("Failed attempt to wait for object %s to be deleted.\n",
  *delObjs.Key)
    } else {
     log.Printf("Deleted %s.\n", *delObjs.Key)
    }
   }
  }
  return err
}
```

- Einzelheiten zur API finden Sie DeleteObjectsin der AWS SDK für Go API-Referenz.

`GetObject`

Das folgende Codebeispiel zeigt die Verwendung`GetObject`.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import (
 "bytes"
 "context"
 "errors"
 "fmt"
 "io"
 "log"
 "os"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/s3/manager"
 "github.com/aws/aws-sdk-go-v2/service/s3"
 "github.com/aws/aws-sdk-go-v2/service/s3/types"
 "github.com/aws/smithy-go"
)

// BucketBasics encapsulates the Amazon Simple Storage Service (Amazon S3) actions
// used in the examples.
// It contains S3Client, an Amazon S3 service client that is used to perform bucket
// and object actions.
type BucketBasics struct {
 S3Client *s3.Client
}



// DownloadFile gets an object from a bucket and stores it in a local file.
func (basics BucketBasics) DownloadFile(ctx context.Context, bucketName string,
 objectKey string, fileName string) error {
 result, err := basics.S3Client.GetObject(ctx, &s3.GetObjectInput{
  Bucket: aws.String(bucketName),
  Key:    aws.String(objectKey),
 })
 if err != nil {
  var noKey *types.NoSuchKey
  if errors.As(err, &noKey) {
   log.Printf("Can't get object %s from bucket %s. No such key exists.\n",
 objectKey, bucketName)
   err = noKey
  } else {
```

```
    log.Printf("Couldn't get object %v:%v. Here's why: %v\n", bucketName, objectKey,
 err)
  }
  return err
 }
 defer result.Body.Close()
 file, err := os.Create(fileName)
 if err != nil {
  log.Printf("Couldn't create file %v. Here's why: %v\n", fileName, err)
  return err
 }
 defer file.Close()
 body, err := io.ReadAll(result.Body)
 if err != nil {
  log.Printf("Couldn't read object body from %v. Here's why: %v\n", objectKey, err)
 }
 _, err = file.Write(body)
 return err
}
```

- Einzelheiten zur API finden Sie GetObjectin der AWS SDK für Go API-Referenz.

`GetObjectLegalHold`

Das folgende Codebeispiel zeigt die Verwendung`GetObjectLegalHold`.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import (
 "bytes"
 "context"
 "errors"
 "fmt"
```

```
 "log"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/s3/manager"
 "github.com/aws/aws-sdk-go-v2/service/s3"
 "github.com/aws/aws-sdk-go-v2/service/s3/types"
 "github.com/aws/smithy-go"
)

// S3Actions wraps S3 service actions.
type S3Actions struct {
 S3Client  *s3.Client
 S3Manager *manager.Uploader
}



// GetObjectLegalHold retrieves the legal hold status for an S3 object.
func (actor S3Actions) GetObjectLegalHold(ctx context.Context, bucket string, key
 string, versionId string) (*types.ObjectLockLegalHoldStatus, error) {
 var status *types.ObjectLockLegalHoldStatus
 input := &s3.GetObjectLegalHoldInput{
  Bucket:    aws.String(bucket),
  Key:       aws.String(key),
  VersionId: aws.String(versionId),
 }

 output, err := actor.S3Client.GetObjectLegalHold(ctx, input)
 if err != nil {
  var noSuchKeyErr *types.NoSuchKey
  var apiErr *smithy.GenericAPIError
  if errors.As(err, &noSuchKeyErr) {
   log.Printf("Object %s does not exist in bucket %s.\n", key, bucket)
   err = noSuchKeyErr
  } else if errors.As(err, &apiErr) {
   switch apiErr.ErrorCode() {
   case "NoSuchObjectLockConfiguration":
    log.Printf("Object %s does not have an object lock configuration.\n", key)
    err = nil
   case "InvalidRequest":
    log.Printf("Bucket %s does not have an object lock configuration.\n", bucket)
    err = nil
   }
```

```
  }
 } else {
  status = &output.LegalHold.Status
 }

 return status, err
}
```

- Einzelheiten zur API finden Sie GetObjectLegalHoldin der AWS SDK für Go API-Referenz.

`GetObjectLockConfiguration`

Das folgende Codebeispiel zeigt die Verwendung`GetObjectLockConfiguration`.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import (
 "bytes"
 "context"
 "errors"
 "fmt"
 "log"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/s3/manager"
 "github.com/aws/aws-sdk-go-v2/service/s3"
 "github.com/aws/aws-sdk-go-v2/service/s3/types"
 "github.com/aws/smithy-go"
)

// S3Actions wraps S3 service actions.
```

```
type S3Actions struct {
 S3Client  *s3.Client
 S3Manager *manager.Uploader
}



// GetObjectLockConfiguration retrieves the object lock configuration for an S3
 bucket.
func (actor S3Actions) GetObjectLockConfiguration(ctx context.Context, bucket
 string) (*types.ObjectLockConfiguration, error) {
 var lockConfig *types.ObjectLockConfiguration
 input := &s3.GetObjectLockConfigurationInput{
  Bucket: aws.String(bucket),
 }

 output, err := actor.S3Client.GetObjectLockConfiguration(ctx, input)
 if err != nil {
  var noBucket *types.NoSuchBucket
  var apiErr *smithy.GenericAPIError
  if errors.As(err, &noBucket) {
   log.Printf("Bucket %s does not exist.\n", bucket)
   err = noBucket
  } else if errors.As(err, &apiErr) && apiErr.ErrorCode() ==
 "ObjectLockConfigurationNotFoundError" {
   log.Printf("Bucket %s does not have an object lock configuration.\n", bucket)
   err = nil
  }
 } else {
  lockConfig = output.ObjectLockConfiguration
 }

 return lockConfig, err
}
```

- Einzelheiten zur API finden Sie GetObjectLockConfigurationin der AWS SDK für Go API-Referenz.

`GetObjectRetention`

Das folgende Codebeispiel zeigt die Verwendung`GetObjectRetention`.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import (
 "bytes"
 "context"
 "errors"
 "fmt"
 "log"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/s3/manager"
 "github.com/aws/aws-sdk-go-v2/service/s3"
 "github.com/aws/aws-sdk-go-v2/service/s3/types"
 "github.com/aws/smithy-go"
)

// S3Actions wraps S3 service actions.
type S3Actions struct {
 S3Client  *s3.Client
 S3Manager *manager.Uploader
}



// GetObjectRetention retrieves the object retention configuration for an S3 object.
func (actor S3Actions) GetObjectRetention(ctx context.Context, bucket string, key
 string) (*types.ObjectLockRetention, error) {
 var retention *types.ObjectLockRetention
 input := &s3.GetObjectRetentionInput{
  Bucket: aws.String(bucket),
  Key:    aws.String(key),
 }

 output, err := actor.S3Client.GetObjectRetention(ctx, input)
```

```
 if err != nil {
  var noKey *types.NoSuchKey
  var apiErr *smithy.GenericAPIError
  if errors.As(err, &noKey) {
   log.Printf("Object %s does not exist in bucket %s.\n", key, bucket)
   err = noKey
  } else if errors.As(err, &apiErr) {
   switch apiErr.ErrorCode() {
   case "NoSuchObjectLockConfiguration":
    err = nil
   case "InvalidRequest":
    log.Printf("Bucket %s does not have locking enabled.", bucket)
    err = nil
   }
  }
 } else {
  retention = output.Retention
 }

 return retention, err
}
```

- Einzelheiten zur API finden Sie GetObjectRetentionin der AWS SDK für Go API-Referenz.

`HeadBucket`

Das folgende Codebeispiel zeigt die Verwendung`HeadBucket`.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import (
 "bytes"
 "context"
```

```
 "errors"
 "fmt"
 "io"
 "log"
 "os"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/s3/manager"
 "github.com/aws/aws-sdk-go-v2/service/s3"
 "github.com/aws/aws-sdk-go-v2/service/s3/types"
 "github.com/aws/smithy-go"
)

// BucketBasics encapsulates the Amazon Simple Storage Service (Amazon S3) actions
// used in the examples.
// It contains S3Client, an Amazon S3 service client that is used to perform bucket
// and object actions.
type BucketBasics struct {
 S3Client *s3.Client
}



// BucketExists checks whether a bucket exists in the current account.
func (basics BucketBasics) BucketExists(ctx context.Context, bucketName string)
 (bool, error) {
 _, err := basics.S3Client.HeadBucket(ctx, &s3.HeadBucketInput{
  Bucket: aws.String(bucketName),
 })
 exists := true
 if err != nil {
  var apiError smithy.APIError
  if errors.As(err, &apiError) {
   switch apiError.(type) {
   case *types.NotFound:
    log.Printf("Bucket %v is available.\n", bucketName)
    exists = false
    err = nil
   default:
    log.Printf("Either you don't have access to bucket %v or another error occurred.
 "+
     "Here's what happened: %v\n", bucketName, err)
   }
```

```
  }
 } else {
  log.Printf("Bucket %v exists and you already own it.", bucketName)
 }

 return exists, err
}
```

- Einzelheiten zur API finden Sie HeadBucketin der AWS SDK für Go API-Referenz.

`ListBuckets`

Das folgende Codebeispiel zeigt die Verwendung`ListBuckets`.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import (
 "bytes"
 "context"
 "errors"
 "fmt"
 "io"
 "log"
 "os"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/s3/manager"
 "github.com/aws/aws-sdk-go-v2/service/s3"
 "github.com/aws/aws-sdk-go-v2/service/s3/types"
 "github.com/aws/smithy-go"
)
```

```
// BucketBasics encapsulates the Amazon Simple Storage Service (Amazon S3) actions
// used in the examples.
// It contains S3Client, an Amazon S3 service client that is used to perform bucket
// and object actions.
type BucketBasics struct {
 S3Client *s3.Client
}



// ListBuckets lists the buckets in the current account.
func (basics BucketBasics) ListBuckets(ctx context.Context) ([]types.Bucket, error)
 {
 var err error
 var output *s3.ListBucketsOutput
 var buckets []types.Bucket
 bucketPaginator := s3.NewListBucketsPaginator(basics.S3Client,
 &s3.ListBucketsInput{})
 for bucketPaginator.HasMorePages() {
  output, err = bucketPaginator.NextPage(ctx)
  if err != nil {
   var apiErr smithy.APIError
   if errors.As(err, &apiErr) && apiErr.ErrorCode() == "AccessDenied" {
    fmt.Println("You don't have permission to list buckets for this account.")
    err = apiErr
   } else {
    log.Printf("Couldn't list buckets for your account. Here's why: %v\n", err)
   }
   break
  } else {
   buckets = append(buckets, output.Buckets...)
  }
 }
 return buckets, err
}
```

- Einzelheiten zur API finden Sie ListBucketsin der AWS SDK für Go API-Referenz.

`ListObjectVersions`

Das folgende Codebeispiel zeigt die Verwendung`ListObjectVersions`.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import (
 "bytes"
 "context"
 "errors"
 "fmt"
 "log"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/s3/manager"
 "github.com/aws/aws-sdk-go-v2/service/s3"
 "github.com/aws/aws-sdk-go-v2/service/s3/types"
 "github.com/aws/smithy-go"
)

// S3Actions wraps S3 service actions.
type S3Actions struct {
 S3Client  *s3.Client
 S3Manager *manager.Uploader
}



// ListObjectVersions lists all versions of all objects in a bucket.
func (actor S3Actions) ListObjectVersions(ctx context.Context, bucket string)
 ([]types.ObjectVersion, error) {
 var err error
 var output *s3.ListObjectVersionsOutput
 var versions []types.ObjectVersion
 input := &s3.ListObjectVersionsInput{Bucket: aws.String(bucket)}
 versionPaginator := s3.NewListObjectVersionsPaginator(actor.S3Client, input)
 for versionPaginator.HasMorePages() {
  output, err = versionPaginator.NextPage(ctx)
```

```
  if err != nil {
   var noBucket *types.NoSuchBucket
   if errors.As(err, &noBucket) {
    log.Printf("Bucket %s does not exist.\n", bucket)
    err = noBucket
   }
   break
  } else {
   versions = append(versions, output.Versions...)
  }
 }
 return versions, err
}
```

- Einzelheiten zur API finden Sie ListObjectVersionsin der AWS SDK für Go API-Referenz.

`ListObjectsV2`

Das folgende Codebeispiel zeigt die Verwendung`ListObjectsV2`.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import (
 "bytes"
 "context"
 "errors"
 "fmt"
 "io"
 "log"
 "os"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
```

```
 "github.com/aws/aws-sdk-go-v2/feature/s3/manager"
 "github.com/aws/aws-sdk-go-v2/service/s3"
 "github.com/aws/aws-sdk-go-v2/service/s3/types"
 "github.com/aws/smithy-go"
)

// BucketBasics encapsulates the Amazon Simple Storage Service (Amazon S3) actions
// used in the examples.
// It contains S3Client, an Amazon S3 service client that is used to perform bucket
// and object actions.
type BucketBasics struct {
 S3Client *s3.Client
}



// ListObjects lists the objects in a bucket.
func (basics BucketBasics) ListObjects(ctx context.Context, bucketName string)
 ([]types.Object, error) {
 var err error
 var output *s3.ListObjectsV2Output
 input := &s3.ListObjectsV2Input{
  Bucket: aws.String(bucketName),
 }
 var objects []types.Object
 objectPaginator := s3.NewListObjectsV2Paginator(basics.S3Client, input)
 for objectPaginator.HasMorePages() {
  output, err = objectPaginator.NextPage(ctx)
  if err != nil {
   var noBucket *types.NoSuchBucket
   if errors.As(err, &noBucket) {
    log.Printf("Bucket %s does not exist.\n", bucketName)
    err = noBucket
   }
   break
  } else {
   objects = append(objects, output.Contents...)
  }
 }
 return objects, err
}
```

- Einzelheiten zur API finden Sie unter ListObjectsV2 in der AWS SDK für Go API-Referenz.

`PutObject`

Das folgende Codebeispiel zeigt die Verwendung`PutObject`.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Fügen Sie mithilfe der Low-Level-API ein Objekt in einen Bucket ein.

```
import (
 "bytes"
 "context"
 "errors"
 "fmt"
 "io"
 "log"
 "os"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/s3/manager"
 "github.com/aws/aws-sdk-go-v2/service/s3"
 "github.com/aws/aws-sdk-go-v2/service/s3/types"
 "github.com/aws/smithy-go"
)

// BucketBasics encapsulates the Amazon Simple Storage Service (Amazon S3) actions
// used in the examples.
// It contains S3Client, an Amazon S3 service client that is used to perform bucket
// and object actions.
type BucketBasics struct {
 S3Client *s3.Client
}
```

```
// UploadFile reads from a file and puts the data into an object in a bucket.
func (basics BucketBasics) UploadFile(ctx context.Context, bucketName string,
 objectKey string, fileName string) error {
 file, err := os.Open(fileName)
 if err != nil {
  log.Printf("Couldn't open file %v to upload. Here's why: %v\n", fileName, err)
 } else {
  defer file.Close()
  _, err = basics.S3Client.PutObject(ctx, &s3.PutObjectInput{
   Bucket: aws.String(bucketName),
   Key:    aws.String(objectKey),
   Body:   file,
  })
  if err != nil {
   var apiErr smithy.APIError
   if errors.As(err, &apiErr) && apiErr.ErrorCode() == "EntityTooLarge" {
    log.Printf("Error while uploading object to %s. The object is too large.\n"+
     "To upload objects larger than 5GB, use the S3 console (160GB max)\n"+
     "or the multipart upload API (5TB max).", bucketName)
   } else {
    log.Printf("Couldn't upload file %v to %v:%v. Here's why: %v\n",
     fileName, bucketName, objectKey, err)
   }
  } else {
   err = s3.NewObjectExistsWaiter(basics.S3Client).Wait(
    ctx, &s3.HeadObjectInput{Bucket: aws.String(bucketName), Key:
 aws.String(objectKey)}, time.Minute)
   if err != nil {
    log.Printf("Failed attempt to wait for object %s to exist.\n", objectKey)
   }
  }
 }
 return err
}
```

Laden Sie ein Objekt mithilfe eines Transfer-Managers in einen Bucket hoch.

```
import (
```

```
 "bytes"
 "context"
 "errors"
 "fmt"
 "log"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/s3/manager"
 "github.com/aws/aws-sdk-go-v2/service/s3"
 "github.com/aws/aws-sdk-go-v2/service/s3/types"
 "github.com/aws/smithy-go"
)

// S3Actions wraps S3 service actions.
type S3Actions struct {
 S3Client  *s3.Client
 S3Manager *manager.Uploader
}



// UploadObject uses the S3 upload manager to upload an object to a bucket.
func (actor S3Actions) UploadObject(ctx context.Context, bucket string, key string,
 contents string) (string, error) {
 var outKey string
 input := &s3.PutObjectInput{
  Bucket:            aws.String(bucket),
  Key:               aws.String(key),
  Body:              bytes.NewReader([]byte(contents)),
  ChecksumAlgorithm: types.ChecksumAlgorithmSha256,
 }
 output, err := actor.S3Manager.Upload(ctx, input)
 if err != nil {
  var noBucket *types.NoSuchBucket
  if errors.As(err, &noBucket) {
   log.Printf("Bucket %s does not exist.\n", bucket)
   err = noBucket
  }
 } else {
  err := s3.NewObjectExistsWaiter(actor.S3Client).Wait(ctx, &s3.HeadObjectInput{
   Bucket: aws.String(bucket),
   Key:    aws.String(key),
  }, time.Minute)
```

```
  if err != nil {
   log.Printf("Failed attempt to wait for object %s to exist in %s.\n", key, bucket)
  } else {
   outKey = *output.Key
  }
 }
 return outKey, err
}
```

- Einzelheiten zur API finden Sie PutObjectunter AWS SDK für Go API-Referenz.

`PutObjectLegalHold`

Das folgende Codebeispiel zeigt die Verwendung`PutObjectLegalHold`.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import (
 "bytes"
 "context"
 "errors"
 "fmt"
 "log"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/s3/manager"
 "github.com/aws/aws-sdk-go-v2/service/s3"
 "github.com/aws/aws-sdk-go-v2/service/s3/types"
 "github.com/aws/smithy-go"
)
```

```
// S3Actions wraps S3 service actions.
type S3Actions struct {
 S3Client  *s3.Client
 S3Manager *manager.Uploader
}



// PutObjectLegalHold sets the legal hold configuration for an S3 object.
func (actor S3Actions) PutObjectLegalHold(ctx context.Context, bucket string, key
 string, versionId string, legalHoldStatus types.ObjectLockLegalHoldStatus) error {
 input := &s3.PutObjectLegalHoldInput{
  Bucket: aws.String(bucket),
  Key:    aws.String(key),
  LegalHold: &types.ObjectLockLegalHold{
   Status: legalHoldStatus,
  },
 }
 if versionId != "" {
  input.VersionId = aws.String(versionId)
 }

 _, err := actor.S3Client.PutObjectLegalHold(ctx, input)
 if err != nil {
  var noKey *types.NoSuchKey
  if errors.As(err, &noKey) {
   log.Printf("Object %s does not exist in bucket %s.\n", key, bucket)
   err = noKey
  }
 }

 return err
}
```

- Einzelheiten zur API finden Sie PutObjectLegalHold in der AWS SDK für Go API-Referenz.

`PutObjectLockConfiguration`

Das folgende Codebeispiel zeigt die Verwendung `PutObjectLockConfiguration`.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Legt die Objektsperrkonfiguration eines Buckets fest.

```
import (
 "bytes"
 "context"
 "errors"
 "fmt"
 "log"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/s3/manager"
 "github.com/aws/aws-sdk-go-v2/service/s3"
 "github.com/aws/aws-sdk-go-v2/service/s3/types"
 "github.com/aws/smithy-go"
)

// S3Actions wraps S3 service actions.
type S3Actions struct {
 S3Client  *s3.Client
 S3Manager *manager.Uploader
}



// EnableObjectLockOnBucket enables object locking on an existing bucket.
func (actor S3Actions) EnableObjectLockOnBucket(ctx context.Context, bucket string)
 error {
 // Versioning must be enabled on the bucket before object locking is enabled.
 verInput := &s3.PutBucketVersioningInput{
  Bucket: aws.String(bucket),
  VersioningConfiguration: &types.VersioningConfiguration{
   MFADelete: types.MFADeleteDisabled,
   Status:    types.BucketVersioningStatusEnabled,
```

```
  },
 }
 _, err := actor.S3Client.PutBucketVersioning(ctx, verInput)
 if err != nil {
  var noBucket *types.NoSuchBucket
  if errors.As(err, &noBucket) {
   log.Printf("Bucket %s does not exist.\n", bucket)
   err = noBucket
  }
  return err
 }

 input := &s3.PutObjectLockConfigurationInput{
  Bucket: aws.String(bucket),
  ObjectLockConfiguration: &types.ObjectLockConfiguration{
   ObjectLockEnabled: types.ObjectLockEnabledEnabled,
  },
 }
 _, err = actor.S3Client.PutObjectLockConfiguration(ctx, input)
 if err != nil {
  var noBucket *types.NoSuchBucket
  if errors.As(err, &noBucket) {
   log.Printf("Bucket %s does not exist.\n", bucket)
   err = noBucket
  }
 }

 return err
}
```

Legen Sie die Standardaufbewahrungsdauer eines Buckets fest.

```
import (
 "bytes"
 "context"
 "errors"
 "fmt"
 "log"
 "time"
```

```
 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/s3/manager"
 "github.com/aws/aws-sdk-go-v2/service/s3"
 "github.com/aws/aws-sdk-go-v2/service/s3/types"
 "github.com/aws/smithy-go"
)

// S3Actions wraps S3 service actions.
type S3Actions struct {
 S3Client  *s3.Client
 S3Manager *manager.Uploader
}



// ModifyDefaultBucketRetention modifies the default retention period of an existing
 bucket.
func (actor S3Actions) ModifyDefaultBucketRetention(
 ctx context.Context, bucket string, lockMode types.ObjectLockEnabled,
 retentionPeriod int32, retentionMode types.ObjectLockRetentionMode) error {

 input := &s3.PutObjectLockConfigurationInput{
  Bucket: aws.String(bucket),
  ObjectLockConfiguration: &types.ObjectLockConfiguration{
   ObjectLockEnabled: lockMode,
   Rule: &types.ObjectLockRule{
    DefaultRetention: &types.DefaultRetention{
     Days: aws.Int32(retentionPeriod),
     Mode: retentionMode,
    },
   },
  },
 }
 _, err := actor.S3Client.PutObjectLockConfiguration(ctx, input)
 if err != nil {
  var noBucket *types.NoSuchBucket
  if errors.As(err, &noBucket) {
   log.Printf("Bucket %s does not exist.\n", bucket)
   err = noBucket
  }
 }

 return err
}
```

- Einzelheiten zur API finden Sie PutObjectLockConfigurationunter AWS SDK für Go API-Referenz.

`PutObjectRetention`

Das folgende Codebeispiel zeigt die Verwendung`PutObjectRetention`.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import (
 "bytes"
 "context"
 "errors"
 "fmt"
 "log"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/s3/manager"
 "github.com/aws/aws-sdk-go-v2/service/s3"
 "github.com/aws/aws-sdk-go-v2/service/s3/types"
 "github.com/aws/smithy-go"
)

// S3Actions wraps S3 service actions.
type S3Actions struct {
 S3Client  *s3.Client
 S3Manager *manager.Uploader
}
```

```
// PutObjectRetention sets the object retention configuration for an S3 object.
func (actor S3Actions) PutObjectRetention(ctx context.Context, bucket string, key
 string, retentionMode types.ObjectLockRetentionMode, retentionPeriodDays int32)
 error {
 input := &s3.PutObjectRetentionInput{
  Bucket: aws.String(bucket),
  Key:    aws.String(key),
  Retention: &types.ObjectLockRetention{
   Mode:            retentionMode,
   RetainUntilDate: aws.Time(time.Now().AddDate(0, 0, int(retentionPeriodDays))),
  },
  BypassGovernanceRetention: aws.Bool(true),
 }

 _, err := actor.S3Client.PutObjectRetention(ctx, input)
 if err != nil {
  var noKey *types.NoSuchKey
  if errors.As(err, &noKey) {
   log.Printf("Object %s does not exist in bucket %s.\n", key, bucket)
   err = noKey
  }
 }

 return err
}
```

- Einzelheiten zur API finden Sie PutObjectRetentionin der AWS SDK für Go API-Referenz.

Szenarien

Eine vorsignierte URL erstellen

Das folgende Codebeispiel zeigt, wie Sie eine vorsignierte URL für Amazon S3 erstellen und ein Objekt hochladen.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Erstellen Sie Funktionen, die S3-Vorsignierungsaktionen umschließen.

```
import (
 "context"
 "log"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 v4 "github.com/aws/aws-sdk-go-v2/aws/signer/v4"
 "github.com/aws/aws-sdk-go-v2/service/s3"
)

// Presigner encapsulates the Amazon Simple Storage Service (Amazon S3) presign
 actions
// used in the examples.
// It contains PresignClient, a client that is used to presign requests to Amazon
 S3.
// Presigned requests contain temporary credentials and can be made from any HTTP
 client.
type Presigner struct {
 PresignClient *s3.PresignClient
}



// GetObject makes a presigned request that can be used to get an object from a
 bucket.
// The presigned request is valid for the specified number of seconds.
func (presigner Presigner) GetObject(
 ctx context.Context, bucketName string, objectKey string, lifetimeSecs int64)
 (*v4.PresignedHTTPRequest, error) {
 request, err := presigner.PresignClient.PresignGetObject(ctx, &s3.GetObjectInput{
  Bucket: aws.String(bucketName),
  Key:    aws.String(objectKey),
```

```
 }, func(opts *s3.PresignOptions) {
  opts.Expires = time.Duration(lifetimeSecs * int64(time.Second))
 })
 if err != nil {
  log.Printf("Couldn't get a presigned request to get %v:%v. Here's why: %v\n",
   bucketName, objectKey, err)
 }
 return request, err
}



// PutObject makes a presigned request that can be used to put an object in a
 bucket.
// The presigned request is valid for the specified number of seconds.
func (presigner Presigner) PutObject(
 ctx context.Context, bucketName string, objectKey string, lifetimeSecs int64)
 (*v4.PresignedHTTPRequest, error) {
 request, err := presigner.PresignClient.PresignPutObject(ctx, &s3.PutObjectInput{
  Bucket: aws.String(bucketName),
  Key:    aws.String(objectKey),
 }, func(opts *s3.PresignOptions) {
  opts.Expires = time.Duration(lifetimeSecs * int64(time.Second))
 })
 if err != nil {
  log.Printf("Couldn't get a presigned request to put %v:%v. Here's why: %v\n",
   bucketName, objectKey, err)
 }
 return request, err
}



// DeleteObject makes a presigned request that can be used to delete an object from
 a bucket.
func (presigner Presigner) DeleteObject(ctx context.Context, bucketName string,
 objectKey string) (*v4.PresignedHTTPRequest, error) {
 request, err := presigner.PresignClient.PresignDeleteObject(ctx,
 &s3.DeleteObjectInput{
  Bucket: aws.String(bucketName),
  Key:    aws.String(objectKey),
 })
 if err != nil {
```

```
  log.Printf("Couldn't get a presigned request to delete object %v. Here's why: %v
\n", objectKey, err)
 }
 return request, err
}



func (presigner Presigner) PresignPostObject(ctx context.Context, bucketName string,
 objectKey string, lifetimeSecs int64) (*s3.PresignedPostRequest, error) {
 request, err := presigner.PresignClient.PresignPostObject(ctx, &s3.PutObjectInput{
  Bucket: aws.String(bucketName),
  Key:    aws.String(objectKey),
 }, func(options *s3.PresignPostOptions) {
  options.Expires = time.Duration(lifetimeSecs) * time.Second
 })
 if err != nil {
  log.Printf("Couldn't get a presigned post request to put %v:%v. Here's why: %v\n",
 bucketName, objectKey, err)
 }
 return request, nil
}
```

Führen Sie ein interaktives Beispiel aus, das Presigned generiert und verwendet, URLs um ein S3-Objekt hochzuladen, herunterzuladen und zu löschen.

```
import (
 "bytes"
 "context"
 "io"
 "log"
 "mime/multipart"
 "net/http"
 "os"
 "strings"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/s3"
 "github.com/awsdocs/aws-doc-sdk-examples/gov2/demotools"
```

```
 "github.com/awsdocs/aws-doc-sdk-examples/gov2/s3/actions"
)



// RunPresigningScenario is an interactive example that shows you how to get
 presigned
// HTTP requests that you can use to move data into and out of Amazon Simple Storage
// Service (Amazon S3). The presigned requests contain temporary credentials and can
// be used by an HTTP client.
//
// 1. Get a presigned request to put an object in a bucket.
// 2. Use the net/http package to use the presigned request to upload a local file
 to the bucket.
// 3. Get a presigned request to get an object from a bucket.
// 4. Use the net/http package to use the presigned request to download the object
 to a local file.
// 5. Get a presigned request to delete an object from a bucket.
// 6. Use the net/http package to use the presigned request to delete the object.
//
// This example creates an Amazon S3 presign client from the specified sdkConfig so
 that
// you can replace it with a mocked or stubbed config for unit testing.
//
// It uses a questioner from the `demotools` package to get input during the
 example.
// This package can be found in the ..\..\demotools folder of this repo.
//
// It uses an IHttpRequester interface to abstract HTTP requests so they can be
 mocked
// during testing.
func RunPresigningScenario(ctx context.Context, sdkConfig aws.Config, questioner
 demotools.IQuestioner, httpRequester IHttpRequester) {
 defer func() {
  if r := recover(); r != nil {
   log.Println("Something went wrong with the demo.")
   _, isMock := questioner.(*demotools.MockQuestioner)
   if isMock || questioner.AskBool("Do you want to see the full error message (y/
n)?", "y") {
    log.Println(r)
   }
  }
 }()
```

```
log.Println(strings.Repeat("-", 88))
log.Println("Welcome to the Amazon S3 presigning demo.")
log.Println(strings.Repeat("-", 88))

s3Client := s3.NewFromConfig(sdkConfig)
bucketBasics := actions.BucketBasics{S3Client: s3Client}
presignClient := s3.NewPresignClient(s3Client)
presigner := actions.Presigner{PresignClient: presignClient}

bucketName := questioner.Ask("We'll need a bucket. Enter a name for a bucket "+
 "you own or one you want to create:", demotools.NotEmpty{})
bucketExists, err := bucketBasics.BucketExists(ctx, bucketName)
if err != nil {
 panic(err)
}
if !bucketExists {
 err = bucketBasics.CreateBucket(ctx, bucketName, sdkConfig.Region)
 if err != nil {
  panic(err)
 } else {
  log.Println("Bucket created.")
 }
}
log.Println(strings.Repeat("-", 88))

log.Printf("Let's presign a request to upload a file to your bucket.")
uploadFilename := questioner.Ask("Enter the path to a file you want to upload:",
 demotools.NotEmpty{})
uploadKey := questioner.Ask("What would you like to name the uploaded object?",
 demotools.NotEmpty{})
uploadFile, err := os.Open(uploadFilename)
if err != nil {
 panic(err)
}
defer uploadFile.Close()
presignedPutRequest, err := presigner.PutObject(ctx, bucketName, uploadKey, 60)
if err != nil {
 panic(err)
}
log.Printf("Got a presigned %v request to URL:\n\t%v\n",
presignedPutRequest.Method,
 presignedPutRequest.URL)
log.Println("Using net/http to send the request...")
info, err := uploadFile.Stat()
```

```
if err != nil {
 panic(err)
}
putResponse, err := httpRequester.Put(presignedPutRequest.URL, info.Size(),
uploadFile)
if err != nil {
 panic(err)
}
log.Printf("%v object %v with presigned URL returned %v.",
presignedPutRequest.Method,
 uploadKey, putResponse.StatusCode)
log.Println(strings.Repeat("-", 88))

log.Printf("Let's presign a request to download the object.")
questioner.Ask("Press Enter when you're ready.")
presignedGetRequest, err := presigner.GetObject(ctx, bucketName, uploadKey, 60)
if err != nil {
 panic(err)
}
log.Printf("Got a presigned %v request to URL:\n\t%v\n",
presignedGetRequest.Method,
 presignedGetRequest.URL)
log.Println("Using net/http to send the request...")
getResponse, err := httpRequester.Get(presignedGetRequest.URL)
if err != nil {
 panic(err)
}
log.Printf("%v object %v with presigned URL returned %v.",
presignedGetRequest.Method,
 uploadKey, getResponse.StatusCode)
defer getResponse.Body.Close()
downloadBody, err := io.ReadAll(getResponse.Body)
if err != nil {
 panic(err)
}
log.Printf("Downloaded %v bytes. Here are the first 100 of them:\n",
len(downloadBody))
log.Println(strings.Repeat("-", 88))
log.Println(string(downloadBody[:100]))
log.Println(strings.Repeat("-", 88))

log.Println("Now we'll create a new request to put the same object using a
presigned post request")
questioner.Ask("Press Enter when you're ready.")
```

```
 presignPostRequest, err := presigner.PresignPostObject(ctx, bucketName, uploadKey,
 60)
 if err != nil {
  panic(err)
 }
 log.Printf("Got a presigned post request to url %v with values %v\n",
 presignPostRequest.URL, presignPostRequest.Values)
 log.Println("Using net/http multipart to send the request...")
 uploadFile, err = os.Open(uploadFilename)
 if err != nil {
  panic(err)
 }
 defer uploadFile.Close()
 multiPartResponse, err := sendMultipartRequest(presignPostRequest.URL,
 presignPostRequest.Values, uploadFile, uploadKey, httpRequester)
 if err != nil {
  panic(err)
 }
 log.Printf("Presign post object %v with presigned URL returned %v.", uploadKey,
 multiPartResponse.StatusCode)

 log.Println("Let's presign a request to delete the object.")
 questioner.Ask("Press Enter when you're ready.")
 presignedDelRequest, err := presigner.DeleteObject(ctx, bucketName, uploadKey)
 if err != nil {
  panic(err)
 }
 log.Printf("Got a presigned %v request to URL:\n\t%v\n",
 presignedDelRequest.Method,
  presignedDelRequest.URL)
 log.Println("Using net/http to send the request...")
 delResponse, err := httpRequester.Delete(presignedDelRequest.URL)
 if err != nil {
  panic(err)
 }
 log.Printf("%v object %v with presigned URL returned %v.\n",
 presignedDelRequest.Method,
  uploadKey, delResponse.StatusCode)
 log.Println(strings.Repeat("-", 88))

 log.Println("Thanks for watching!")
 log.Println(strings.Repeat("-", 88))
}
```

Definieren Sie einen HTTP-Request-Wrapper, der im Beispiel verwendet wird, um HTTP-Anfragen zu stellen.

```
// IHttpRequester abstracts HTTP requests into an interface so it can be mocked
 during
// unit testing.
type IHttpRequester interface {
 Get(url string) (resp *http.Response, err error)
 Post(url, contentType string, body io.Reader) (resp *http.Response, err error)
 Put(url string, contentLength int64, body io.Reader) (resp *http.Response, err
 error)
 Delete(url string) (resp *http.Response, err error)
}

// HttpRequester uses the net/http package to make HTTP requests during the
 scenario.
type HttpRequester struct{}

func (httpReq HttpRequester) Get(url string) (resp *http.Response, err error) {
 return http.Get(url)
}
func (httpReq HttpRequester) Post(url, contentType string, body io.Reader) (resp
 *http.Response, err error) {
 postRequest, err := http.NewRequest("POST", url, body)
 if err != nil {
  return nil, err
 }
 postRequest.Header.Set("Content-Type", contentType)
 return http.DefaultClient.Do(postRequest)
}

func (httpReq HttpRequester) Put(url string, contentLength int64, body io.Reader)
 (resp *http.Response, err error) {
 putRequest, err := http.NewRequest("PUT", url, body)
 if err != nil {
  return nil, err
 }
 putRequest.ContentLength = contentLength
 return http.DefaultClient.Do(putRequest)
}
```

```
func (httpReq HttpRequester) Delete(url string) (resp *http.Response, err error) {
 delRequest, err := http.NewRequest("DELETE", url, nil)
 if err != nil {
  return nil, err
 }
 return http.DefaultClient.Do(delRequest)
}
```

Amazon S3 S3-Objekte sperren

Das folgende Codebeispiel zeigt, wie Sie mit den Funktionen zum Sperren von Objekten in S3 arbeiten.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Führen Sie ein interaktives Szenario aus, in dem die Objektsperrfunktionen von Amazon S3 demonstriert werden.

```
import (
 "context"
 "fmt"
 "log"
 "strings"

 "s3_object_lock/actions"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/s3/manager"
 "github.com/aws/aws-sdk-go-v2/service/s3"
 "github.com/aws/aws-sdk-go-v2/service/s3/types"
 "github.com/awsdocs/aws-doc-sdk-examples/gov2/demotools"
)
```

```
// ObjectLockScenario contains the steps to run the S3 Object Lock workflow.
type ObjectLockScenario struct {
 questioner demotools.IQuestioner
 resources  Resources
 s3Actions  *actions.S3Actions
 sdkConfig  aws.Config
}

// NewObjectLockScenario constructs a new ObjectLockScenario instance.
func NewObjectLockScenario(sdkConfig aws.Config, questioner demotools.IQuestioner)
 ObjectLockScenario {
 scenario := ObjectLockScenario{
  questioner: questioner,
  resources:  Resources{},
  s3Actions:  &actions.S3Actions{S3Client: s3.NewFromConfig(sdkConfig)},
  sdkConfig:  sdkConfig,
 }
 scenario.s3Actions.S3Manager = manager.NewUploader(scenario.s3Actions.S3Client)
 scenario.resources.init(scenario.s3Actions, questioner)
 return scenario
}

type nameLocked struct {
 name   string
 locked bool
}

var createInfo = []nameLocked{
 {"standard-bucket", false},
 {"lock-bucket", true},
 {"retention-bucket", false},
}

// CreateBuckets creates the S3 buckets required for the workflow.
func (scenario *ObjectLockScenario) CreateBuckets(ctx context.Context) {
 log.Println("Let's create some S3 buckets to use for this workflow.")
 success := false
 for !success {
  prefix := scenario.questioner.Ask(
   "This example creates three buckets. Enter a prefix to name your buckets
 (remember bucket names must be globally unique):")

  for _, info := range createInfo {
```

```
   log.Println(fmt.Sprintf("%s.%s", prefix, info.name))
   bucketName, err := scenario.s3Actions.CreateBucketWithLock(ctx, fmt.Sprintf("%s.
%s", prefix, info.name), scenario.sdkConfig.Region, info.locked)
   if err != nil {
    switch err.(type) {
    case *types.BucketAlreadyExists, *types.BucketAlreadyOwnedByYou:
     log.Printf("Couldn't create bucket %s.\n", bucketName)
    default:
     panic(err)
    }
    break
   }
   scenario.resources.demoBuckets[info.name] = &DemoBucket{
    name:       bucketName,
    objectKeys: []string{},
   }
   log.Printf("Created bucket %s.\n", bucketName)
  }

  if len(scenario.resources.demoBuckets) < len(createInfo) {
   scenario.resources.deleteBuckets(ctx)
  } else {
   success = true
  }
 }

 log.Println("S3 buckets created.")
 log.Println(strings.Repeat("-", 88))
}

// EnableLockOnBucket enables object locking on an existing bucket.
func (scenario *ObjectLockScenario) EnableLockOnBucket(ctx context.Context) {
 log.Println("\nA bucket can be configured to use object locking.")
 scenario.questioner.Ask("Press Enter to continue.")

 var err error
 bucket := scenario.resources.demoBuckets["retention-bucket"]
 err = scenario.s3Actions.EnableObjectLockOnBucket(ctx, bucket.name)
 if err != nil {
  switch err.(type) {
  case *types.NoSuchBucket:
   log.Printf("Couldn't enable object locking on bucket %s.\n", bucket.name)
  default:
   panic(err)
```

```
  }
 } else {
  log.Printf("Object locking enabled on bucket %s.", bucket.name)
 }

 log.Println(strings.Repeat("-", 88))
}

// SetDefaultRetentionPolicy sets a default retention governance policy on a bucket.
func (scenario *ObjectLockScenario) SetDefaultRetentionPolicy(ctx context.Context) {
 log.Println("\nA bucket can be configured to use object locking with a default
 retention period.")

 bucket := scenario.resources.demoBuckets["retention-bucket"]
 retentionPeriod := scenario.questioner.AskInt("Enter the default retention period
 in days: ")
 err := scenario.s3Actions.ModifyDefaultBucketRetention(ctx,
 bucket.name, types.ObjectLockEnabledEnabled, int32(retentionPeriod),
 types.ObjectLockRetentionModeGovernance)
 if err != nil {
  switch err.(type) {
  case *types.NoSuchBucket:
   log.Printf("Couldn't configure a default retention period on bucket %s.\n",
 bucket.name)
  default:
   panic(err)
  }
 } else {
  log.Printf("Default retention policy set on bucket %s with %d day retention
 period.", bucket.name, retentionPeriod)
  bucket.retentionEnabled = true
 }

 log.Println(strings.Repeat("-", 88))
}

// UploadTestObjects uploads test objects to the S3 buckets.
func (scenario *ObjectLockScenario) UploadTestObjects(ctx context.Context) {
 log.Println("Uploading test objects to S3 buckets.")

 for _, info := range createInfo {
  bucket := scenario.resources.demoBuckets[info.name]
  for i := 0; i < 2; i++ {
```

```
    key, err := scenario.s3Actions.UploadObject(ctx, bucket.name,
 fmt.Sprintf("example-%d", i),
    fmt.Sprintf("Example object content #%d in bucket %s.", i, bucket.name))
   if err != nil {
    switch err.(type) {
    case *types.NoSuchBucket:
     log.Printf("Couldn't upload %s to bucket %s.\n", key, bucket.name)
    default:
     panic(err)
    }
   } else {
    log.Printf("Uploaded %s to bucket %s.\n", key, bucket.name)
    bucket.objectKeys = append(bucket.objectKeys, key)
   }
  }
 }

 scenario.questioner.Ask("Test objects uploaded. Press Enter to continue.")
 log.Println(strings.Repeat("-", 88))
}

// SetObjectLockConfigurations sets object lock configurations on the test objects.
func (scenario *ObjectLockScenario) SetObjectLockConfigurations(ctx context.Context)
 {
 log.Println("Now let's set object lock configurations on individual objects.")

 buckets := []*DemoBucket{scenario.resources.demoBuckets["lock-bucket"],
 scenario.resources.demoBuckets["retention-bucket"]}
 for _, bucket := range buckets {
  for index, objKey := range bucket.objectKeys {
   switch index {
   case 0:
    if scenario.questioner.AskBool(fmt.Sprintf("\nDo you want to add a legal hold to
 %s in %s (y/n)? ", objKey, bucket.name), "y") {
     err := scenario.s3Actions.PutObjectLegalHold(ctx, bucket.name, objKey, "",
 types.ObjectLockLegalHoldStatusOn)
     if err != nil {
      switch err.(type) {
      case *types.NoSuchKey:
       log.Printf("Couldn't set legal hold on %s.\n", objKey)
      default:
       panic(err)
      }
     } else {
```

```
      log.Printf("Legal hold set on %s.\n", objKey)
     }
    }
   case 1:
    q := fmt.Sprintf("\nDo you want to add a 1 day Governance retention period to %s
 in %s?\n"+
     "Reminder: Only a user with the s3:BypassGovernanceRetention permission is able
 to delete this object\n"+
     "or its bucket until the retention period has expired. (y/n) ", objKey,
 bucket.name)
    if scenario.questioner.AskBool(q, "y") {
     err := scenario.s3Actions.PutObjectRetention(ctx, bucket.name, objKey,
 types.ObjectLockRetentionModeGovernance, 1)
     if err != nil {
      switch err.(type) {
      case *types.NoSuchKey:
       log.Printf("Couldn't set retention period on %s in %s.\n", objKey,
 bucket.name)
      default:
       panic(err)
      }
     } else {
      log.Printf("Retention period set to 1 for %s.", objKey)
      bucket.retentionEnabled = true
     }
    }
   }
  }
 }
 log.Println(strings.Repeat("-", 88))
}

const (
 ListAll = iota
 DeleteObject
 DeleteRetentionObject
 OverwriteObject
 ViewRetention
 ViewLegalHold
 Finish
)

// InteractWithObjects allows the user to interact with the objects and test the
 object lock configurations.
```

```
func (scenario *ObjectLockScenario) InteractWithObjects(ctx context.Context) {
 log.Println("Now you can interact with the objects to explore the object lock
 configurations.")
 interactiveChoices := []string{
  "List all objects and buckets.",
  "Attempt to delete an object.",
  "Attempt to delete an object with retention period bypass.",
  "Attempt to overwrite a file.",
  "View the retention settings for an object.",
  "View the legal hold settings for an object.",
  "Finish the workflow."}

 choice := ListAll
 for choice != Finish {
  objList := scenario.GetAllObjects(ctx)
  objChoices := scenario.makeObjectChoiceList(objList)
  choice = scenario.questioner.AskChoice("Choose an action from the menu:\n",
interactiveChoices)
  switch choice {
  case ListAll:
   log.Println("The current objects in the example buckets are:")
   for _, objChoice := range objChoices {
    log.Println("\t", objChoice)
   }
  case DeleteObject, DeleteRetentionObject:
   objChoice := scenario.questioner.AskChoice("Enter the number of the object to
delete:\n", objChoices)
   obj := objList[objChoice]
   deleted, err := scenario.s3Actions.DeleteObject(ctx, obj.bucket, obj.key,
obj.versionId, choice == DeleteRetentionObject)
   if err != nil {
    switch err.(type) {
    case *types.NoSuchKey:
     log.Println("Nothing to delete.")
    default:
     panic(err)
    }
   } else if deleted {
    log.Printf("Object %s deleted.\n", obj.key)
   }
  case OverwriteObject:
   objChoice := scenario.questioner.AskChoice("Enter the number of the object to
overwrite:\n", objChoices)
   obj := objList[objChoice]
```

```
  _, err := scenario.s3Actions.UploadObject(ctx, obj.bucket, obj.key,
fmt.Sprintf("New content in object %s.", obj.key))
  if err != nil {
   switch err.(type) {
   case *types.NoSuchBucket:
    log.Println("Couldn't upload to nonexistent bucket.")
   default:
    panic(err)
   }
  } else {
   log.Printf("Uploaded new content to object %s.\n", obj.key)
  }
 case ViewRetention:
  objChoice := scenario.questioner.AskChoice("Enter the number of the object to
view:\n", objChoices)
  obj := objList[objChoice]
  retention, err := scenario.s3Actions.GetObjectRetention(ctx, obj.bucket, obj.key)
  if err != nil {
   switch err.(type) {
   case *types.NoSuchKey:
    log.Printf("Can't get retention configuration for %s.\n", obj.key)
   default:
    panic(err)
   }
  } else if retention != nil {
   log.Printf("Object %s has retention mode %s until %v.\n", obj.key,
retention.Mode, retention.RetainUntilDate)
  } else {
   log.Printf("Object %s does not have object retention configured.\n", obj.key)
  }
 case ViewLegalHold:
  objChoice := scenario.questioner.AskChoice("Enter the number of the object to
view:\n", objChoices)
  obj := objList[objChoice]
  legalHold, err := scenario.s3Actions.GetObjectLegalHold(ctx, obj.bucket, obj.key,
obj.versionId)
  if err != nil {
   switch err.(type) {
   case *types.NoSuchKey:
    log.Printf("Can't get legal hold configuration for %s.\n", obj.key)
   default:
    panic(err)
   }
  } else if legalHold != nil {
```

```
    log.Printf("Object %s has legal hold %v.", obj.key, *legalHold)
   } else {
    log.Printf("Object %s does not have legal hold configured.", obj.key)
   }
  case Finish:
   log.Println("Let's clean up.")
  }
  log.Println(strings.Repeat("-", 88))
 }
}

type BucketKeyVersionId struct {
 bucket    string
 key       string
 versionId string
}

// GetAllObjects gets the object versions in the example S3 buckets and returns them
 in a flattened list.
func (scenario *ObjectLockScenario) GetAllObjects(ctx context.Context)
 []BucketKeyVersionId {
 var objectList []BucketKeyVersionId
 for _, info := range createInfo {
  bucket := scenario.resources.demoBuckets[info.name]
  versions, err := scenario.s3Actions.ListObjectVersions(ctx, bucket.name)
  if err != nil {
   switch err.(type) {
   case *types.NoSuchBucket:
    log.Printf("Couldn't get object versions for %s.\n", bucket.name)
   default:
    panic(err)
   }
  } else {
   for _, version := range versions {
    objectList = append(objectList,
     BucketKeyVersionId{bucket: bucket.name, key: *version.Key, versionId:
 *version.VersionId})
   }
  }
 }
 return objectList
}
```

```
// makeObjectChoiceList makes the object version list into a list of strings that
 are displayed
// as choices.
func (scenario *ObjectLockScenario) makeObjectChoiceList(bucketObjects
 []BucketKeyVersionId) []string {
 choices := make([]string, len(bucketObjects))
 for i := 0; i < len(bucketObjects); i++ {
  choices[i] = fmt.Sprintf("%s in %s with VersionId %s.",
   bucketObjects[i].key, bucketObjects[i].bucket, bucketObjects[i].versionId)
 }
 return choices
}

// Run runs the S3 Object Lock scenario.
func (scenario *ObjectLockScenario) Run(ctx context.Context) {
 defer func() {
  if r := recover(); r != nil {
   log.Println("Something went wrong with the demo.")
   _, isMock := scenario.questioner.(*demotools.MockQuestioner)
   if isMock || scenario.questioner.AskBool("Do you want to see the full error
 message (y/n)?", "y") {
    log.Println(r)
   }
   scenario.resources.Cleanup(ctx)
  }
 }()

 log.Println(strings.Repeat("-", 88))
 log.Println("Welcome to the Amazon S3 Object Lock Feature Scenario.")
 log.Println(strings.Repeat("-", 88))

 scenario.CreateBuckets(ctx)
 scenario.EnableLockOnBucket(ctx)
 scenario.SetDefaultRetentionPolicy(ctx)
 scenario.UploadTestObjects(ctx)
 scenario.SetObjectLockConfigurations(ctx)
 scenario.InteractWithObjects(ctx)

 scenario.resources.Cleanup(ctx)

 log.Println(strings.Repeat("-", 88))
 log.Println("Thanks for watching!")
 log.Println(strings.Repeat("-", 88))
}
```

Definieren Sie eine Struktur, die die in diesem Beispiel verwendeten S3-Aktionen umschließt.

```
import (
 "bytes"
 "context"
 "errors"
 "fmt"
 "log"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/s3/manager"
 "github.com/aws/aws-sdk-go-v2/service/s3"
 "github.com/aws/aws-sdk-go-v2/service/s3/types"
 "github.com/aws/smithy-go"
)

// S3Actions wraps S3 service actions.
type S3Actions struct {
 S3Client  *s3.Client
 S3Manager *manager.Uploader
}



// CreateBucketWithLock creates a new S3 bucket with optional object locking enabled
// and waits for the bucket to exist before returning.
func (actor S3Actions) CreateBucketWithLock(ctx context.Context, bucket string,
 region string, enableObjectLock bool) (string, error) {
 input := &s3.CreateBucketInput{
  Bucket: aws.String(bucket),
  CreateBucketConfiguration: &types.CreateBucketConfiguration{
   LocationConstraint: types.BucketLocationConstraint(region),
  },
 }

 if enableObjectLock {
  input.ObjectLockEnabledForBucket = aws.Bool(true)
 }
```

```
 _, err := actor.S3Client.CreateBucket(ctx, input)
 if err != nil {
  var owned *types.BucketAlreadyOwnedByYou
  var exists *types.BucketAlreadyExists
  if errors.As(err, &owned) {
   log.Printf("You already own bucket %s.\n", bucket)
   err = owned
  } else if errors.As(err, &exists) {
   log.Printf("Bucket %s already exists.\n", bucket)
   err = exists
  }
 } else {
  err = s3.NewBucketExistsWaiter(actor.S3Client).Wait(
   ctx, &s3.HeadBucketInput{Bucket: aws.String(bucket)}, time.Minute)
  if err != nil {
   log.Printf("Failed attempt to wait for bucket %s to exist.\n", bucket)
  }
 }

 return bucket, err
}



// GetObjectLegalHold retrieves the legal hold status for an S3 object.
func (actor S3Actions) GetObjectLegalHold(ctx context.Context, bucket string, key
 string, versionId string) (*types.ObjectLockLegalHoldStatus, error) {
 var status *types.ObjectLockLegalHoldStatus
 input := &s3.GetObjectLegalHoldInput{
  Bucket:    aws.String(bucket),
  Key:       aws.String(key),
  VersionId: aws.String(versionId),
 }

 output, err := actor.S3Client.GetObjectLegalHold(ctx, input)
 if err != nil {
  var noSuchKeyErr *types.NoSuchKey
  var apiErr *smithy.GenericAPIError
  if errors.As(err, &noSuchKeyErr) {
   log.Printf("Object %s does not exist in bucket %s.\n", key, bucket)
   err = noSuchKeyErr
  } else if errors.As(err, &apiErr) {
   switch apiErr.ErrorCode() {
```

```
   case "NoSuchObjectLockConfiguration":
    log.Printf("Object %s does not have an object lock configuration.\n", key)
    err = nil
   case "InvalidRequest":
    log.Printf("Bucket %s does not have an object lock configuration.\n", bucket)
    err = nil
   }
  }
 } else {
  status = &output.LegalHold.Status
 }

 return status, err
}



// GetObjectLockConfiguration retrieves the object lock configuration for an S3
 bucket.
func (actor S3Actions) GetObjectLockConfiguration(ctx context.Context, bucket
 string) (*types.ObjectLockConfiguration, error) {
 var lockConfig *types.ObjectLockConfiguration
 input := &s3.GetObjectLockConfigurationInput{
  Bucket: aws.String(bucket),
 }

 output, err := actor.S3Client.GetObjectLockConfiguration(ctx, input)
 if err != nil {
  var noBucket *types.NoSuchBucket
  var apiErr *smithy.GenericAPIError
  if errors.As(err, &noBucket) {
   log.Printf("Bucket %s does not exist.\n", bucket)
   err = noBucket
  } else if errors.As(err, &apiErr) && apiErr.ErrorCode() ==
 "ObjectLockConfigurationNotFoundError" {
   log.Printf("Bucket %s does not have an object lock configuration.\n", bucket)
   err = nil
  }
 } else {
  lockConfig = output.ObjectLockConfiguration
 }

 return lockConfig, err
}
```

```
// GetObjectRetention retrieves the object retention configuration for an S3 object.
func (actor S3Actions) GetObjectRetention(ctx context.Context, bucket string, key
 string) (*types.ObjectLockRetention, error) {
 var retention *types.ObjectLockRetention
 input := &s3.GetObjectRetentionInput{
  Bucket: aws.String(bucket),
  Key:    aws.String(key),
 }

 output, err := actor.S3Client.GetObjectRetention(ctx, input)
 if err != nil {
  var noKey *types.NoSuchKey
  var apiErr *smithy.GenericAPIError
  if errors.As(err, &noKey) {
   log.Printf("Object %s does not exist in bucket %s.\n", key, bucket)
   err = noKey
  } else if errors.As(err, &apiErr) {
   switch apiErr.ErrorCode() {
   case "NoSuchObjectLockConfiguration":
    err = nil
   case "InvalidRequest":
    log.Printf("Bucket %s does not have locking enabled.", bucket)
    err = nil
   }
  }
 } else {
  retention = output.Retention
 }

 return retention, err
}



// PutObjectLegalHold sets the legal hold configuration for an S3 object.
func (actor S3Actions) PutObjectLegalHold(ctx context.Context, bucket string, key
 string, versionId string, legalHoldStatus types.ObjectLockLegalHoldStatus) error {
 input := &s3.PutObjectLegalHoldInput{
  Bucket: aws.String(bucket),
  Key:    aws.String(key),
  LegalHold: &types.ObjectLockLegalHold{
```

```
   Status: legalHoldStatus,
  },
 }
 if versionId != "" {
  input.VersionId = aws.String(versionId)
 }

 _, err := actor.S3Client.PutObjectLegalHold(ctx, input)
 if err != nil {
  var noKey *types.NoSuchKey
  if errors.As(err, &noKey) {
   log.Printf("Object %s does not exist in bucket %s.\n", key, bucket)
   err = noKey
  }
 }

 return err
}



// ModifyDefaultBucketRetention modifies the default retention period of an existing
 bucket.
func (actor S3Actions) ModifyDefaultBucketRetention(
 ctx context.Context, bucket string, lockMode types.ObjectLockEnabled,
 retentionPeriod int32, retentionMode types.ObjectLockRetentionMode) error {

 input := &s3.PutObjectLockConfigurationInput{
  Bucket: aws.String(bucket),
  ObjectLockConfiguration: &types.ObjectLockConfiguration{
   ObjectLockEnabled: lockMode,
   Rule: &types.ObjectLockRule{
    DefaultRetention: &types.DefaultRetention{
     Days: aws.Int32(retentionPeriod),
     Mode: retentionMode,
    },
   },
  },
 }
 _, err := actor.S3Client.PutObjectLockConfiguration(ctx, input)
 if err != nil {
  var noBucket *types.NoSuchBucket
  if errors.As(err, &noBucket) {
   log.Printf("Bucket %s does not exist.\n", bucket)
```

```
   err = noBucket
  }
 }

 return err
}



// EnableObjectLockOnBucket enables object locking on an existing bucket.
func (actor S3Actions) EnableObjectLockOnBucket(ctx context.Context, bucket string)
 error {
 // Versioning must be enabled on the bucket before object locking is enabled.
 verInput := &s3.PutBucketVersioningInput{
  Bucket: aws.String(bucket),
  VersioningConfiguration: &types.VersioningConfiguration{
   MFADelete: types.MFADeleteDisabled,
   Status:    types.BucketVersioningStatusEnabled,
  },
 }
 _, err := actor.S3Client.PutBucketVersioning(ctx, verInput)
 if err != nil {
  var noBucket *types.NoSuchBucket
  if errors.As(err, &noBucket) {
   log.Printf("Bucket %s does not exist.\n", bucket)
   err = noBucket
  }
  return err
 }

 input := &s3.PutObjectLockConfigurationInput{
  Bucket: aws.String(bucket),
  ObjectLockConfiguration: &types.ObjectLockConfiguration{
   ObjectLockEnabled: types.ObjectLockEnabledEnabled,
  },
 }
 _, err = actor.S3Client.PutObjectLockConfiguration(ctx, input)
 if err != nil {
  var noBucket *types.NoSuchBucket
  if errors.As(err, &noBucket) {
   log.Printf("Bucket %s does not exist.\n", bucket)
   err = noBucket
  }
 }
```

```
 return err
}



// PutObjectRetention sets the object retention configuration for an S3 object.
func (actor S3Actions) PutObjectRetention(ctx context.Context, bucket string, key
 string, retentionMode types.ObjectLockRetentionMode, retentionPeriodDays int32)
 error {
 input := &s3.PutObjectRetentionInput{
  Bucket: aws.String(bucket),
  Key:    aws.String(key),
  Retention: &types.ObjectLockRetention{
   Mode:            retentionMode,
   RetainUntilDate: aws.Time(time.Now().AddDate(0, 0, int(retentionPeriodDays))),
  },
  BypassGovernanceRetention: aws.Bool(true),
 }

 _, err := actor.S3Client.PutObjectRetention(ctx, input)
 if err != nil {
  var noKey *types.NoSuchKey
  if errors.As(err, &noKey) {
   log.Printf("Object %s does not exist in bucket %s.\n", key, bucket)
   err = noKey
  }
 }

 return err
}



// UploadObject uses the S3 upload manager to upload an object to a bucket.
func (actor S3Actions) UploadObject(ctx context.Context, bucket string, key string,
 contents string) (string, error) {
 var outKey string
 input := &s3.PutObjectInput{
  Bucket:            aws.String(bucket),
  Key:               aws.String(key),
  Body:              bytes.NewReader([]byte(contents)),
  ChecksumAlgorithm: types.ChecksumAlgorithmSha256,
 }
```

```
 output, err := actor.S3Manager.Upload(ctx, input)
 if err != nil {
  var noBucket *types.NoSuchBucket
  if errors.As(err, &noBucket) {
   log.Printf("Bucket %s does not exist.\n", bucket)
   err = noBucket
  }
 } else {
  err := s3.NewObjectExistsWaiter(actor.S3Client).Wait(ctx, &s3.HeadObjectInput{
   Bucket: aws.String(bucket),
   Key:    aws.String(key),
  }, time.Minute)
  if err != nil {
   log.Printf("Failed attempt to wait for object %s to exist in %s.\n", key, bucket)
  } else {
   outKey = *output.Key
  }
 }
 return outKey, err
}



// ListObjectVersions lists all versions of all objects in a bucket.
func (actor S3Actions) ListObjectVersions(ctx context.Context, bucket string)
 ([]types.ObjectVersion, error) {
 var err error
 var output *s3.ListObjectVersionsOutput
 var versions []types.ObjectVersion
 input := &s3.ListObjectVersionsInput{Bucket: aws.String(bucket)}
 versionPaginator := s3.NewListObjectVersionsPaginator(actor.S3Client, input)
 for versionPaginator.HasMorePages() {
  output, err = versionPaginator.NextPage(ctx)
  if err != nil {
   var noBucket *types.NoSuchBucket
   if errors.As(err, &noBucket) {
    log.Printf("Bucket %s does not exist.\n", bucket)
    err = noBucket
   }
   break
  } else {
   versions = append(versions, output.Versions...)
  }
 }
```

```
 return versions, err
}



// DeleteObject deletes an object from a bucket.
func (actor S3Actions) DeleteObject(ctx context.Context, bucket string, key string,
 versionId string, bypassGovernance bool) (bool, error) {
 deleted := false
 input := &s3.DeleteObjectInput{
  Bucket: aws.String(bucket),
  Key:    aws.String(key),
 }
 if versionId != "" {
  input.VersionId = aws.String(versionId)
 }
 if bypassGovernance {
  input.BypassGovernanceRetention = aws.Bool(true)
 }
 _, err := actor.S3Client.DeleteObject(ctx, input)
 if err != nil {
  var noKey *types.NoSuchKey
  var apiErr *smithy.GenericAPIError
  if errors.As(err, &noKey) {
   log.Printf("Object %s does not exist in %s.\n", key, bucket)
   err = noKey
  } else if errors.As(err, &apiErr) {
   switch apiErr.ErrorCode() {
   case "AccessDenied":
    log.Printf("Access denied: cannot delete object %s from %s.\n", key, bucket)
    err = nil
   case "InvalidArgument":
    if bypassGovernance {
     log.Printf("You cannot specify bypass governance on a bucket without lock
enabled.")
     err = nil
    }
   }
  }
 } else {
  err = s3.NewObjectNotExistsWaiter(actor.S3Client).Wait(
   ctx, &s3.HeadObjectInput{Bucket: aws.String(bucket), Key: aws.String(key)},
time.Minute)
  if err != nil {
```

```
   log.Printf("Failed attempt to wait for object %s in bucket %s to be deleted.\n",
 key, bucket)
  } else {
   deleted = true
  }
 }
 return deleted, err
}



// DeleteObjects deletes a list of objects from a bucket.
func (actor S3Actions) DeleteObjects(ctx context.Context, bucket string, objects
 []types.ObjectIdentifier, bypassGovernance bool) error {
 if len(objects) == 0 {
  return nil
 }

 input := s3.DeleteObjectsInput{
  Bucket: aws.String(bucket),
  Delete: &types.Delete{
   Objects: objects,
   Quiet:   aws.Bool(true),
  },
 }
 if bypassGovernance {
  input.BypassGovernanceRetention = aws.Bool(true)
 }
 delOut, err := actor.S3Client.DeleteObjects(ctx, &input)
 if err != nil || len(delOut.Errors) > 0 {
  log.Printf("Error deleting objects from bucket %s.\n", bucket)
  if err != nil {
   var noBucket *types.NoSuchBucket
   if errors.As(err, &noBucket) {
    log.Printf("Bucket %s does not exist.\n", bucket)
    err = noBucket
   }
  } else if len(delOut.Errors) > 0 {
   for _, outErr := range delOut.Errors {
    log.Printf("%s: %s\n", *outErr.Key, *outErr.Message)
   }
   err = fmt.Errorf("%s", *delOut.Errors[0].Message)
  }
 } else {
```

```
  for _, delObjs := range delOut.Deleted {
   err = s3.NewObjectNotExistsWaiter(actor.S3Client).Wait(
    ctx, &s3.HeadObjectInput{Bucket: aws.String(bucket), Key: delObjs.Key},
 time.Minute)
   if err != nil {
    log.Printf("Failed attempt to wait for object %s to be deleted.\n",
 *delObjs.Key)
   } else {
    log.Printf("Deleted %s.\n", *delObjs.Key)
   }
  }
 }
 return err
}
```

Bereinigen von Ressourcen.

```
import (
 "context"
 "log"
 "s3_object_lock/actions"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/s3"
 "github.com/aws/aws-sdk-go-v2/service/s3/types"
 "github.com/awsdocs/aws-doc-sdk-examples/gov2/demotools"
)

// DemoBucket contains metadata for buckets used in this example.
type DemoBucket struct {
 name             string
 retentionEnabled bool
 objectKeys       []string
}

// Resources keeps track of AWS resources created during the ObjectLockScenario and
 handles
// cleanup when the scenario finishes.
```

```
type Resources struct {
 demoBuckets map[string]*DemoBucket

 s3Actions  *actions.S3Actions
 questioner demotools.IQuestioner
}

// init initializes objects in the Resources struct.
func (resources *Resources) init(s3Actions *actions.S3Actions, questioner
 demotools.IQuestioner) {
 resources.s3Actions = s3Actions
 resources.questioner = questioner
 resources.demoBuckets = map[string]*DemoBucket{}
}

// Cleanup deletes all AWS resources created during the ObjectLockScenario.
func (resources *Resources) Cleanup(ctx context.Context) {
 defer func() {
  if r := recover(); r != nil {
   log.Printf("Something went wrong during cleanup.\n%v\n", r)
   log.Println("Use the AWS Management Console to remove any remaining resources " +
    "that were created for this scenario.")
  }
 }()

 wantDelete := resources.questioner.AskBool("Do you want to remove all of the AWS
 resources that were created "+
  "during this demo (y/n)?", "y")
 if !wantDelete {
  log.Println("Be sure to remove resources when you're done with them to avoid
 unexpected charges!")
  return
 }

 log.Println("Removing objects from S3 buckets and deleting buckets...")
 resources.deleteBuckets(ctx)
 //resources.deleteRetentionObjects(resources.retentionBucket,
 resources.retentionObjects)

 log.Println("Cleanup complete.")
}

// deleteBuckets empties and then deletes all buckets created during the
 ObjectLockScenario.
```

```
func (resources *Resources) deleteBuckets(ctx context.Context) {
 for _, info := range createInfo {
  bucket := resources.demoBuckets[info.name]
  resources.deleteObjects(ctx, bucket)
  _, err := resources.s3Actions.S3Client.DeleteBucket(ctx, &s3.DeleteBucketInput{
   Bucket: aws.String(bucket.name),
  })
  if err != nil {
   panic(err)
  }
 }
 for _, info := range createInfo {
  bucket := resources.demoBuckets[info.name]
  err := s3.NewBucketNotExistsWaiter(resources.s3Actions.S3Client).Wait(
   ctx, &s3.HeadBucketInput{Bucket: aws.String(bucket.name)}, time.Minute)
  if err != nil {
   log.Printf("Failed attempt to wait for bucket %s to be deleted.\n", bucket.name)
  } else {
   log.Printf("Deleted %s.\n", bucket.name)
  }
 }
 resources.demoBuckets = map[string]*DemoBucket{}
}

// deleteObjects deletes all objects in the specified bucket.
func (resources *Resources) deleteObjects(ctx context.Context, bucket *DemoBucket) {
 lockConfig, err := resources.s3Actions.GetObjectLockConfiguration(ctx, bucket.name)
 if err != nil {
  panic(err)
 }
 versions, err := resources.s3Actions.ListObjectVersions(ctx, bucket.name)
 if err != nil {
  switch err.(type) {
  case *types.NoSuchBucket:
   log.Printf("No objects to get from %s.\n", bucket.name)
  default:
   panic(err)
  }
 }
 delObjects := make([]types.ObjectIdentifier, len(versions))
 for i, version := range versions {
  if lockConfig != nil && lockConfig.ObjectLockEnabled ==
 types.ObjectLockEnabledEnabled {
```

```
    status, err := resources.s3Actions.GetObjectLegalHold(ctx, bucket.name,
 *version.Key, *version.VersionId)
    if err != nil {
     switch err.(type) {
     case *types.NoSuchKey:
      log.Printf("Couldn't determine legal hold status for %s in %s.\n",
 *version.Key, bucket.name)
     default:
      panic(err)
     }
    } else if status != nil && *status == types.ObjectLockLegalHoldStatusOn {
     err = resources.s3Actions.PutObjectLegalHold(ctx, bucket.name, *version.Key,
 *version.VersionId, types.ObjectLockLegalHoldStatusOff)
     if err != nil {
      switch err.(type) {
      case *types.NoSuchKey:
       log.Printf("Couldn't turn off legal hold for %s in %s.\n", *version.Key,
 bucket.name)
      default:
       panic(err)
      }
     }
    }
   }
   delObjects[i] = types.ObjectIdentifier{Key: version.Key, VersionId:
 version.VersionId}
 }
 err = resources.s3Actions.DeleteObjects(ctx, bucket.name, delObjects,
 bucket.retentionEnabled)
 if err != nil {
  switch err.(type) {
  case *types.NoSuchBucket:
   log.Println("Nothing to delete.")
  default:
   panic(err)
  }
 }
}
```

- API-Details finden Sie in den folgenden Themen der AWS SDK für Go -API-Referenz.
  - GetObjectLegalHold

- GetObjectLockConfiguration
- GetObjectRetention
- PutObjectLegalHold
- PutObjectLockConfiguration
- PutObjectRetention

Hoch- oder Herunterladen großer Dateien

Das folgende Codebeispiel zeigt, wie Sie große Dateien zu und von Amazon S3 hochladen oder herunterladen.

Weitere Informationen finden Sie unter Hochladen eines Objekts mit Multipart-Upload.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Erstellen Sie Funktionen, die Upload- und Download-Manager verwenden, um die Daten in Teile zu zerlegen und sie gleichzeitig zu übertragen.

```
import (
 "bytes"
 "context"
 "errors"
 "fmt"
 "io"
 "log"
 "os"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/s3/manager"
 "github.com/aws/aws-sdk-go-v2/service/s3"
 "github.com/aws/aws-sdk-go-v2/service/s3/types"
 "github.com/aws/smithy-go"
```

```
)

// BucketBasics encapsulates the Amazon Simple Storage Service (Amazon S3) actions
// used in the examples.
// It contains S3Client, an Amazon S3 service client that is used to perform bucket
// and object actions.
type BucketBasics struct {
 S3Client *s3.Client
}



// UploadLargeObject uses an upload manager to upload data to an object in a bucket.
// The upload manager breaks large data into parts and uploads the parts
 concurrently.
func (basics BucketBasics) UploadLargeObject(ctx context.Context, bucketName string,
 objectKey string, largeObject []byte) error {
 largeBuffer := bytes.NewReader(largeObject)
 var partMiBs int64 = 10
 uploader := manager.NewUploader(basics.S3Client, func(u *manager.Uploader) {
  u.PartSize = partMiBs * 1024 * 1024
 })
 _, err := uploader.Upload(ctx, &s3.PutObjectInput{
  Bucket: aws.String(bucketName),
  Key:    aws.String(objectKey),
  Body:   largeBuffer,
 })
 if err != nil {
  var apiErr smithy.APIError
  if errors.As(err, &apiErr) && apiErr.ErrorCode() == "EntityTooLarge" {
   log.Printf("Error while uploading object to %s. The object is too large.\n"+
    "The maximum size for a multipart upload is 5TB.", bucketName)
  } else {
   log.Printf("Couldn't upload large object to %v:%v. Here's why: %v\n",
    bucketName, objectKey, err)
  }
 } else {
  err = s3.NewObjectExistsWaiter(basics.S3Client).Wait(
   ctx, &s3.HeadObjectInput{Bucket: aws.String(bucketName), Key:
 aws.String(objectKey)}, time.Minute)
  if err != nil {
   log.Printf("Failed attempt to wait for object %s to exist.\n", objectKey)
  }
 }
```

```
 return err
}



// DownloadLargeObject uses a download manager to download an object from a bucket.
// The download manager gets the data in parts and writes them to a buffer until all
 of
// the data has been downloaded.
func (basics BucketBasics) DownloadLargeObject(ctx context.Context, bucketName
 string, objectKey string) ([]byte, error) {
 var partMiBs int64 = 10
 downloader := manager.NewDownloader(basics.S3Client, func(d *manager.Downloader) {
  d.PartSize = partMiBs * 1024 * 1024
 })
 buffer := manager.NewWriteAtBuffer([]byte{})
 _, err := downloader.Download(ctx, buffer, &s3.GetObjectInput{
  Bucket: aws.String(bucketName),
  Key:    aws.String(objectKey),
 })
 if err != nil {
  log.Printf("Couldn't download large object from %v:%v. Here's why: %v\n",
   bucketName, objectKey, err)
 }
 return buffer.Bytes(), err
}
```

Führen Sie ein interaktives Szenario aus, das Ihnen zeigt, wie Sie die Upload- und Download-Manager im Kontext verwenden.

```
import (
 "context"
 "crypto/rand"
 "log"
 "strings"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/s3"
 "github.com/awsdocs/aws-doc-sdk-examples/gov2/demotools"
```

```
 "github.com/awsdocs/aws-doc-sdk-examples/gov2/s3/actions"
)

// RunLargeObjectScenario is an interactive example that shows you how to use Amazon
// Simple Storage Service (Amazon S3) to upload and download large objects.
//
// 1. Create a bucket.
// 3. Upload a large object to the bucket by using an upload manager.
// 5. Download a large object by using a download manager.
// 8. Delete all objects in the bucket.
// 9. Delete the bucket.
//
// This example creates an Amazon S3 service client from the specified sdkConfig so
 that
// you can replace it with a mocked or stubbed config for unit testing.
//
// It uses a questioner from the `demotools` package to get input during the
 example.
// This package can be found in the ..\..\demotools folder of this repo.
func RunLargeObjectScenario(ctx context.Context, sdkConfig aws.Config, questioner
 demotools.IQuestioner) {
 defer func() {
  if r := recover(); r != nil {
   log.Println("Something went wrong with the demo.")
   _, isMock := questioner.(*demotools.MockQuestioner)
   if isMock || questioner.AskBool("Do you want to see the full error message (y/
n)?", "y") {
    log.Println(r)
   }
  }
 }()

 log.Println(strings.Repeat("-", 88))
 log.Println("Welcome to the Amazon S3 large object demo.")
 log.Println(strings.Repeat("-", 88))

 s3Client := s3.NewFromConfig(sdkConfig)
 bucketBasics := actions.BucketBasics{S3Client: s3Client}

 bucketName := questioner.Ask("Let's create a bucket. Enter a name for your
 bucket:",
  demotools.NotEmpty{})
 bucketExists, err := bucketBasics.BucketExists(ctx, bucketName)
 if err != nil {
```

```
 panic(err)
}
if !bucketExists {
 err = bucketBasics.CreateBucket(ctx, bucketName, sdkConfig.Region)
 if err != nil {
  panic(err)
 } else {
  log.Println("Bucket created.")
 }
}
log.Println(strings.Repeat("-", 88))

mibs := 30
log.Printf("Let's create a slice of %v MiB of random bytes and upload it to your
bucket. ", mibs)
questioner.Ask("Press Enter when you're ready.")
largeBytes := make([]byte, 1024*1024*mibs)
_, _ = rand.Read(largeBytes)
largeKey := "doc-example-large"
log.Println("Uploading...")
err = bucketBasics.UploadLargeObject(ctx, bucketName, largeKey, largeBytes)
if err != nil {
 panic(err)
}
log.Printf("Uploaded %v MiB object as %v", mibs, largeKey)
log.Println(strings.Repeat("-", 88))

log.Printf("Let's download the %v MiB object.", mibs)
questioner.Ask("Press Enter when you're ready.")
log.Println("Downloading...")
largeDownload, err := bucketBasics.DownloadLargeObject(ctx, bucketName, largeKey)
if err != nil {
 panic(err)
}
log.Printf("Downloaded %v bytes.", len(largeDownload))
log.Println(strings.Repeat("-", 88))

if questioner.AskBool("Do you want to delete your bucket and all of its "+
 "contents? (y/n)", "y") {
 log.Println("Deleting object.")
 err = bucketBasics.DeleteObjects(ctx, bucketName, []string{largeKey})
 if err != nil {
  panic(err)
 }
```

```
  log.Println("Deleting bucket.")
  err = bucketBasics.DeleteBucket(ctx, bucketName)
  if err != nil {
   panic(err)
  }
 } else {
  log.Println("Okay. Don't forget to delete objects from your bucket to avoid
 charges.")
 }
 log.Println(strings.Repeat("-", 88))

 log.Println("Thanks for watching!")
 log.Println(strings.Repeat("-", 88))
}
```

Serverless-Beispiele

Aufrufen einer Lambda-Funktion über einen Amazon-S3-Auslöser

Das folgende Codebeispiel zeigt, wie eine Lambda-Funktion implementiert wird, die ein Ereignis empfängt, das durch das Hochladen eines Objekts in einen S3-Bucket ausgelöst wird. Die Funktion ruft den Namen des S3-Buckets sowie den Objektschlüssel aus dem Ereignisparameter ab und ruft die Amazon-S3-API auf, um den Inhaltstyp des Objekts abzurufen und zu protokollieren.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu. GitHub Das vollständige Beispiel sowie eine Anleitung zum Einrichten und Ausführen finden Sie im Repository mit Serverless-Beispielen.

Nutzen eines S3-Ereignisses mit Lambda unter Verwendung von Go

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
package main

import (
```

```
 "context"
 "log"

 "github.com/aws/aws-lambda-go/events"
 "github.com/aws/aws-lambda-go/lambda"
 "github.com/aws/aws-sdk-go-v2/config"
 "github.com/aws/aws-sdk-go-v2/service/s3"
)

func handler(ctx context.Context, s3Event events.S3Event) error {
 sdkConfig, err := config.LoadDefaultConfig(ctx)
 if err != nil {
  log.Printf("failed to load default config: %s", err)
  return err
 }
 s3Client := s3.NewFromConfig(sdkConfig)

 for _, record := range s3Event.Records {
  bucket := record.S3.Bucket.Name
  key := record.S3.Object.URLDecodedKey
  headOutput, err := s3Client.HeadObject(ctx, &s3.HeadObjectInput{
   Bucket: &bucket,
   Key:    &key,
  })
  if err != nil {
   log.Printf("error getting head of object %s/%s: %s", bucket, key, err)
   return err
  }
  log.Printf("successfully retrieved %s/%s of type %s", bucket, key,
 *headOutput.ContentType)
 }

 return nil
}

func main() {
 lambda.Start(handler)
}
```

Amazon SNS SNS-Beispiele mit SDK for Go V2

Die folgenden Codebeispiele zeigen Ihnen, wie Sie mithilfe von AWS SDK für Go V2 mit Amazon SNS Aktionen ausführen und allgemeine Szenarien implementieren.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Service-Funktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarios anzeigen.

Szenarien sind Code-Beispiele, die Ihnen zeigen, wie Sie bestimmte Aufgaben ausführen, indem Sie mehrere Funktionen innerhalb eines Services aufrufen oder mit anderen AWS-Services kombinieren.

Jedes Beispiel enthält einen Link zum vollständigen Quellcode, in dem Sie Anweisungen zur Einrichtung und Ausführung des Codes im Kontext finden.

Erste Schritte

Hello Amazon SNS

Die folgenden Codebeispiele veranschaulichen die ersten Schritte mit Amazon SNS.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
package main

import (
 "context"
 "fmt"
 "log"

 "github.com/aws/aws-sdk-go-v2/config"
 "github.com/aws/aws-sdk-go-v2/service/sns"
 "github.com/aws/aws-sdk-go-v2/service/sns/types"
)
```

```
// main uses the AWS SDK for Go V2 to create an Amazon Simple Notification Service
// (Amazon SNS) client and list the topics in your account.
// This example uses the default settings specified in your shared credentials
// and config files.
func main() {
 ctx := context.Background()
 sdkConfig, err := config.LoadDefaultConfig(ctx)
 if err != nil {
  fmt.Println("Couldn't load default configuration. Have you set up your AWS
 account?")
  fmt.Println(err)
  return
 }
 snsClient := sns.NewFromConfig(sdkConfig)
 fmt.Println("Let's list the topics for your account.")
 var topics []types.Topic
 paginator := sns.NewListTopicsPaginator(snsClient, &sns.ListTopicsInput{})
 for paginator.HasMorePages() {
  output, err := paginator.NextPage(ctx)
  if err != nil {
   log.Printf("Couldn't get topics. Here's why: %v\n", err)
   break
  } else {
   topics = append(topics, output.Topics...)
  }
 }
 if len(topics) == 0 {
  fmt.Println("You don't have any topics!")
 } else {
  for _, topic := range topics {
   fmt.Printf("\t%v\n", *topic.TopicArn)
  }
 }
}
```

- Einzelheiten zur API finden Sie ListTopicsin der AWS SDK für Go API-Referenz.

Themen

- Aktionen
- Szenarien

- Serverless-Beispiele

Aktionen

`CreateTopic`

Das folgende Codebeispiel zeigt die Verwendung`CreateTopic`.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import (
 "context"
 "encoding/json"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/sns"
 "github.com/aws/aws-sdk-go-v2/service/sns/types"
)

// SnsActions encapsulates the Amazon Simple Notification Service (Amazon SNS)
 actions
// used in the examples.
type SnsActions struct {
 SnsClient *sns.Client
}



// CreateTopic creates an Amazon SNS topic with the specified name. You can
 optionally
// specify that the topic is created as a FIFO topic and whether it uses content-
based
// deduplication instead of ID-based deduplication.
```

```
func (actor SnsActions) CreateTopic(ctx context.Context, topicName string,
 isFifoTopic bool, contentBasedDeduplication bool) (string, error) {
 var topicArn string
 topicAttributes := map[string]string{}
 if isFifoTopic {
  topicAttributes["FifoTopic"] = "true"
 }
 if contentBasedDeduplication {
  topicAttributes["ContentBasedDeduplication"] = "true"
 }
 topic, err := actor.SnsClient.CreateTopic(ctx, &sns.CreateTopicInput{
  Name:       aws.String(topicName),
  Attributes: topicAttributes,
 })
 if err != nil {
  log.Printf("Couldn't create topic %v. Here's why: %v\n", topicName, err)
 } else {
  topicArn = *topic.TopicArn
 }

 return topicArn, err
}
```

- Einzelheiten zur API finden Sie CreateTopicin der AWS SDK für Go API-Referenz.

`DeleteTopic`

Das folgende Codebeispiel zeigt die Verwendung`DeleteTopic`.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import (
 "context"
```

```
 "encoding/json"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/sns"
 "github.com/aws/aws-sdk-go-v2/service/sns/types"
)

// SnsActions encapsulates the Amazon Simple Notification Service (Amazon SNS)
 actions
// used in the examples.
type SnsActions struct {
 SnsClient *sns.Client
}



// DeleteTopic delete an Amazon SNS topic.
func (actor SnsActions) DeleteTopic(ctx context.Context, topicArn string) error {
 _, err := actor.SnsClient.DeleteTopic(ctx, &sns.DeleteTopicInput{
  TopicArn: aws.String(topicArn)})
 if err != nil {
  log.Printf("Couldn't delete topic %v. Here's why: %v\n", topicArn, err)
 }
 return err
}
```

- Einzelheiten zur API finden Sie DeleteTopicin der AWS SDK für Go API-Referenz.

`ListTopics`

Das folgende Codebeispiel zeigt die Verwendung`ListTopics`.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
package main

import (
 "context"
 "fmt"
 "log"

 "github.com/aws/aws-sdk-go-v2/config"
 "github.com/aws/aws-sdk-go-v2/service/sns"
 "github.com/aws/aws-sdk-go-v2/service/sns/types"
)

// main uses the AWS SDK for Go V2 to create an Amazon Simple Notification Service
// (Amazon SNS) client and list the topics in your account.
// This example uses the default settings specified in your shared credentials
// and config files.
func main() {
 ctx := context.Background()
 sdkConfig, err := config.LoadDefaultConfig(ctx)
 if err != nil {
  fmt.Println("Couldn't load default configuration. Have you set up your AWS
 account?")
  fmt.Println(err)
  return
 }
 snsClient := sns.NewFromConfig(sdkConfig)
 fmt.Println("Let's list the topics for your account.")
 var topics []types.Topic
 paginator := sns.NewListTopicsPaginator(snsClient, &sns.ListTopicsInput{})
 for paginator.HasMorePages() {
  output, err := paginator.NextPage(ctx)
  if err != nil {
   log.Printf("Couldn't get topics. Here's why: %v\n", err)
   break
  } else {
   topics = append(topics, output.Topics...)
  }
 }
 if len(topics) == 0 {
  fmt.Println("You don't have any topics!")
 } else {
  for _, topic := range topics {
```

```
    fmt.Printf("\t%v\n", *topic.TopicArn)
   }
  }
}
```

- Einzelheiten zur API finden Sie ListTopicsin der AWS SDK für Go API-Referenz.

`Publish`

Das folgende Codebeispiel zeigt die Verwendung`Publish`.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import (
 "context"
 "encoding/json"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/sns"
 "github.com/aws/aws-sdk-go-v2/service/sns/types"
)

// SnsActions encapsulates the Amazon Simple Notification Service (Amazon SNS)
 actions
// used in the examples.
type SnsActions struct {
 SnsClient *sns.Client
}
```

```
// Publish publishes a message to an Amazon SNS topic. The message is then sent to
 all
// subscribers. When the topic is a FIFO topic, the message must also contain a
 group ID
// and, when ID-based deduplication is used, a deduplication ID. An optional key-
value
// filter attribute can be specified so that the message can be filtered according
 to
// a filter policy.
func (actor SnsActions) Publish(ctx context.Context, topicArn string, message
 string, groupId string, dedupId string, filterKey string, filterValue string) error
 {
 publishInput := sns.PublishInput{TopicArn: aws.String(topicArn), Message:
 aws.String(message)}
 if groupId != "" {
  publishInput.MessageGroupId = aws.String(groupId)
 }
 if dedupId != "" {
  publishInput.MessageDeduplicationId = aws.String(dedupId)
 }
 if filterKey != "" && filterValue != "" {
  publishInput.MessageAttributes = map[string]types.MessageAttributeValue{
   filterKey: {DataType: aws.String("String"), StringValue:
 aws.String(filterValue)},
  }
 }
 _, err := actor.SnsClient.Publish(ctx, &publishInput)
 if err != nil {
  log.Printf("Couldn't publish message to topic %v. Here's why: %v", topicArn, err)
 }
 return err
}
```

- Details zu API finden Sie unter Veröffentlichen in der AWS SDK für Go -API-Referenz.

`Subscribe`

Das folgende Codebeispiel zeigt, wie man es benutzt`Subscribe`.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Abonnieren Sie eine Warteschlange für ein Thema mit optionalen Filtern.

```
import (
 "context"
 "encoding/json"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/sns"
 "github.com/aws/aws-sdk-go-v2/service/sns/types"
)

// SnsActions encapsulates the Amazon Simple Notification Service (Amazon SNS)
 actions
// used in the examples.
type SnsActions struct {
 SnsClient *sns.Client
}



// SubscribeQueue subscribes an Amazon Simple Queue Service (Amazon SQS) queue to an
// Amazon SNS topic. When filterMap is not nil, it is used to specify a filter
 policy
// so that messages are only sent to the queue when the message has the specified
 attributes.
func (actor SnsActions) SubscribeQueue(ctx context.Context, topicArn string,
 queueArn string, filterMap map[string][]string) (string, error) {
 var subscriptionArn string
 var attributes map[string]string
 if filterMap != nil {
  filterBytes, err := json.Marshal(filterMap)
  if err != nil {
   log.Printf("Couldn't create filter policy, here's why: %v\n", err)
```

```
   return "", err
  }
  attributes = map[string]string{"FilterPolicy": string(filterBytes)}
 }
 output, err := actor.SnsClient.Subscribe(ctx, &sns.SubscribeInput{
  Protocol:              aws.String("sqs"),
  TopicArn:              aws.String(topicArn),
  Attributes:            attributes,
  Endpoint:              aws.String(queueArn),
  ReturnSubscriptionArn: true,
 })
 if err != nil {
  log.Printf("Couldn't susbscribe queue %v to topic %v. Here's why: %v\n",
   queueArn, topicArn, err)
 } else {
  subscriptionArn = *output.SubscriptionArn
 }

 return subscriptionArn, err
}
```

- Details zu API finden Sie unter Abonnieren in der AWS SDK für Go -API-Referenz.

Szenarien

Veröffentlichen Sie Nachrichten in Warteschlangen

Wie das aussehen kann, sehen Sie am nachfolgenden Beispielcode:

- Erstellen Sie ein Thema (FIFO oder Nicht-FIFO).
- Abonnieren Sie mehrere Warteschlangen für das Thema mit der Option, einen Filter anzuwenden.
- Veröffentlichen Sie eine Nachricht im Thema.
- Fragen Sie die Warteschlangen nach empfangenen Nachrichten ab.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Führen Sie ein interaktives Szenario an einem Prompt aus.

```
import (
 "context"
 "encoding/json"
 "fmt"
 "log"
 "strings"
 "topics_and_queues/actions"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/sns"
 "github.com/aws/aws-sdk-go-v2/service/sqs"
 "github.com/aws/aws-sdk-go-v2/service/sqs/types"
 "github.com/awsdocs/aws-doc-sdk-examples/gov2/demotools"
)

const FIFO_SUFFIX = ".fifo"
const TONE_KEY = "tone"

var ToneChoices = []string{"cheerful", "funny", "serious", "sincere"}

// MessageBody is used to deserialize the body of a message from a JSON string.
type MessageBody struct {
 Message string
}

// ScenarioRunner separates the steps of this scenario into individual functions so
 that
// they are simpler to read and understand.
type ScenarioRunner struct {
 questioner demotools.IQuestioner
 snsActor   *actions.SnsActions
 sqsActor   *actions.SqsActions
```

```
}

func (runner ScenarioRunner) CreateTopic(ctx context.Context) (string, string, bool,
 bool) {
 log.Println("SNS topics can be configured as FIFO (First-In-First-Out) or standard.
\n" +
  "FIFO topics deliver messages in order and support deduplication and message
 filtering.")
 isFifoTopic := runner.questioner.AskBool("\nWould you like to work with FIFO
 topics? (y/n) ", "y")

 contentBasedDeduplication := false
 if isFifoTopic {
  log.Println(strings.Repeat("-", 88))
  log.Println("Because you have chosen a FIFO topic, deduplication is supported.\n"
 +
   "Deduplication IDs are either set in the message or are automatically generated
\n" +
   "from content using a hash function. If a message is successfully published to\n"
 +
   "an SNS FIFO topic, any message published and determined to have the same\n" +
   "deduplication ID, within the five-minute deduplication interval, is accepted\n"
 +
   "but not delivered. For more information about deduplication, see:\n" +
   "\thttps://docs.aws.amazon.com/sns/latest/dg/fifo-message-dedup.html.")
  contentBasedDeduplication = runner.questioner.AskBool(
   "\nDo you want to use content-based deduplication instead of entering a
 deduplication ID? (y/n) ", "y")
 }
 log.Println(strings.Repeat("-", 88))

 topicName := runner.questioner.Ask("Enter a name for your SNS topic. ")
 if isFifoTopic {
  topicName = fmt.Sprintf("%v%v", topicName, FIFO_SUFFIX)
  log.Printf("Because you have selected a FIFO topic, '%v' must be appended to\n"+
   "the topic name.", FIFO_SUFFIX)
 }

 topicArn, err := runner.snsActor.CreateTopic(ctx, topicName, isFifoTopic,
 contentBasedDeduplication)
 if err != nil {
  panic(err)
 }
 log.Printf("Your new topic with the name '%v' and Amazon Resource Name (ARN) \n"+
```

```
  "'%v' has been created.", topicName, topicArn)

 return topicName, topicArn, isFifoTopic, contentBasedDeduplication
}

func (runner ScenarioRunner) CreateQueue(ctx context.Context, ordinal string,
 isFifoTopic bool) (string, string) {
 queueName := runner.questioner.Ask(fmt.Sprintf("Enter a name for the %v SQS queue.
 ", ordinal))
 if isFifoTopic {
  queueName = fmt.Sprintf("%v%v", queueName, FIFO_SUFFIX)
  if ordinal == "first" {
   log.Printf("Because you are creating a FIFO SQS queue, '%v' must "+
    "be appended to the queue name.\n", FIFO_SUFFIX)
  }
 }
 queueUrl, err := runner.sqsActor.CreateQueue(ctx, queueName, isFifoTopic)
 if err != nil {
  panic(err)
 }
 log.Printf("Your new SQS queue with the name '%v' and the queue URL "+
  "'%v' has been created.", queueName, queueUrl)

 return queueName, queueUrl
}

func (runner ScenarioRunner) SubscribeQueueToTopic(
 ctx context.Context, queueName string, queueUrl string, topicName string, topicArn
 string, ordinal string,
 isFifoTopic bool) (string, bool) {

 queueArn, err := runner.sqsActor.GetQueueArn(ctx, queueUrl)
 if err != nil {
  panic(err)
 }
 log.Printf("The ARN of your queue is: %v.\n", queueArn)

 err = runner.sqsActor.AttachSendMessagePolicy(ctx, queueUrl, queueArn, topicArn)
 if err != nil {
  panic(err)
 }
 log.Println("Attached an IAM policy to the queue so the SNS topic can send " +
  "messages to it.")
 log.Println(strings.Repeat("-", 88))
```

```
var filterPolicy map[string][]string
if isFifoTopic {
 if ordinal == "first" {
  log.Println("Subscriptions to a FIFO topic can have filters.\n" +
   "If you add a filter to this subscription, then only the filtered messages\n" +
   "will be received in the queue.\n" +
   "For information about message filtering, see\n" +
   "\thttps://docs.aws.amazon.com/sns/latest/dg/sns-message-filtering.html\n" +
   "For this example, you can filter messages by a \"tone\" attribute.")
 }

 wantFiltering := runner.questioner.AskBool(
  fmt.Sprintf("Do you want to filter messages that are sent to \"%v\"\n"+
   "from the %v topic? (y/n) ", queueName, topicName), "y")
 if wantFiltering {
  log.Println("You can filter messages by one or more of the following \"tone\"
attributes.")

  var toneSelections []string
  askAboutTones := true
  for askAboutTones {
   toneIndex := runner.questioner.AskChoice(
    "Enter the number of the tone you want to filter by:\n", ToneChoices)
   toneSelections = append(toneSelections, ToneChoices[toneIndex])
   askAboutTones = runner.questioner.AskBool("Do you want to add another tone to
the filter? (y/n) ", "y")
  }
  log.Printf("Your subscription will be filtered to only pass the following tones:
%v\n", toneSelections)
  filterPolicy = map[string][]string{TONE_KEY: toneSelections}
 }
}

subscriptionArn, err := runner.snsActor.SubscribeQueue(ctx, topicArn, queueArn,
filterPolicy)
if err != nil {
 panic(err)
}
log.Printf("The queue %v is now subscribed to the topic %v with the subscription
ARN %v.\n",
 queueName, topicName, subscriptionArn)

return subscriptionArn, filterPolicy != nil
```

```
}

func (runner ScenarioRunner) PublishMessages(ctx context.Context, topicArn string,
 isFifoTopic bool, contentBasedDeduplication bool, usingFilters bool) {
 var message string
 var groupId string
 var dedupId string
 var toneSelection string
 publishMore := true
 for publishMore {
  groupId = ""
  dedupId = ""
  toneSelection = ""
  message = runner.questioner.Ask("Enter a message to publish: ")
  if isFifoTopic {
   log.Println("Because you are using a FIFO topic, you must set a message group ID.
\n" +
    "All messages within the same group will be received in the order they were
 published.")
   groupId = runner.questioner.Ask("Enter a message group ID: ")
   if !contentBasedDeduplication {
    log.Println("Because you are not using content-based deduplication,\n" +
     "you must enter a deduplication ID.")
    dedupId = runner.questioner.Ask("Enter a deduplication ID: ")
   }
  }
  if usingFilters {
   if runner.questioner.AskBool("Add a tone attribute so this message can be
 filtered? (y/n) ", "y") {
    toneIndex := runner.questioner.AskChoice(
     "Enter the number of the tone you want to filter by:\n", ToneChoices)
    toneSelection = ToneChoices[toneIndex]
   }
  }

  err := runner.snsActor.Publish(ctx, topicArn, message, groupId, dedupId, TONE_KEY,
 toneSelection)
  if err != nil {
   panic(err)
  }
  log.Println(("Your message was published."))

  publishMore = runner.questioner.AskBool("Do you want to publish another messsage?
 (y/n) ", "y")
```

```
 }
}

func (runner ScenarioRunner) PollForMessages(ctx context.Context, queueUrls
 []string) {
 log.Println("Polling queues for messages...")
 for _, queueUrl := range queueUrls {
  var messages []types.Message
  for {
   currentMsgs, err := runner.sqsActor.GetMessages(ctx, queueUrl, 10, 1)
   if err != nil {
    panic(err)
   }
   if len(currentMsgs) == 0 {
    break
   }
   messages = append(messages, currentMsgs...)
  }
  if len(messages) == 0 {
   log.Printf("No messages were received by queue %v.\n", queueUrl)
  } else if len(messages) == 1 {
   log.Printf("One message was received by queue %v:\n", queueUrl)

  } else {
   log.Printf("%v messages were received by queue %v:\n", len(messages), queueUrl)
  }
  for msgIndex, message := range messages {
   messageBody := MessageBody{}
   err := json.Unmarshal([]byte(*message.Body), &messageBody)
   if err != nil {
    panic(err)
   }
   log.Printf("Message %v: %v\n", msgIndex+1, messageBody.Message)
  }

  if len(messages) > 0 {
   log.Printf("Deleting %v messages from queue %v.\n", len(messages), queueUrl)
   err := runner.sqsActor.DeleteMessages(ctx, queueUrl, messages)
   if err != nil {
    panic(err)
   }
  }
 }
}
```

```
// RunTopicsAndQueuesScenario is an interactive example that shows you how to use
 the
// AWS SDK for Go to create and use Amazon SNS topics and Amazon SQS queues.
//
// 1. Create a topic (FIFO or non-FIFO).
// 2. Subscribe several queues to the topic with an option to apply a filter.
// 3. Publish messages to the topic.
// 4. Poll the queues for messages received.
// 5. Delete the topic and the queues.
//
// This example creates service clients from the specified sdkConfig so that
// you can replace it with a mocked or stubbed config for unit testing.
//
// It uses a questioner from the `demotools` package to get input during the
 example.
// This package can be found in the ..\..\demotools folder of this repo.
func RunTopicsAndQueuesScenario(
 ctx context.Context, sdkConfig aws.Config, questioner demotools.IQuestioner) {
 resources := Resources{}
 defer func() {
  if r := recover(); r != nil {
   log.Println("Something went wrong with the demo.\n" +
    "Cleaning up any resources that were created...")
   resources.Cleanup(ctx)
  }
 }()
 queueCount := 2

 log.Println(strings.Repeat("-", 88))
 log.Printf("Welcome to messaging with topics and queues.\n\n"+
  "In this scenario, you will create an SNS topic and subscribe %v SQS queues to the
\n"+
  "topic. You can select from several options for configuring the topic and the\n"+
  "subscriptions for the queues. You can then post to the topic and see the results
\n"+
  "in the queues.\n", queueCount)

 log.Println(strings.Repeat("-", 88))

 runner := ScenarioRunner{
  questioner: questioner,
  snsActor:   &actions.SnsActions{SnsClient: sns.NewFromConfig(sdkConfig)},
  sqsActor:   &actions.SqsActions{SqsClient: sqs.NewFromConfig(sdkConfig)},
```

```
 }
 resources.snsActor = runner.snsActor
 resources.sqsActor = runner.sqsActor

 topicName, topicArn, isFifoTopic, contentBasedDeduplication :=
 runner.CreateTopic(ctx)
 resources.topicArn = topicArn
 log.Println(strings.Repeat("-", 88))

 log.Printf("Now you will create %v SQS queues and subscribe them to the topic.\n",
 queueCount)
 ordinals := []string{"first", "next"}
 usingFilters := false
 for _, ordinal := range ordinals {
  queueName, queueUrl := runner.CreateQueue(ctx, ordinal, isFifoTopic)
  resources.queueUrls = append(resources.queueUrls, queueUrl)

  _, filtering := runner.SubscribeQueueToTopic(ctx, queueName, queueUrl, topicName,
 topicArn, ordinal, isFifoTopic)
  usingFilters = usingFilters || filtering
 }

 log.Println(strings.Repeat("-", 88))
 runner.PublishMessages(ctx, topicArn, isFifoTopic, contentBasedDeduplication,
 usingFilters)
 log.Println(strings.Repeat("-", 88))
 runner.PollForMessages(ctx, resources.queueUrls)

 log.Println(strings.Repeat("-", 88))

 wantCleanup := questioner.AskBool("Do you want to remove all AWS resources created
 for this scenario? (y/n) ", "y")
 if wantCleanup {
  log.Println("Cleaning up resources...")
  resources.Cleanup(ctx)
 }

 log.Println(strings.Repeat("-", 88))
 log.Println("Thanks for watching!")
 log.Println(strings.Repeat("-", 88))
}
```

Definieren Sie eine Struktur, die die in diesem Beispiel verwendeten Amazon SNS SNS-Aktionen umschließt.

```
import (
 "context"
 "encoding/json"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/sns"
 "github.com/aws/aws-sdk-go-v2/service/sns/types"
)

// SnsActions encapsulates the Amazon Simple Notification Service (Amazon SNS)
 actions
// used in the examples.
type SnsActions struct {
 SnsClient *sns.Client
}



// CreateTopic creates an Amazon SNS topic with the specified name. You can
 optionally
// specify that the topic is created as a FIFO topic and whether it uses content-
based
// deduplication instead of ID-based deduplication.
func (actor SnsActions) CreateTopic(ctx context.Context, topicName string,
 isFifoTopic bool, contentBasedDeduplication bool) (string, error) {
 var topicArn string
 topicAttributes := map[string]string{}
 if isFifoTopic {
  topicAttributes["FifoTopic"] = "true"
 }
 if contentBasedDeduplication {
  topicAttributes["ContentBasedDeduplication"] = "true"
 }
 topic, err := actor.SnsClient.CreateTopic(ctx, &sns.CreateTopicInput{
  Name:       aws.String(topicName),
  Attributes: topicAttributes,
 })
 if err != nil {
```

```
  log.Printf("Couldn't create topic %v. Here's why: %v\n", topicName, err)
 } else {
  topicArn = *topic.TopicArn
 }

 return topicArn, err
}



// DeleteTopic delete an Amazon SNS topic.
func (actor SnsActions) DeleteTopic(ctx context.Context, topicArn string) error {
 _, err := actor.SnsClient.DeleteTopic(ctx, &sns.DeleteTopicInput{
  TopicArn: aws.String(topicArn)})
 if err != nil {
  log.Printf("Couldn't delete topic %v. Here's why: %v\n", topicArn, err)
 }
 return err
}



// SubscribeQueue subscribes an Amazon Simple Queue Service (Amazon SQS) queue to an
// Amazon SNS topic. When filterMap is not nil, it is used to specify a filter
 policy
// so that messages are only sent to the queue when the message has the specified
 attributes.
func (actor SnsActions) SubscribeQueue(ctx context.Context, topicArn string,
 queueArn string, filterMap map[string][]string) (string, error) {
 var subscriptionArn string
 var attributes map[string]string
 if filterMap != nil {
  filterBytes, err := json.Marshal(filterMap)
  if err != nil {
   log.Printf("Couldn't create filter policy, here's why: %v\n", err)
   return "", err
  }
  attributes = map[string]string{"FilterPolicy": string(filterBytes)}
 }
 output, err := actor.SnsClient.Subscribe(ctx, &sns.SubscribeInput{
  Protocol:              aws.String("sqs"),
  TopicArn:              aws.String(topicArn),
  Attributes:            attributes,
  Endpoint:              aws.String(queueArn),
```

```
  ReturnSubscriptionArn: true,
 })
 if err != nil {
  log.Printf("Couldn't susbscribe queue %v to topic %v. Here's why: %v\n",
   queueArn, topicArn, err)
 } else {
  subscriptionArn = *output.SubscriptionArn
 }

 return subscriptionArn, err
}



// Publish publishes a message to an Amazon SNS topic. The message is then sent to
 all
// subscribers. When the topic is a FIFO topic, the message must also contain a
 group ID
// and, when ID-based deduplication is used, a deduplication ID. An optional key-
value
// filter attribute can be specified so that the message can be filtered according
 to
// a filter policy.
func (actor SnsActions) Publish(ctx context.Context, topicArn string, message
 string, groupId string, dedupId string, filterKey string, filterValue string) error
 {
 publishInput := sns.PublishInput{TopicArn: aws.String(topicArn), Message:
 aws.String(message)}
 if groupId != "" {
  publishInput.MessageGroupId = aws.String(groupId)
 }
 if dedupId != "" {
  publishInput.MessageDeduplicationId = aws.String(dedupId)
 }
 if filterKey != "" && filterValue != "" {
  publishInput.MessageAttributes = map[string]types.MessageAttributeValue{
   filterKey: {DataType: aws.String("String"), StringValue:
 aws.String(filterValue)},
  }
 }
 _, err := actor.SnsClient.Publish(ctx, &publishInput)
 if err != nil {
  log.Printf("Couldn't publish message to topic %v. Here's why: %v", topicArn, err)
 }
```

```
 return err
}
```

Definieren Sie eine Struktur, die die in diesem Beispiel verwendeten Amazon SQS SQS-Aktionen umschließt.

```
import (
 "context"
 "encoding/json"
 "fmt"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/sqs"
 "github.com/aws/aws-sdk-go-v2/service/sqs/types"
)

// SqsActions encapsulates the Amazon Simple Queue Service (Amazon SQS) actions
// used in the examples.
type SqsActions struct {
 SqsClient *sqs.Client
}



// CreateQueue creates an Amazon SQS queue with the specified name. You can specify
// whether the queue is created as a FIFO queue.
func (actor SqsActions) CreateQueue(ctx context.Context, queueName string,
 isFifoQueue bool) (string, error) {
 var queueUrl string
 queueAttributes := map[string]string{}
 if isFifoQueue {
  queueAttributes["FifoQueue"] = "true"
 }
 queue, err := actor.SqsClient.CreateQueue(ctx, &sqs.CreateQueueInput{
  QueueName:  aws.String(queueName),
  Attributes: queueAttributes,
 })
 if err != nil {
  log.Printf("Couldn't create queue %v. Here's why: %v\n", queueName, err)
```

```
 } else {
  queueUrl = *queue.QueueUrl
 }

 return queueUrl, err
}



// GetQueueArn uses the GetQueueAttributes action to get the Amazon Resource Name
 (ARN)
// of an Amazon SQS queue.
func (actor SqsActions) GetQueueArn(ctx context.Context, queueUrl string) (string,
 error) {
 var queueArn string
 arnAttributeName := types.QueueAttributeNameQueueArn
 attribute, err := actor.SqsClient.GetQueueAttributes(ctx,
 &sqs.GetQueueAttributesInput{
  QueueUrl:       aws.String(queueUrl),
  AttributeNames: []types.QueueAttributeName{arnAttributeName},
 })
 if err != nil {
  log.Printf("Couldn't get ARN for queue %v. Here's why: %v\n", queueUrl, err)
 } else {
  queueArn = attribute.Attributes[string(arnAttributeName)]
 }
 return queueArn, err
}



// AttachSendMessagePolicy uses the SetQueueAttributes action to attach a policy to
 an
// Amazon SQS queue that allows the specified Amazon SNS topic to send messages to
 the
// queue.
func (actor SqsActions) AttachSendMessagePolicy(ctx context.Context, queueUrl
 string, queueArn string, topicArn string) error {
 policyDoc := PolicyDocument{
  Version: "2012-10-17",
  Statement: []PolicyStatement{{
   Effect:    "Allow",
   Action:    "sqs:SendMessage",
   Principal: map[string]string{"Service": "sns.amazonaws.com"},
```

```
    Resource:  aws.String(queueArn),
    Condition: PolicyCondition{"ArnEquals": map[string]string{"aws:SourceArn":
 topicArn}},
  }},
 }
 policyBytes, err := json.Marshal(policyDoc)
 if err != nil {
  log.Printf("Couldn't create policy document. Here's why: %v\n", err)
  return err
 }
 _, err = actor.SqsClient.SetQueueAttributes(ctx, &sqs.SetQueueAttributesInput{
  Attributes: map[string]string{
   string(types.QueueAttributeNamePolicy): string(policyBytes),
  },
  QueueUrl: aws.String(queueUrl),
 })
 if err != nil {
  log.Printf("Couldn't set send message policy on queue %v. Here's why: %v\n",
 queueUrl, err)
 }
 return err
}

// PolicyDocument defines a policy document as a Go struct that can be serialized
// to JSON.
type PolicyDocument struct {
 Version   string
 Statement []PolicyStatement
}

// PolicyStatement defines a statement in a policy document.
type PolicyStatement struct {
 Effect    string
 Action    string
 Principal map[string]string `json:",omitempty"`
 Resource  *string           `json:",omitempty"`
 Condition PolicyCondition   `json:",omitempty"`
}

// PolicyCondition defines a condition in a policy.
type PolicyCondition map[string]map[string]string
```

```
// GetMessages uses the ReceiveMessage action to get messages from an Amazon SQS
 queue.
func (actor SqsActions) GetMessages(ctx context.Context, queueUrl string,
 maxMessages int32, waitTime int32) ([]types.Message, error) {
 var messages []types.Message
 result, err := actor.SqsClient.ReceiveMessage(ctx, &sqs.ReceiveMessageInput{
  QueueUrl:            aws.String(queueUrl),
  MaxNumberOfMessages: maxMessages,
  WaitTimeSeconds:     waitTime,
 })
 if err != nil {
  log.Printf("Couldn't get messages from queue %v. Here's why: %v\n", queueUrl, err)
 } else {
  messages = result.Messages
 }
 return messages, err
}



// DeleteMessages uses the DeleteMessageBatch action to delete a batch of messages
 from
// an Amazon SQS queue.
func (actor SqsActions) DeleteMessages(ctx context.Context, queueUrl string,
 messages []types.Message) error {
 entries := make([]types.DeleteMessageBatchRequestEntry, len(messages))
 for msgIndex := range messages {
  entries[msgIndex].Id = aws.String(fmt.Sprintf("%v", msgIndex))
  entries[msgIndex].ReceiptHandle = messages[msgIndex].ReceiptHandle
 }
 _, err := actor.SqsClient.DeleteMessageBatch(ctx, &sqs.DeleteMessageBatchInput{
  Entries:  entries,
  QueueUrl: aws.String(queueUrl),
 })
 if err != nil {
  log.Printf("Couldn't delete messages from queue %v. Here's why: %v\n", queueUrl,
 err)
 }
 return err
}



// DeleteQueue deletes an Amazon SQS queue.
```

```
func (actor SqsActions) DeleteQueue(ctx context.Context, queueUrl string) error {
 _, err := actor.SqsClient.DeleteQueue(ctx, &sqs.DeleteQueueInput{
  QueueUrl: aws.String(queueUrl)})
 if err != nil {
  log.Printf("Couldn't delete queue %v. Here's why: %v\n", queueUrl, err)
 }
 return err
}
```

Bereinigen von Ressourcen.

```
import (
 "context"
 "fmt"
 "log"
 "topics_and_queues/actions"
)

// Resources keeps track of AWS resources created during an example and handles
// cleanup when the example finishes.
type Resources struct {
 topicArn  string
 queueUrls []string
 snsActor  *actions.SnsActions
 sqsActor  *actions.SqsActions
}

// Cleanup deletes all AWS resources created during an example.
func (resources Resources) Cleanup(ctx context.Context) {
 defer func() {
  if r := recover(); r != nil {
   fmt.Println("Something went wrong during cleanup. Use the AWS Management Console
\n" +
    "to remove any remaining resources that were created for this scenario.")
  }
 }()

 var err error
 if resources.topicArn != "" {
  log.Printf("Deleting topic %v.\n", resources.topicArn)
```

```
  err = resources.snsActor.DeleteTopic(ctx, resources.topicArn)
  if err != nil {
   panic(err)
  }
 }

 for _, queueUrl := range resources.queueUrls {
  log.Printf("Deleting queue %v.\n", queueUrl)
  err = resources.sqsActor.DeleteQueue(ctx, queueUrl)
  if err != nil {
   panic(err)
  }
 }
}
```

- API-Details finden Sie in den folgenden Themen der AWS SDK für Go -API-Referenz.
  - CreateQueue
  - CreateTopic
  - DeleteMessageBatch
  - DeleteQueue
  - DeleteTopic
  - GetQueueAttributes
  - Veröffentlichen
  - ReceiveMessage
  - SetQueueAttributes
  - Abonnieren
  - Unsubscribe

Serverless-Beispiele

Eine Lambda-Funktion über einen Amazon-SNS-Trigger aufrufen

Das folgende Codebeispiel zeigt, wie eine Lambda-Funktion implementiert wird, die ein Ereignis empfängt, das durch den Empfang von Nachrichten von einem SNS-Thema ausgelöst wird. Die

Funktion ruft die Nachrichten aus dem Ereignisparameter ab und protokolliert den Inhalt jeder Nachricht.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu. GitHub Das vollständige Beispiel sowie eine Anleitung zum Einrichten und Ausführen finden Sie im Repository mit Serverless-Beispielen.

Nutzen eines SNS-Ereignisses mit Lambda unter Verwendung von Go.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
package main

import (
 "context"
 "fmt"

 "github.com/aws/aws-lambda-go/events"
 "github.com/aws/aws-lambda-go/lambda"
)

func handler(ctx context.Context, snsEvent events.SNSEvent) {
 for _, record := range snsEvent.Records {
  processMessage(record)
 }
 fmt.Println("done")
}

func processMessage(record events.SNSEventRecord) {
 message := record.SNS.Message
 fmt.Printf("Processed message: %s\n", message)
 // TODO: Process your record here
}

func main() {
 lambda.Start(handler)
}
```

Amazon SQS SQS-Beispiele mit SDK for Go V2

Die folgenden Codebeispiele zeigen Ihnen, wie Sie mithilfe von AWS SDK für Go V2 mit Amazon SQS Aktionen ausführen und allgemeine Szenarien implementieren.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Service-Funktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarios anzeigen.

Szenarien sind Code-Beispiele, die Ihnen zeigen, wie Sie bestimmte Aufgaben ausführen, indem Sie mehrere Funktionen innerhalb eines Services aufrufen oder mit anderen AWS-Services kombinieren.

Jedes Beispiel enthält einen Link zum vollständigen Quellcode, in dem Sie Anweisungen zur Einrichtung und Ausführung des Codes im Kontext finden.

Erste Schritte

Hallo Amazon SQS

Die folgenden Codebeispiele zeigen, wie Sie mit Amazon SQS beginnen können.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
package main

import (
 "context"
 "fmt"
 "log"

 "github.com/aws/aws-sdk-go-v2/config"
 "github.com/aws/aws-sdk-go-v2/service/sqs"
)
```

```
// main uses the AWS SDK for Go V2 to create an Amazon Simple Queue Service
// (Amazon SQS) client and list the queues in your account.
// This example uses the default settings specified in your shared credentials
// and config files.
func main() {
 ctx := context.Background()
 sdkConfig, err := config.LoadDefaultConfig(ctx)
 if err != nil {
  fmt.Println("Couldn't load default configuration. Have you set up your AWS
 account?")
  fmt.Println(err)
  return
 }
 sqsClient := sqs.NewFromConfig(sdkConfig)
 fmt.Println("Let's list the queues for your account.")
 var queueUrls []string
 paginator := sqs.NewListQueuesPaginator(sqsClient, &sqs.ListQueuesInput{})
 for paginator.HasMorePages() {
  output, err := paginator.NextPage(ctx)
  if err != nil {
   log.Printf("Couldn't get queues. Here's why: %v\n", err)
   break
  } else {
   queueUrls = append(queueUrls, output.QueueUrls...)
  }
 }
 if len(queueUrls) == 0 {
  fmt.Println("You don't have any queues!")
 } else {
  for _, queueUrl := range queueUrls {
   fmt.Printf("\t%v\n", queueUrl)
  }
 }
}
```

- Einzelheiten zur API finden Sie ListQueuesin der AWS SDK für Go API-Referenz.

Themen

- Aktionen
- Szenarien

- Serverless-Beispiele

Aktionen

`CreateQueue`

Das folgende Codebeispiel zeigt die Verwendung`CreateQueue`.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import (
 "context"
 "encoding/json"
 "fmt"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/sqs"
 "github.com/aws/aws-sdk-go-v2/service/sqs/types"
)

// SqsActions encapsulates the Amazon Simple Queue Service (Amazon SQS) actions
// used in the examples.
type SqsActions struct {
 SqsClient *sqs.Client
}



// CreateQueue creates an Amazon SQS queue with the specified name. You can specify
// whether the queue is created as a FIFO queue.
func (actor SqsActions) CreateQueue(ctx context.Context, queueName string,
 isFifoQueue bool) (string, error) {
 var queueUrl string
```

```
 queueAttributes := map[string]string{}
 if isFifoQueue {
  queueAttributes["FifoQueue"] = "true"
 }
 queue, err := actor.SqsClient.CreateQueue(ctx, &sqs.CreateQueueInput{
  QueueName:  aws.String(queueName),
  Attributes: queueAttributes,
 })
 if err != nil {
  log.Printf("Couldn't create queue %v. Here's why: %v\n", queueName, err)
 } else {
  queueUrl = *queue.QueueUrl
 }

 return queueUrl, err
}
```

- Einzelheiten zur API finden Sie CreateQueuein der AWS SDK für Go API-Referenz.

`DeleteMessageBatch`

Das folgende Codebeispiel zeigt die Verwendung`DeleteMessageBatch`.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import (
 "context"
 "encoding/json"
 "fmt"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
```

```
 "github.com/aws/aws-sdk-go-v2/service/sqs"
 "github.com/aws/aws-sdk-go-v2/service/sqs/types"
)

// SqsActions encapsulates the Amazon Simple Queue Service (Amazon SQS) actions
// used in the examples.
type SqsActions struct {
 SqsClient *sqs.Client
}



// DeleteMessages uses the DeleteMessageBatch action to delete a batch of messages
 from
// an Amazon SQS queue.
func (actor SqsActions) DeleteMessages(ctx context.Context, queueUrl string,
 messages []types.Message) error {
 entries := make([]types.DeleteMessageBatchRequestEntry, len(messages))
 for msgIndex := range messages {
  entries[msgIndex].Id = aws.String(fmt.Sprintf("%v", msgIndex))
  entries[msgIndex].ReceiptHandle = messages[msgIndex].ReceiptHandle
 }
 _, err := actor.SqsClient.DeleteMessageBatch(ctx, &sqs.DeleteMessageBatchInput{
  Entries:  entries,
  QueueUrl: aws.String(queueUrl),
 })
 if err != nil {
  log.Printf("Couldn't delete messages from queue %v. Here's why: %v\n", queueUrl,
 err)
 }
 return err
}
```

- Einzelheiten zur API finden Sie DeleteMessageBatchin der AWS SDK für Go API-Referenz.

`DeleteQueue`

Das folgende Codebeispiel zeigt die Verwendung`DeleteQueue`.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import (
 "context"
 "encoding/json"
 "fmt"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/sqs"
 "github.com/aws/aws-sdk-go-v2/service/sqs/types"
)

// SqsActions encapsulates the Amazon Simple Queue Service (Amazon SQS) actions
// used in the examples.
type SqsActions struct {
 SqsClient *sqs.Client
}



// DeleteQueue deletes an Amazon SQS queue.
func (actor SqsActions) DeleteQueue(ctx context.Context, queueUrl string) error {
 _, err := actor.SqsClient.DeleteQueue(ctx, &sqs.DeleteQueueInput{
  QueueUrl: aws.String(queueUrl)})
 if err != nil {
  log.Printf("Couldn't delete queue %v. Here's why: %v\n", queueUrl, err)
 }
 return err
}
```

- Einzelheiten zur API finden Sie DeleteQueuein der AWS SDK für Go API-Referenz.

`GetQueueAttributes`

Das folgende Codebeispiel zeigt die Verwendung`GetQueueAttributes`.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import (
 "context"
 "encoding/json"
 "fmt"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/sqs"
 "github.com/aws/aws-sdk-go-v2/service/sqs/types"
)

// SqsActions encapsulates the Amazon Simple Queue Service (Amazon SQS) actions
// used in the examples.
type SqsActions struct {
 SqsClient *sqs.Client
}



// GetQueueArn uses the GetQueueAttributes action to get the Amazon Resource Name
 (ARN)
// of an Amazon SQS queue.
func (actor SqsActions) GetQueueArn(ctx context.Context, queueUrl string) (string,
 error) {
 var queueArn string
 arnAttributeName := types.QueueAttributeNameQueueArn
 attribute, err := actor.SqsClient.GetQueueAttributes(ctx,
 &sqs.GetQueueAttributesInput{
  QueueUrl:       aws.String(queueUrl),
  AttributeNames: []types.QueueAttributeName{arnAttributeName},
```

```
 })
 if err != nil {
  log.Printf("Couldn't get ARN for queue %v. Here's why: %v\n", queueUrl, err)
 } else {
  queueArn = attribute.Attributes[string(arnAttributeName)]
 }
 return queueArn, err
}
```

- Einzelheiten zur API finden Sie GetQueueAttributesin der AWS SDK für Go API-Referenz.

`ListQueues`

Das folgende Codebeispiel zeigt die Verwendung`ListQueues`.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
package main

import (
 "context"
 "fmt"
 "log"

 "github.com/aws/aws-sdk-go-v2/config"
 "github.com/aws/aws-sdk-go-v2/service/sqs"
)

// main uses the AWS SDK for Go V2 to create an Amazon Simple Queue Service
// (Amazon SQS) client and list the queues in your account.
// This example uses the default settings specified in your shared credentials
// and config files.
```

```
func main() {
 ctx := context.Background()
 sdkConfig, err := config.LoadDefaultConfig(ctx)
 if err != nil {
  fmt.Println("Couldn't load default configuration. Have you set up your AWS
 account?")
  fmt.Println(err)
  return
 }
 sqsClient := sqs.NewFromConfig(sdkConfig)
 fmt.Println("Let's list the queues for your account.")
 var queueUrls []string
 paginator := sqs.NewListQueuesPaginator(sqsClient, &sqs.ListQueuesInput{})
 for paginator.HasMorePages() {
  output, err := paginator.NextPage(ctx)
  if err != nil {
   log.Printf("Couldn't get queues. Here's why: %v\n", err)
   break
  } else {
   queueUrls = append(queueUrls, output.QueueUrls...)
  }
 }
 if len(queueUrls) == 0 {
  fmt.Println("You don't have any queues!")
 } else {
  for _, queueUrl := range queueUrls {
   fmt.Printf("\t%v\n", queueUrl)
  }
 }
}
```

- Einzelheiten zur API finden Sie ListQueuesin der AWS SDK für Go API-Referenz.

`ReceiveMessage`

Das folgende Codebeispiel zeigt die Verwendung`ReceiveMessage`.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import (
 "context"
 "encoding/json"
 "fmt"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/sqs"
 "github.com/aws/aws-sdk-go-v2/service/sqs/types"
)

// SqsActions encapsulates the Amazon Simple Queue Service (Amazon SQS) actions
// used in the examples.
type SqsActions struct {
 SqsClient *sqs.Client
}



// GetMessages uses the ReceiveMessage action to get messages from an Amazon SQS
 queue.
func (actor SqsActions) GetMessages(ctx context.Context, queueUrl string,
 maxMessages int32, waitTime int32) ([]types.Message, error) {
 var messages []types.Message
 result, err := actor.SqsClient.ReceiveMessage(ctx, &sqs.ReceiveMessageInput{
  QueueUrl:            aws.String(queueUrl),
  MaxNumberOfMessages: maxMessages,
  WaitTimeSeconds:     waitTime,
 })
 if err != nil {
  log.Printf("Couldn't get messages from queue %v. Here's why: %v\n", queueUrl, err)
 } else {
  messages = result.Messages
```

```
 }
 return messages, err
}
```

- Einzelheiten zur API finden Sie ReceiveMessagein der AWS SDK für Go API-Referenz.

`SetQueueAttributes`

Das folgende Codebeispiel zeigt die Verwendung`SetQueueAttributes`.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import (
 "context"
 "encoding/json"
 "fmt"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/sqs"
 "github.com/aws/aws-sdk-go-v2/service/sqs/types"
)

// SqsActions encapsulates the Amazon Simple Queue Service (Amazon SQS) actions
// used in the examples.
type SqsActions struct {
 SqsClient *sqs.Client
}



// AttachSendMessagePolicy uses the SetQueueAttributes action to attach a policy to
 an
```

```
// Amazon SQS queue that allows the specified Amazon SNS topic to send messages to
 the
// queue.
func (actor SqsActions) AttachSendMessagePolicy(ctx context.Context, queueUrl
 string, queueArn string, topicArn string) error {
 policyDoc := PolicyDocument{
  Version: "2012-10-17",
  Statement: []PolicyStatement{{
   Effect:    "Allow",
   Action:    "sqs:SendMessage",
   Principal: map[string]string{"Service": "sns.amazonaws.com"},
   Resource:  aws.String(queueArn),
   Condition: PolicyCondition{"ArnEquals": map[string]string{"aws:SourceArn":
 topicArn}},
  }},
 }
 policyBytes, err := json.Marshal(policyDoc)
 if err != nil {
  log.Printf("Couldn't create policy document. Here's why: %v\n", err)
  return err
 }
 _, err = actor.SqsClient.SetQueueAttributes(ctx, &sqs.SetQueueAttributesInput{
  Attributes: map[string]string{
   string(types.QueueAttributeNamePolicy): string(policyBytes),
  },
  QueueUrl: aws.String(queueUrl),
 })
 if err != nil {
  log.Printf("Couldn't set send message policy on queue %v. Here's why: %v\n",
 queueUrl, err)
 }
 return err
}

// PolicyDocument defines a policy document as a Go struct that can be serialized
// to JSON.
type PolicyDocument struct {
 Version   string
 Statement []PolicyStatement
}

// PolicyStatement defines a statement in a policy document.
type PolicyStatement struct {
 Effect    string
```

```
 Action    string
 Principal map[string]string `json:",omitempty"`
 Resource  *string           `json:",omitempty"`
 Condition PolicyCondition   `json:",omitempty"`
}

// PolicyCondition defines a condition in a policy.
type PolicyCondition map[string]map[string]string
```

- Einzelheiten zur API finden Sie SetQueueAttributesin der AWS SDK für Go API-Referenz.

Szenarien

Veröffentlichen Sie Nachrichten in Warteschlangen

Wie das aussehen kann, sehen Sie am nachfolgenden Beispielcode:

- Erstellen Sie ein Thema (FIFO oder Nicht-FIFO).
- Abonnieren Sie mehrere Warteschlangen für das Thema mit der Option, einen Filter anzuwenden.
- Veröffentlichen Sie eine Nachricht im Thema.
- Fragen Sie die Warteschlangen nach empfangenen Nachrichten ab.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Führen Sie ein interaktives Szenario an einem Prompt aus.

```
import (
 "context"
 "encoding/json"
 "fmt"
 "log"
```

```
 "strings"
 "topics_and_queues/actions"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/sns"
 "github.com/aws/aws-sdk-go-v2/service/sqs"
 "github.com/aws/aws-sdk-go-v2/service/sqs/types"
 "github.com/awsdocs/aws-doc-sdk-examples/gov2/demotools"
)

const FIFO_SUFFIX = ".fifo"
const TONE_KEY = "tone"

var ToneChoices = []string{"cheerful", "funny", "serious", "sincere"}

// MessageBody is used to deserialize the body of a message from a JSON string.
type MessageBody struct {
 Message string
}

// ScenarioRunner separates the steps of this scenario into individual functions so
 that
// they are simpler to read and understand.
type ScenarioRunner struct {
 questioner demotools.IQuestioner
 snsActor   *actions.SnsActions
 sqsActor   *actions.SqsActions
}

func (runner ScenarioRunner) CreateTopic(ctx context.Context) (string, string, bool,
 bool) {
 log.Println("SNS topics can be configured as FIFO (First-In-First-Out) or standard.
\n" +
  "FIFO topics deliver messages in order and support deduplication and message
 filtering.")
 isFifoTopic := runner.questioner.AskBool("\nWould you like to work with FIFO
 topics? (y/n) ", "y")

 contentBasedDeduplication := false
 if isFifoTopic {
  log.Println(strings.Repeat("-", 88))
  log.Println("Because you have chosen a FIFO topic, deduplication is supported.\n"
 +
```

```
   "Deduplication IDs are either set in the message or are automatically generated
\n" +
   "from content using a hash function. If a message is successfully published to\n"
 +
   "an SNS FIFO topic, any message published and determined to have the same\n" +
   "deduplication ID, within the five-minute deduplication interval, is accepted\n"
 +
   "but not delivered. For more information about deduplication, see:\n" +
   "\thttps://docs.aws.amazon.com/sns/latest/dg/fifo-message-dedup.html.")
  contentBasedDeduplication = runner.questioner.AskBool(
   "\nDo you want to use content-based deduplication instead of entering a
 deduplication ID? (y/n) ", "y")
 }
 log.Println(strings.Repeat("-", 88))

 topicName := runner.questioner.Ask("Enter a name for your SNS topic. ")
 if isFifoTopic {
  topicName = fmt.Sprintf("%v%v", topicName, FIFO_SUFFIX)
  log.Printf("Because you have selected a FIFO topic, '%v' must be appended to\n"+
   "the topic name.", FIFO_SUFFIX)
 }

 topicArn, err := runner.snsActor.CreateTopic(ctx, topicName, isFifoTopic,
 contentBasedDeduplication)
 if err != nil {
  panic(err)
 }
 log.Printf("Your new topic with the name '%v' and Amazon Resource Name (ARN) \n"+
  "'%v' has been created.", topicName, topicArn)

 return topicName, topicArn, isFifoTopic, contentBasedDeduplication
}

func (runner ScenarioRunner) CreateQueue(ctx context.Context, ordinal string,
 isFifoTopic bool) (string, string) {
 queueName := runner.questioner.Ask(fmt.Sprintf("Enter a name for the %v SQS queue.
 ", ordinal))
 if isFifoTopic {
  queueName = fmt.Sprintf("%v%v", queueName, FIFO_SUFFIX)
  if ordinal == "first" {
   log.Printf("Because you are creating a FIFO SQS queue, '%v' must "+
    "be appended to the queue name.\n", FIFO_SUFFIX)
  }
 }
```

```
 queueUrl, err := runner.sqsActor.CreateQueue(ctx, queueName, isFifoTopic)
 if err != nil {
  panic(err)
 }
 log.Printf("Your new SQS queue with the name '%v' and the queue URL "+
  "'%v' has been created.", queueName, queueUrl)

 return queueName, queueUrl
}

func (runner ScenarioRunner) SubscribeQueueToTopic(
 ctx context.Context, queueName string, queueUrl string, topicName string, topicArn
 string, ordinal string,
 isFifoTopic bool) (string, bool) {

 queueArn, err := runner.sqsActor.GetQueueArn(ctx, queueUrl)
 if err != nil {
  panic(err)
 }
 log.Printf("The ARN of your queue is: %v.\n", queueArn)

 err = runner.sqsActor.AttachSendMessagePolicy(ctx, queueUrl, queueArn, topicArn)
 if err != nil {
  panic(err)
 }
 log.Println("Attached an IAM policy to the queue so the SNS topic can send " +
  "messages to it.")
 log.Println(strings.Repeat("-", 88))

 var filterPolicy map[string][]string
 if isFifoTopic {
  if ordinal == "first" {
   log.Println("Subscriptions to a FIFO topic can have filters.\n" +
    "If you add a filter to this subscription, then only the filtered messages\n" +
    "will be received in the queue.\n" +
    "For information about message filtering, see\n" +
    "\thttps://docs.aws.amazon.com/sns/latest/dg/sns-message-filtering.html\n" +
    "For this example, you can filter messages by a \"tone\" attribute.")
  }

  wantFiltering := runner.questioner.AskBool(
   fmt.Sprintf("Do you want to filter messages that are sent to \"%v\"\n"+
    "from the %v topic? (y/n) ", queueName, topicName), "y")
  if wantFiltering {
```

```
    log.Println("You can filter messages by one or more of the following \"tone\"
  attributes.")

    var toneSelections []string
    askAboutTones := true
    for askAboutTones {
     toneIndex := runner.questioner.AskChoice(
      "Enter the number of the tone you want to filter by:\n", ToneChoices)
     toneSelections = append(toneSelections, ToneChoices[toneIndex])
     askAboutTones = runner.questioner.AskBool("Do you want to add another tone to
  the filter? (y/n) ", "y")
    }
    log.Printf("Your subscription will be filtered to only pass the following tones:
  %v\n", toneSelections)
    filterPolicy = map[string][]string{TONE_KEY: toneSelections}
   }
  }

  subscriptionArn, err := runner.snsActor.SubscribeQueue(ctx, topicArn, queueArn,
  filterPolicy)
  if err != nil {
   panic(err)
  }
  log.Printf("The queue %v is now subscribed to the topic %v with the subscription
  ARN %v.\n",
   queueName, topicName, subscriptionArn)

  return subscriptionArn, filterPolicy != nil
}

func (runner ScenarioRunner) PublishMessages(ctx context.Context, topicArn string,
  isFifoTopic bool, contentBasedDeduplication bool, usingFilters bool) {
  var message string
  var groupId string
  var dedupId string
  var toneSelection string
  publishMore := true
  for publishMore {
   groupId = ""
   dedupId = ""
   toneSelection = ""
   message = runner.questioner.Ask("Enter a message to publish: ")
   if isFifoTopic {
```

```
   log.Println("Because you are using a FIFO topic, you must set a message group ID.
\n" +
    "All messages within the same group will be received in the order they were
 published.")
   groupId = runner.questioner.Ask("Enter a message group ID: ")
   if !contentBasedDeduplication {
    log.Println("Because you are not using content-based deduplication,\n" +
     "you must enter a deduplication ID.")
    dedupId = runner.questioner.Ask("Enter a deduplication ID: ")
   }
  }
  if usingFilters {
   if runner.questioner.AskBool("Add a tone attribute so this message can be
 filtered? (y/n) ", "y") {
    toneIndex := runner.questioner.AskChoice(
     "Enter the number of the tone you want to filter by:\n", ToneChoices)
    toneSelection = ToneChoices[toneIndex]
   }
  }

  err := runner.snsActor.Publish(ctx, topicArn, message, groupId, dedupId, TONE_KEY,
 toneSelection)
  if err != nil {
   panic(err)
  }
  log.Println(("Your message was published."))

  publishMore = runner.questioner.AskBool("Do you want to publish another messsage?
 (y/n) ", "y")
 }
}

func (runner ScenarioRunner) PollForMessages(ctx context.Context, queueUrls
 []string) {
 log.Println("Polling queues for messages...")
 for _, queueUrl := range queueUrls {
  var messages []types.Message
  for {
   currentMsgs, err := runner.sqsActor.GetMessages(ctx, queueUrl, 10, 1)
   if err != nil {
    panic(err)
   }
   if len(currentMsgs) == 0 {
    break
```

```
    }
    messages = append(messages, currentMsgs...)
   }
   if len(messages) == 0 {
    log.Printf("No messages were received by queue %v.\n", queueUrl)
   } else if len(messages) == 1 {
    log.Printf("One message was received by queue %v:\n", queueUrl)

   } else {
    log.Printf("%v messages were received by queue %v:\n", len(messages), queueUrl)
   }
   for msgIndex, message := range messages {
    messageBody := MessageBody{}
    err := json.Unmarshal([]byte(*message.Body), &messageBody)
    if err != nil {
     panic(err)
    }
    log.Printf("Message %v: %v\n", msgIndex+1, messageBody.Message)
   }

   if len(messages) > 0 {
    log.Printf("Deleting %v messages from queue %v.\n", len(messages), queueUrl)
    err := runner.sqsActor.DeleteMessages(ctx, queueUrl, messages)
    if err != nil {
     panic(err)
    }
   }
  }
}

// RunTopicsAndQueuesScenario is an interactive example that shows you how to use
 the
// AWS SDK for Go to create and use Amazon SNS topics and Amazon SQS queues.
//
// 1. Create a topic (FIFO or non-FIFO).
// 2. Subscribe several queues to the topic with an option to apply a filter.
// 3. Publish messages to the topic.
// 4. Poll the queues for messages received.
// 5. Delete the topic and the queues.
//
// This example creates service clients from the specified sdkConfig so that
// you can replace it with a mocked or stubbed config for unit testing.
//
```

```
// It uses a questioner from the `demotools` package to get input during the
 example.
// This package can be found in the ..\..\demotools folder of this repo.
func RunTopicsAndQueuesScenario(
 ctx context.Context, sdkConfig aws.Config, questioner demotools.IQuestioner) {
 resources := Resources{}
 defer func() {
  if r := recover(); r != nil {
   log.Println("Something went wrong with the demo.\n" +
    "Cleaning up any resources that were created...")
   resources.Cleanup(ctx)
  }
 }()
 queueCount := 2

 log.Println(strings.Repeat("-", 88))
 log.Printf("Welcome to messaging with topics and queues.\n\n"+
  "In this scenario, you will create an SNS topic and subscribe %v SQS queues to the
\n"+
  "topic. You can select from several options for configuring the topic and the\n"+
  "subscriptions for the queues. You can then post to the topic and see the results
\n"+
  "in the queues.\n", queueCount)

 log.Println(strings.Repeat("-", 88))

 runner := ScenarioRunner{
  questioner: questioner,
  snsActor:   &actions.SnsActions{SnsClient: sns.NewFromConfig(sdkConfig)},
  sqsActor:   &actions.SqsActions{SqsClient: sqs.NewFromConfig(sdkConfig)},
 }
 resources.snsActor = runner.snsActor
 resources.sqsActor = runner.sqsActor

 topicName, topicArn, isFifoTopic, contentBasedDeduplication :=
 runner.CreateTopic(ctx)
 resources.topicArn = topicArn
 log.Println(strings.Repeat("-", 88))

 log.Printf("Now you will create %v SQS queues and subscribe them to the topic.\n",
 queueCount)
 ordinals := []string{"first", "next"}
 usingFilters := false
 for _, ordinal := range ordinals {
```

```
  queueName, queueUrl := runner.CreateQueue(ctx, ordinal, isFifoTopic)
  resources.queueUrls = append(resources.queueUrls, queueUrl)

  _, filtering := runner.SubscribeQueueToTopic(ctx, queueName, queueUrl, topicName,
 topicArn, ordinal, isFifoTopic)
  usingFilters = usingFilters || filtering
 }

 log.Println(strings.Repeat("-", 88))
 runner.PublishMessages(ctx, topicArn, isFifoTopic, contentBasedDeduplication,
 usingFilters)
 log.Println(strings.Repeat("-", 88))
 runner.PollForMessages(ctx, resources.queueUrls)

 log.Println(strings.Repeat("-", 88))

 wantCleanup := questioner.AskBool("Do you want to remove all AWS resources created
 for this scenario? (y/n) ", "y")
 if wantCleanup {
  log.Println("Cleaning up resources...")
  resources.Cleanup(ctx)
 }

 log.Println(strings.Repeat("-", 88))
 log.Println("Thanks for watching!")
 log.Println(strings.Repeat("-", 88))
}
```

Definieren Sie eine Struktur, die die in diesem Beispiel verwendeten Amazon SNS SNS-Aktionen umschließt.

```
import (
 "context"
 "encoding/json"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/sns"
 "github.com/aws/aws-sdk-go-v2/service/sns/types"
)
```

```
// SnsActions encapsulates the Amazon Simple Notification Service (Amazon SNS)
 actions
// used in the examples.
type SnsActions struct {
 SnsClient *sns.Client
}



// CreateTopic creates an Amazon SNS topic with the specified name. You can
 optionally
// specify that the topic is created as a FIFO topic and whether it uses content-
based
// deduplication instead of ID-based deduplication.
func (actor SnsActions) CreateTopic(ctx context.Context, topicName string,
 isFifoTopic bool, contentBasedDeduplication bool) (string, error) {
 var topicArn string
 topicAttributes := map[string]string{}
 if isFifoTopic {
  topicAttributes["FifoTopic"] = "true"
 }
 if contentBasedDeduplication {
  topicAttributes["ContentBasedDeduplication"] = "true"
 }
 topic, err := actor.SnsClient.CreateTopic(ctx, &sns.CreateTopicInput{
  Name:       aws.String(topicName),
  Attributes: topicAttributes,
 })
 if err != nil {
  log.Printf("Couldn't create topic %v. Here's why: %v\n", topicName, err)
 } else {
  topicArn = *topic.TopicArn
 }

 return topicArn, err
}



// DeleteTopic delete an Amazon SNS topic.
func (actor SnsActions) DeleteTopic(ctx context.Context, topicArn string) error {
 _, err := actor.SnsClient.DeleteTopic(ctx, &sns.DeleteTopicInput{
  TopicArn: aws.String(topicArn)})
```

```
 if err != nil {
  log.Printf("Couldn't delete topic %v. Here's why: %v\n", topicArn, err)
 }
 return err
}



// SubscribeQueue subscribes an Amazon Simple Queue Service (Amazon SQS) queue to an
// Amazon SNS topic. When filterMap is not nil, it is used to specify a filter
 policy
// so that messages are only sent to the queue when the message has the specified
 attributes.
func (actor SnsActions) SubscribeQueue(ctx context.Context, topicArn string,
 queueArn string, filterMap map[string][]string) (string, error) {
 var subscriptionArn string
 var attributes map[string]string
 if filterMap != nil {
  filterBytes, err := json.Marshal(filterMap)
  if err != nil {
   log.Printf("Couldn't create filter policy, here's why: %v\n", err)
   return "", err
  }
  attributes = map[string]string{"FilterPolicy": string(filterBytes)}
 }
 output, err := actor.SnsClient.Subscribe(ctx, &sns.SubscribeInput{
  Protocol:              aws.String("sqs"),
  TopicArn:              aws.String(topicArn),
  Attributes:            attributes,
  Endpoint:              aws.String(queueArn),
  ReturnSubscriptionArn: true,
 })
 if err != nil {
  log.Printf("Couldn't susbscribe queue %v to topic %v. Here's why: %v\n",
   queueArn, topicArn, err)
 } else {
  subscriptionArn = *output.SubscriptionArn
 }

 return subscriptionArn, err
}
```

```
// Publish publishes a message to an Amazon SNS topic. The message is then sent to
 all
// subscribers. When the topic is a FIFO topic, the message must also contain a
 group ID
// and, when ID-based deduplication is used, a deduplication ID. An optional key-
value
// filter attribute can be specified so that the message can be filtered according
 to
// a filter policy.
func (actor SnsActions) Publish(ctx context.Context, topicArn string, message
 string, groupId string, dedupId string, filterKey string, filterValue string) error
 {
 publishInput := sns.PublishInput{TopicArn: aws.String(topicArn), Message:
 aws.String(message)}
 if groupId != "" {
  publishInput.MessageGroupId = aws.String(groupId)
 }
 if dedupId != "" {
  publishInput.MessageDeduplicationId = aws.String(dedupId)
 }
 if filterKey != "" && filterValue != "" {
  publishInput.MessageAttributes = map[string]types.MessageAttributeValue{
   filterKey: {DataType: aws.String("String"), StringValue:
 aws.String(filterValue)},
  }
 }
 _, err := actor.SnsClient.Publish(ctx, &publishInput)
 if err != nil {
  log.Printf("Couldn't publish message to topic %v. Here's why: %v", topicArn, err)
 }
 return err
}
```

Definieren Sie eine Struktur, die die in diesem Beispiel verwendeten Amazon SQS SQS-Aktionen umschließt.

```
import (
 "context"
 "encoding/json"
 "fmt"
```

```
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/sqs"
 "github.com/aws/aws-sdk-go-v2/service/sqs/types"
)

// SqsActions encapsulates the Amazon Simple Queue Service (Amazon SQS) actions
// used in the examples.
type SqsActions struct {
 SqsClient *sqs.Client
}



// CreateQueue creates an Amazon SQS queue with the specified name. You can specify
// whether the queue is created as a FIFO queue.
func (actor SqsActions) CreateQueue(ctx context.Context, queueName string,
 isFifoQueue bool) (string, error) {
 var queueUrl string
 queueAttributes := map[string]string{}
 if isFifoQueue {
  queueAttributes["FifoQueue"] = "true"
 }
 queue, err := actor.SqsClient.CreateQueue(ctx, &sqs.CreateQueueInput{
  QueueName:  aws.String(queueName),
  Attributes: queueAttributes,
 })
 if err != nil {
  log.Printf("Couldn't create queue %v. Here's why: %v\n", queueName, err)
 } else {
  queueUrl = *queue.QueueUrl
 }

 return queueUrl, err
}



// GetQueueArn uses the GetQueueAttributes action to get the Amazon Resource Name
 (ARN)
// of an Amazon SQS queue.
func (actor SqsActions) GetQueueArn(ctx context.Context, queueUrl string) (string,
 error) {
```

```
 var queueArn string
 arnAttributeName := types.QueueAttributeNameQueueArn
 attribute, err := actor.SqsClient.GetQueueAttributes(ctx,
 &sqs.GetQueueAttributesInput{
  QueueUrl:       aws.String(queueUrl),
  AttributeNames: []types.QueueAttributeName{arnAttributeName},
 })
 if err != nil {
  log.Printf("Couldn't get ARN for queue %v. Here's why: %v\n", queueUrl, err)
 } else {
  queueArn = attribute.Attributes[string(arnAttributeName)]
 }
 return queueArn, err
}



// AttachSendMessagePolicy uses the SetQueueAttributes action to attach a policy to
 an
// Amazon SQS queue that allows the specified Amazon SNS topic to send messages to
 the
// queue.
func (actor SqsActions) AttachSendMessagePolicy(ctx context.Context, queueUrl
 string, queueArn string, topicArn string) error {
 policyDoc := PolicyDocument{
  Version: "2012-10-17",
  Statement: []PolicyStatement{{
   Effect:    "Allow",
   Action:    "sqs:SendMessage",
   Principal: map[string]string{"Service": "sns.amazonaws.com"},
   Resource:  aws.String(queueArn),
   Condition: PolicyCondition{"ArnEquals": map[string]string{"aws:SourceArn":
 topicArn}},
  }},
 }
 policyBytes, err := json.Marshal(policyDoc)
 if err != nil {
  log.Printf("Couldn't create policy document. Here's why: %v\n", err)
  return err
 }
 _, err = actor.SqsClient.SetQueueAttributes(ctx, &sqs.SetQueueAttributesInput{
  Attributes: map[string]string{
   string(types.QueueAttributeNamePolicy): string(policyBytes),
  },
```

```
  QueueUrl: aws.String(queueUrl),
 })
 if err != nil {
  log.Printf("Couldn't set send message policy on queue %v. Here's why: %v\n",
 queueUrl, err)
 }
 return err
}

// PolicyDocument defines a policy document as a Go struct that can be serialized
// to JSON.
type PolicyDocument struct {
 Version   string
 Statement []PolicyStatement
}

// PolicyStatement defines a statement in a policy document.
type PolicyStatement struct {
 Effect    string
 Action    string
 Principal map[string]string `json:",omitempty"`
 Resource  *string           `json:",omitempty"`
 Condition PolicyCondition   `json:",omitempty"`
}

// PolicyCondition defines a condition in a policy.
type PolicyCondition map[string]map[string]string



// GetMessages uses the ReceiveMessage action to get messages from an Amazon SQS
 queue.
func (actor SqsActions) GetMessages(ctx context.Context, queueUrl string,
 maxMessages int32, waitTime int32) ([]types.Message, error) {
 var messages []types.Message
 result, err := actor.SqsClient.ReceiveMessage(ctx, &sqs.ReceiveMessageInput{
  QueueUrl:            aws.String(queueUrl),
  MaxNumberOfMessages: maxMessages,
  WaitTimeSeconds:     waitTime,
 })
 if err != nil {
  log.Printf("Couldn't get messages from queue %v. Here's why: %v\n", queueUrl, err)
 } else {
  messages = result.Messages
```

```
 }
 return messages, err
}



// DeleteMessages uses the DeleteMessageBatch action to delete a batch of messages
 from
// an Amazon SQS queue.
func (actor SqsActions) DeleteMessages(ctx context.Context, queueUrl string,
 messages []types.Message) error {
 entries := make([]types.DeleteMessageBatchRequestEntry, len(messages))
 for msgIndex := range messages {
  entries[msgIndex].Id = aws.String(fmt.Sprintf("%v", msgIndex))
  entries[msgIndex].ReceiptHandle = messages[msgIndex].ReceiptHandle
 }
 _, err := actor.SqsClient.DeleteMessageBatch(ctx, &sqs.DeleteMessageBatchInput{
  Entries:  entries,
  QueueUrl: aws.String(queueUrl),
 })
 if err != nil {
  log.Printf("Couldn't delete messages from queue %v. Here's why: %v\n", queueUrl,
 err)
 }
 return err
}



// DeleteQueue deletes an Amazon SQS queue.
func (actor SqsActions) DeleteQueue(ctx context.Context, queueUrl string) error {
 _, err := actor.SqsClient.DeleteQueue(ctx, &sqs.DeleteQueueInput{
  QueueUrl: aws.String(queueUrl)})
 if err != nil {
  log.Printf("Couldn't delete queue %v. Here's why: %v\n", queueUrl, err)
 }
 return err
}
```

Bereinigen von Ressourcen.

```
import (
 "context"
 "fmt"
 "log"
 "topics_and_queues/actions"
)

// Resources keeps track of AWS resources created during an example and handles
// cleanup when the example finishes.
type Resources struct {
 topicArn  string
 queueUrls []string
 snsActor  *actions.SnsActions
 sqsActor  *actions.SqsActions
}

// Cleanup deletes all AWS resources created during an example.
func (resources Resources) Cleanup(ctx context.Context) {
 defer func() {
  if r := recover(); r != nil {
   fmt.Println("Something went wrong during cleanup. Use the AWS Management Console
\n" +
    "to remove any remaining resources that were created for this scenario.")
  }
 }()

 var err error
 if resources.topicArn != "" {
  log.Printf("Deleting topic %v.\n", resources.topicArn)
  err = resources.snsActor.DeleteTopic(ctx, resources.topicArn)
  if err != nil {
   panic(err)
  }
 }

 for _, queueUrl := range resources.queueUrls {
  log.Printf("Deleting queue %v.\n", queueUrl)
  err = resources.sqsActor.DeleteQueue(ctx, queueUrl)
  if err != nil {
   panic(err)
  }
 }
```

```
}
```

- API-Details finden Sie in den folgenden Themen der AWS SDK für Go -API-Referenz.
  - CreateQueue
  - CreateTopic
  - DeleteMessageBatch
  - DeleteQueue
  - DeleteTopic
  - GetQueueAttributes
  - Veröffentlichen
  - ReceiveMessage
  - SetQueueAttributes
  - Abonnieren
  - Unsubscribe

Serverless-Beispiele

Aufrufen einer Lambda-Funktion über einen Amazon-SQS-Auslöser

Das folgende Codebeispiel zeigt, wie eine Lambda-Funktion implementiert wird, die ein Ereignis empfängt, das durch den Empfang von Nachrichten aus einer SQS-Warteschlange ausgelöst wird. Die Funktion ruft die Nachrichten aus dem Ereignisparameter ab und protokolliert den Inhalt jeder Nachricht.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu. GitHub Das vollständige Beispiel sowie eine Anleitung zum Einrichten und Ausführen finden Sie im Repository mit Serverless-Beispielen.

Nutzen eines SQS-Ereignisses mit Lambda unter Verwendung von Go.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
package integration_sqs_to_lambda

import (
 "fmt"
 "github.com/aws/aws-lambda-go/events"
 "github.com/aws/aws-lambda-go/lambda"
)

func handler(event events.SQSEvent) error {
 for _, record := range event.Records {
  err := processMessage(record)
  if err != nil {
   return err
  }
 }
 fmt.Println("done")
 return nil
}

func processMessage(record events.SQSMessage) error {
 fmt.Printf("Processed message %s\n", record.Body)
 // TODO: Do interesting work based on the new message
 return nil
}

func main() {
 lambda.Start(handler)
}
```

Melden von Batch-Elementfehlern für Lambda-Funktionen mit einem Amazon-SQS-Auslöser

Das folgende Codebeispiel zeigt, wie eine partielle Batch-Antwort für Lambda-Funktionen implementiert wird, die Ereignisse aus einer SQS-Warteschlange empfangen. Die Funktion meldet die Batch-Elementfehler in der Antwort und signalisiert Lambda, diese Nachrichten später erneut zu versuchen.

SDK für Go V2

> **Note**
>
> Es gibt noch mehr dazu. GitHub Das vollständige Beispiel sowie eine Anleitung zum Einrichten und Ausführen finden Sie im Repository mit Serverless-Beispielen.

Melden von SQS-Batchelementfehlern mit Lambda unter Verwendung von Go.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
package main

import (
 "context"
 "encoding/json"
 "fmt"
 "github.com/aws/aws-lambda-go/events"
 "github.com/aws/aws-lambda-go/lambda"
)

func handler(ctx context.Context, sqsEvent events.SQSEvent) (map[string]interface{},
 error) {
 batchItemFailures := []map[string]interface{}{}

 for _, message := range sqsEvent.Records {

  if /* Your message processing condition here */ {
   batchItemFailures = append(batchItemFailures, map[string]interface{}
{"itemIdentifier": message.MessageId})
  }
 }

 sqsBatchResponse := map[string]interface{}{
  "batchItemFailures": batchItemFailures,
 }
 return sqsBatchResponse, nil
}

func main() {
 lambda.Start(handler)
}
```

Migrieren Sie auf die AWS SDK für Go Version 2

Minimale Go-Version

Das AWS SDK für Go erfordert eine Mindestversion von Go 1.20. Die neueste Version von Go kann auf der Download-Seite heruntergeladen werden. In der Versionsübersicht finden Sie weitere Informationen zu den einzelnen Go-Versionen und die für das Upgrade erforderlichen Informationen.

Modularisierung

Das AWS SDK für Go wurde aktualisiert, um die Vorteile der Go-Module zu nutzen, die in Go 1.13 zum Standard-Entwicklungsmodus wurden. Eine Reihe von Paketen, die vom SDK bereitgestellt werden, wurden modularisiert und werden unabhängig voneinander versioniert bzw. veröffentlicht. Diese Änderung ermöglicht eine verbesserte Modellierung von Anwendungsabhängigkeiten und ermöglicht es dem SDK, neue Features und Funktionen bereitzustellen, die der Versionierungsstrategie für Go-Module folgen.

In der folgenden Liste sind einige Go-Module aufgeführt, die vom SDK bereitgestellt werden:

Modul	Beschreibung
`github.com/aws/aws-sdk-go-v2`	Der SDK-Kern
`github.com/aws/aws-sdk-go-v2/config`	Gemeinsame Konfiguration wird geladen
`github.com/aws/aws-sdk-go-v2/credentials`	AWS Anbieter von Anmeldeinformationen
`github.com/aws/aws-sdk-go-v2/feature/ec2/imds`	Amazon EC2 Instance Metadata Service-Client

Die Service-Clients und übergeordneten Dienstprogramm-Module des SDK sind unter den folgenden Importpfaden verschachtelt:

Root importieren	Beschreibung
`github.com/aws/aws-sdk-go-v2/service/`	Service-Client-Module
`github.com/aws/aws-sdk-go-v2/feature/`	Hochrangige Dienstprogramme für Dienste wie den Amazon S3 Transfer Manager

Konfiguration wird geladen

Das Sitzungspaket und die zugehörige Funktionalität werden durch ein vereinfachtes Konfigurationssystem ersetzt, das im Konfigurationspaket bereitgestellt wird. Das `config` Paket ist ein separates Go-Modul und kann mit in die Abhängigkeiten Ihrer Anwendung aufgenommen werden`go get`.

```
go get github.com/aws/aws-sdk-go-v2/config
```

Die Session.new, Sitzung. NewSessionNewSessionWithOptions, und Session.Must müssen zur Konfiguration migriert werden. LoadDefaultConfig.

Das `config` Paket bietet mehrere Hilfsfunktionen, die dabei helfen, das programmgesteuerte Laden der gemeinsam genutzten Konfiguration zu überschreiben. Diesen Funktionsnamen wird ein Präfix vorangestellt, `With` gefolgt von einer Option, die sie überschreiben. Schauen wir uns einige Beispiele an, wie die Nutzung des `session` Pakets migriert werden kann.

Weitere Informationen zum Laden der gemeinsam genutzten Konfiguration finden Sie unterDas SDKs konfigurieren.

Beispiele

Migration von zu NewSession LoadDefaultConfig

Das folgende Beispiel zeigt, wie die Verwendung von `session.NewSession` ohne zusätzliche Argumentparameter migriert wird. `config.LoadDefaultConfig`

```
// V1 using NewSession

import "github.com/aws/aws-sdk-go/aws/session"
```

```
// ...

sess, err := session.NewSession()
if err != nil {
    // handle error
}
```

```
// V2 using LoadDefaultConfig

import "context"
import "github.com/aws/aws-sdk-go-v2/config"

// ...

cfg, err := config.LoadDefaultConfig(context.TODO())
if err != nil {
    // handle error
}
```

Migration von NewSession mit AWS.config-Optionen

Das Beispiel zeigt, wie das Überschreiben von `aws.Config` Werten beim Laden der Konfiguration migriert wird. Es können eine oder mehrere `config.With*` Hilfsfunktionen bereitgestellt werden, `config.LoadDefaultConfig` um die geladenen Konfigurationswerte zu überschreiben. In diesem Beispiel wird die AWS Region überschrieben, sodass die Konfiguration `us-west-2` verwendet wird. WithRegionHilfsfunktion.

```
// V1

import "github.com/aws/aws-sdk-go/aws"
import "github.com/aws/aws-sdk-go/aws/session"

// ...

sess, err := session.NewSession(aws.Config{
    Region: aws.String("us-west-2")
})
if err != nil {
    // handle error
}
```

```
// V2

import "context"
import "github.com/aws/aws-sdk-go-v2/config"

// ...

cfg, err := config.LoadDefaultConfig(context.TODO(),
    config.WithRegion("us-west-2"),
)
if err != nil {
    // handle error
}
```

Migration von NewSessionWithOptions

Dieses Beispiel zeigt, wie überschreibende Werte beim Laden der Konfiguration migriert werden. Es können keine oder mehr `config.With*` Hilfsfunktionen bereitgestellt werden, `config.LoadDefaultConfig` um die geladenen Konfigurationswerte zu überschreiben. In diesem Beispiel zeigen wir, wie das Zielprofil überschrieben wird, das beim Laden der gemeinsam genutzten AWS SDK-Konfiguration verwendet wird.

```
// V1

import "github.com/aws/aws-sdk-go/aws"
import "github.com/aws/aws-sdk-go/aws/session"

// ...

sess, err := session.NewSessionWithOptions(aws.Config{
    Profile: "my-application-profile"
})
if err != nil {
    // handle error
}
```

```
// V2

import "context"
import "github.com/aws/aws-sdk-go-v2/config"
```

```
// ...

cfg, err := config.LoadDefaultConfig(context.TODO(),
    config.WithSharedConfigProfile("my-application-profile"),
)
if err != nil {
    // handle error
}
```

Spott und `*iface`

Die darin enthaltenen `*iface` Pakete und Schnittstellen (z. B. s3iface.S3API) wurden entfernt. Diese Schnittstellendefinitionen sind nicht stabil, da sie jedes Mal kaputt gehen, wenn ein Dienst eine neue Operation hinzufügt.

Die Verwendung von `*iface` sollte durch vom Aufrufer definierte Schnittstellen mit Gültigkeitsbereich für die verwendeten Dienstoperationen ersetzt werden:

```
// V1

import "io"

import "github.com/aws/aws-sdk-go/service/s3"
import "github.com/aws/aws-sdk-go/service/s3/s3iface"

func GetObjectBytes(client s3iface.S3API, bucket, key string) ([]byte, error) {
    object, err := client.GetObject(&s3.GetObjectInput{
        Bucket: &bucket,
        Key:    &key,
    })
    if err != nil {
        return nil, err
    }
    defer object.Body.Close()

    return io.ReadAll(object.Body)
}
```

```
// V2

import "context"
```

```
import "io"

import "github.com/aws/aws-sdk-go-v2/service/s3"

type GetObjectAPIClient interface {
    GetObject(context.Context, *s3.GetObjectInput, ...func(*s3.Options))
 (*s3.GetObjectOutput, error)
}

func GetObjectBytes(ctx context.Context, client GetObjectAPIClient, bucket, key string)
 ([]byte, error) {
    object, err := client.GetObject(ctx, &s3.GetObjectInput{
        Bucket: &bucket,
        Key:    &key,
    })
    if err != nil {
        return nil, err
    }
    defer object.Body.Close()

    return io.ReadAll(object.Body)
}
```

Weitere Informationen finden Sie unter Unit-Tests mit dem AWS SDK für Go v2.

Anmeldeinformationen und Anbieter von Anmeldeinformationen

Das Paket aws/credentials und die zugehörigen Anmeldeinformationsanbieter wurden an den Speicherort für das Anmeldeinformationspaket verschoben. Das `credentials` Paket ist ein Go-Modul, mit dem Sie es abrufen können. `go get`

```
go get github.com/aws/aws-sdk-go-v2/credentials
```

In der Version AWS SDK für Go v2 werden die AWS Credential Providers aktualisiert, um eine konsistente Schnittstelle für das Abrufen von AWS Anmeldeinformationen bereitzustellen. Jeder Anbieter implementiert die aws. CredentialsProviderSchnittstelle, die eine `Retrieve` Methode definiert, die a zurückgibt(`aws.Credentials, error`). aws.Credentials, das dem Typ AWS SDK für Go v1 Credentials.Value entspricht.

Sie müssen Objekte mit aws umschließen. `aws.CredentialsProvider` CredentialsCacheum das Zwischenspeichern von Anmeldeinformationen zu ermöglichen. Sie verwenden, um ein

NewCredentialsCache`aws.CredentialsCache`Objekt zu konstruieren. Standardmäßig `config.LoadDefaultConfig` sind die von konfigurierten Anmeldeinformationen mit eingeschlossen`aws.CredentialsCache`.

In der folgenden Tabelle sind die Standortänderungen der AWS Anmeldeinformationsanbieter von AWS SDK für Go Version 1 zu Version 2 aufgeführt.

Name	V1-Import	V2 importieren
Anmeldeinformationen für Amazon EC2 IAM-Rollen	`github.com/aws/aws-sdk-go/aws/credentials/ec2rolecreds`	`github.com/aws/aws-sdk-go-v2/credentials/ec2rolecreds`
Anmeldeinformationen für Endgeräte	`github.com/aws/aws-sdk-go/aws/credentials/endpointcreds`	`github.com/aws/aws-sdk-go-v2/credentials/endpointcreds`
Anmeldeinformationen verarbeiten	`github.com/aws/aws-sdk-go/aws/credentials/processcreds`	`github.com/aws/aws-sdk-go-v2/credentials/processcreds`
AWS Security Token Service	`github.com/aws/aws-sdk-go/aws/credentials/stscreds`	`github.com/aws/aws-sdk-go-v2/credentials/stscreds`

Statische Anmeldeinformationen

Anwendungen, die Anmeldeinformationen verwenden. NewStaticCredentialsum statische Anmeldeinformationen programmgesteuert zu erstellen, müssen Anmeldeinformationen verwendet werden. NewStaticCredentialsProvider.

Beispiel

```
// V1

import "github.com/aws/aws-sdk-go/aws/credentials"

// ...
```

```
appCreds := credentials.NewStaticCredentials(accessKey, secretKey, sessionToken)
value, err := appCreds.Get()
if err != nil {
    // handle error
}
```

```
// V2

import "context"
import "github.com/aws/aws-sdk-go-v2/aws"
import "github.com/aws/aws-sdk-go-v2/credentials"

// ...

appCreds := aws.NewCredentialsCache(credentials.NewStaticCredentialsProvider(accessKey,
 secretKey, sessionToken))
value, err := appCreds.Retrieve(context.TODO())
if err != nil {
    // handle error
}
```

Anmeldeinformationen für Amazon EC2 IAM-Rollen

Sie müssen die Nutzung von NewCredentialsund NewCredentialsWithClientzur Verwendung von New migrieren.

Das `ec2rolecreds` Paket verwendet funktionale `ec2rolecreds.New` Optionen von ec2RoleCreds.Options als Eingabe, sodass Sie den zu verwendenden Amazon EC2 Instance Metadata Service-Client oder das Ablauffenster für Anmeldeinformationen überschreiben können.

Beispiel

```
// V1

import "github.com/aws/aws-sdk-go/aws/credentials/ec2rolecreds"

// ...

appCreds := ec2rolecreds.NewCredentials(sess)
value, err := appCreds.Get()
if err != nil {
```

```
    // handle error
}
```

```
// V2

import "context"
import "github.com/aws/aws-sdk-go-v2/aws"
import "github.com/aws/aws-sdk-go-v2/credentials/ec2rolecreds"

// ...

// New returns an object of a type that satisfies the aws.CredentialProvider interface
appCreds := aws.NewCredentialsCache(ec2rolecreds.New())
value, err := appCreds.Retrieve(context.TODO())
if err != nil {
    // handle error
}
```

Anmeldeinformationen für Endgeräte

Sie müssen die Verwendung von NewCredentialsClientund NewProviderClientzur Verwendung von New migrieren.

Die `New` Funktion des `endpointcreds` Pakets verwendet ein Zeichenkettenargument, das die URL eines HTTP- oder HTTPS-Endpunkts enthält, von dem Anmeldeinformationen abgerufen werden sollen, sowie funktionale Optionen von EndpointCreds.Options, um den Anbieter für Anmeldeinformationen zu ändern und bestimmte Konfigurationseinstellungen zu überschreiben.

Anmeldeinformationen verarbeiten

Sie müssen die Verwendung von NewCredentials, und migrieren NewCredentialsCommand, NewCredentialsTimeoutum NewProvideroder zu verwenden NewProviderCommand.

Die `NewProvider` Funktion des `processcreds` Pakets verwendet ein String-Argument, bei dem es sich um den Befehl handelt, der in der Shell der Host-Umgebung ausgeführt werden soll, sowie über die Funktionsoptionen von Options, um den Anbieter für Anmeldeinformationen zu ändern und bestimmte Konfigurationseinstellungen zu überschreiben.

`NewProviderCommand`verwendet eine Implementierung der NewCommandBuilderSchnittstelle, die komplexere Prozessbefehle definiert, die ein oder mehrere Befehlszeilenargumente benötigen oder bestimmte Anforderungen an die Ausführungsumgebung stellen können.

DefaultNewCommandBuilderimplementiert diese Schnittstelle und definiert einen Befehlsgenerator für einen Prozess, der mehrere Befehlszeilenargumente erfordert.

Beispiel

```
// V1

import "github.com/aws/aws-sdk-go/aws/credentials/processcreds"

// ...

appCreds := processcreds.NewCredentials("/path/to/command")
value, err := appCreds.Get()
if err != nil {
    // handle error
}
```

```
// V2

import "context"
import "github.com/aws/aws-sdk-go-v2/aws"
import "github.com/aws/aws-sdk-go-v2/credentials/processcreds"

// ...

appCreds := aws.NewCredentialsCache(processcreds.NewProvider("/path/to/command"))
value, err := appCreds.Retrieve(context.TODO())
if err != nil {
    // handle error
}
```

AWS Security Token Service Erweitern Sie im angezeigten Detailbereich die Option

AssumeRole

Sie müssen die Verwendung von NewCredentialsund NewCredentialsWithClientzur Verwendung migrieren. NewAssumeRoleProvider

Die `NewAssumeRoleProvider` Funktion des `stscreds` Pakets muss mit einem STS.Client aufgerufen werden, und der AWS Identity and Access Management Rollen-ARN muss aus den

konfigurierten Anmeldeinformationen `sts.Client` des angegebenen Benutzers übernommen werden. Sie können auch eine Reihe von Funktionsoptionen angeben AssumeRoleOptions, um andere optionale Einstellungen des Anbieters zu ändern.

Beispiel

```
// V1

import "github.com/aws/aws-sdk-go/aws/credentials/stscreds"

// ...

appCreds := stscreds.NewCredentials(sess, "arn:aws:iam::123456789012:role/demo")
value, err := appCreds.Get()
if err != nil {
    // handle error
}
```

```
// V2

import "context"
import "github.com/aws/aws-sdk-go-v2/credentials/stscreds"
import "github.com/aws/aws-sdk-go-v2/service/sts"

// ...

client := sts.NewFromConfig(cfg)

appCreds := stscreds.NewAssumeRoleProvider(client, "arn:aws:iam::123456789012:role/
demo")
value, err := appCreds.Retrieve(context.TODO())
if err != nil {
    // handle error
}
```

AssumeRoleWithWebIdentity

Sie müssen die Verwendung von NewWebIdentityCredentials, und migrieren NewWebIdentityRoleProvider, NewWebIdentityRoleProviderWithTokenum zu verwenden NewWebIdentityRoleProvider.

Die `NewWebIdentityRoleProvider` Funktion des `stscreds` Pakets muss mit einem STS.Client aufgerufen werden, und der AWS Identity and Access Management Rollen-ARN muss unter Verwendung der konfigurierten Anmeldeinformationen `sts.Client` des angegebenen Benutzers und einer Implementierung von a IdentityTokenRetrieverzur Bereitstellung des OAuth 2.0- oder OpenID Connect-ID-Tokens übernommen werden. IdentityTokenFileist ein`IdentityTokenRetriever`, die verwendet werden kann, um das Web-Identitätstoken aus einer Datei bereitzustellen, die sich im Host-Dateisystem der Anwendung befindet. Sie können auch eine Reihe von Funktionsoptionen angeben, WebIdentityRoleOptionsum andere optionale Einstellungen für den Anbieter zu ändern.

Beispiel

```
// V1

import "github.com/aws/aws-sdk-go/aws/credentials/stscreds"

// ...

appCreds := stscreds.NewWebIdentityRoleProvider(sess, "arn:aws:iam::123456789012:role/
demo", "sessionName", "/path/to/token")
value, err := appCreds.Get()
if err != nil {
    // handle error
}
```

```
// V2

import "context"
import "github.com/aws/aws-sdk-go-v2/aws"
import "github.com/aws/aws-sdk-go-v2/credentials/stscreds"
import "github.com/aws/aws-sdk-go-v2/service/sts"

// ...

client := sts.NewFromConfig(cfg)

appCreds := aws.NewCredentialsCache(stscreds.NewWebIdentityRoleProvider(
        client,
        "arn:aws:iam::123456789012:role/demo",
        stscreds.IdentityTokenFile("/path/to/file"),
        func(o *stscreds.WebIdentityRoleOptions) {
```

```
            o.RoleSessionName = "sessionName"
        }))
value, err := appCreds.Retrieve(context.TODO())
if err != nil {
    // handle error
}
```

Kunden bedienen

AWS SDK für Go stellt Service-Client-Module bereit, die unter dem `github.com/aws/aws-sdk-go-v2/service` Importpfad verschachtelt sind. Jeder Service-Client ist in einem Go-Paket enthalten, das die eindeutige Kennung jedes Dienstes verwendet. Die folgende Tabelle enthält einige Beispiele für Service-Importpfade in der AWS SDK für Go.

Service-Name	V1-Importpfad	V2-Importpfad
Amazon S3	`github.com/aws/aws-sdk-go/service/s3`	`github.com/aws/aws-sdk-go-v2/service/s3`
Amazon-DynamoDB	`github.com/aws/aws-sdk-go/service/dynamodb`	`github.com/aws/aws-sdk-go-v2/service/dynamodb`
CloudWatch Amazon-Protokolle	`github.com/aws/aws-sdk-go/service/cloudwatchlogs`	`github.com/aws/aws-sdk-go-v2/service/cloudwatchlogs`

Jedes Service-Client-Paket ist ein unabhängig versioniertes Go-Modul. Um den Service-Client als Abhängigkeit von Ihrer Anwendung hinzuzufügen, verwenden Sie den `go get` Befehl mit dem Importpfad des Dienstes. Um beispielsweise den Amazon S3 S3-Client zu Ihren Abhängigkeiten hinzuzufügen, verwenden Sie

```
go get github.com/aws/aws-sdk-go-v2/service/s3
```

Aufbau eines Kunden

Sie können Clients erstellen, AWS SDK für Go indem Sie entweder die `New` oder die `NewFromConfig` Konstruktorfunktionen im Paket des Clients verwenden. Bei der Migration von AWS

SDK für Go Version 1 empfehlen wir, die `NewFromConfig` Variante zu verwenden, die einen neuen Service-Client mit Werten aus einem zurückgibt. `aws.Config` Der `aws.Config` Wert wurde beim Laden der gemeinsam genutzten SDK-Konfiguration mit `config.LoadDefaultConfig` erstellt. Einzelheiten zum Erstellen von Service-Clients finden Sie unterVerwenden Sie die AWS SDK für Go Version 2 mit AWS Diensten.

Beispiel 1

```
// V1

import "github.com/aws/aws-sdk-go/aws/session"
import "github.com/aws/aws-sdk-go/service/s3"

// ...

sess, err := session.NewSession()
if err != nil {
    // handle error
}

client := s3.New(sess)
```

```
// V2

import "context"
import "github.com/aws/aws-sdk-go-v2/config"
import "github.com/aws/aws-sdk-go-v2/service/s3"

// ...

cfg, err := config.LoadDefaultConfig(context.TODO())
if err != nil {
    // handle error
}

client := s3.NewFromConfig(cfg)
```

Beispiel 2: Client-Einstellungen überschreiben

```
// V1
```

```
import "github.com/aws/aws-sdk-go/aws"
import "github.com/aws/aws-sdk-go/aws/session"
import "github.com/aws/aws-sdk-go/service/s3"

// ...

sess, err := session.NewSession()
if err != nil {
    // handle error
}

client := s3.New(sess, &aws.Config{
    Region: aws.String("us-west-2"),
})
```

```
// V2

import "context"
import "github.com/aws/aws-sdk-go-v2/config"
import "github.com/aws/aws-sdk-go-v2/service/s3"

// ...

cfg, err := config.LoadDefaultConfig(context.TODO())
if err != nil {
    // handle error
}

client := s3.NewFromConfig(cfg, func(o *s3.Options) {
    o.Region = "us-west-2"
})
```

Endpunkte

Das Endpoints-Paket ist nicht mehr in der vorhanden. AWS SDK für Go Jeder Service-Client bettet jetzt seine erforderlichen AWS Endpunkt-Metadaten in das Client-Paket ein. Dadurch wird die gesamte Binärgröße kompilierter Anwendungen reduziert, da keine Endpunktmetadaten für Dienste mehr enthalten sind, die von Ihrer Anwendung nicht verwendet werden.

Darüber hinaus bietet jeder Dienst jetzt seine eigene Schnittstelle für die Endpunktauflösung in`EndpointResolverV2`. Jede API verwendet einen eindeutigen Satz von Parametern für einen

Dienst`EndpointParameters`, dessen Werte vom SDK beim Aufrufen eines Vorgangs von verschiedenen Orten abgerufen werden.

Standardmäßig verwenden Service-Clients ihre konfigurierte AWS Region, um den Service-Endpunkt für die Zielregion aufzulösen. Wenn für Ihre Anwendung ein benutzerdefinierter Endpunkt erforderlich ist, können Sie ein benutzerdefiniertes Verhalten für das `EndpointResolverV2` Feld in der `aws.Config` Struktur angeben. Wenn Ihre Anwendung einen benutzerdefinierten Endpoints.Resolver implementiert, müssen Sie ihn migrieren, damit er dieser neuen Schnittstelle pro Dienst entspricht.

Weitere Informationen zu Endpunkten und zur Implementierung eines benutzerdefinierten Resolvers finden Sie unter. Client-Endpunkte konfigurieren

Authentifizierung

Das AWS SDK für Go unterstützt ein erweitertes Authentifizierungsverhalten, das die Verwendung neuerer AWS Servicefunktionen wie Codecatalyst und S3 Express One Zone ermöglicht. Darüber hinaus kann dieses Verhalten für jeden einzelnen Client angepasst werden.

API-Operationen aufrufen

Die Anzahl der Betriebsmethoden für Service-Clients wurde erheblich reduziert. Die `<OperationName>` Methoden `<OperationName>Request<OperationName>WithContext`, und wurden alle zu einer einzigen Betriebsmethode zusammengefasst`<OperationName>`.

Beispiel

Das folgende Beispiel zeigt, wie Aufrufe des Amazon S3 PutObject S3-Vorgangs von AWS SDK für Go Version 1 auf Version 2 migriert würden.

```
// V1

import "context"
import "github.com/aws/aws-sdk-go/service/s3"

// ...

client := s3.New(sess)

// Pattern 1
```

```
output, err := client.PutObject(&s3.PutObjectInput{
    // input parameters
})

// Pattern 2
output, err := client.PutObjectWithContext(context.TODO(), &s3.PutObjectInput{
    // input parameters
})

// Pattern 3
req, output := client.PutObjectRequest(context.TODO(), &s3.PutObjectInput{
    // input parameters
})
err := req.Send()
```

```
// V2

import "context"
import "github.com/aws/aws-sdk-go-v2/service/s3"

// ...

client := s3.NewFromConfig(cfg)

output, err := client.PutObject(context.TODO(), &s3.PutObjectInput{
    // input parameters
})
```

Service-Datentypen

Die Eingabe- und Ausgabetypen der obersten Ebene eines Vorgangs finden Sie im Service-Client-Paket. Der Eingabe- und Ausgabetyp für eine bestimmte Operation folgt dem Muster von `<OperationName>Input` und`<OperationName>Output`, wobei der Name der Operation `OperationName` ist, die Sie aufrufen. Beispielsweise sind die Eingabe- und Ausgabeform für den Amazon S3 PutObject S3-Vorgang PutObjectOutputjeweils PutObjectInputund.

Alle anderen Servicedatentypen, mit Ausnahme der Eingabe- und Ausgabetypen, wurden in das `types` Paket migriert, das sich unter der Hierarchie des Importpfads für Service-Client-Pakete befindet. Zum Beispiel der S3. AccessControlPolicyDer Typ befindet sich jetzt unter Typen. AccessControlPolicy.

Aufzählungswerte

Das SDK bietet jetzt eine typisierte Oberfläche für alle API-Aufzählungsfelder. Anstatt einen Zeichenkettenliteralwert zu verwenden, der aus der Referenzdokumentation der Service-API kopiert wurde, können Sie jetzt einen der konkreten Typen verwenden, die im Paket des Service-Clients enthalten sind. `types` Beispielsweise können Sie für den Amazon S3 PutObjectInput S3-Vorgang eine ACL bereitstellen, die auf ein Objekt angewendet wird. In AWS SDK für Go Version 1 war dieser Parameter ein `*string` Typ. In der AWS SDK für Go ist dieser Parameter jetzt ein Typ. ObjectCannedACL. Das `types` Paket stellt generierte Konstanten für die gültigen Aufzählungswerte bereit, die diesem Feld zugewiesen werden können. Zum Beispiel Typen. ObjectCannedACLPrivateist die Konstante für den gespeicherten ACL-Wert „privat“. Dieser Wert kann anstelle der Verwaltung von Zeichenkettenkonstanten in Ihrer Anwendung verwendet werden.

Zeiger-Parameter

In AWS SDK für Go Version 1 mussten Zeigerreferenzen für alle Eingabeparameter an Serviceoperationen übergeben werden. AWS SDK für Go Version 2 hat die Erfahrung mit den meisten Diensten vereinfacht, da die Notwendigkeit entfällt, Eingabewerte nach Möglichkeit als Zeiger zu übergeben. Diese Änderung bedeutet, dass Ihre Anwendung bei vielen Service-Client-Vorgängen nicht mehr Zeigerreferenzen für die folgenden Typen übergeben muss:`uint8`,`uint16`,`uint32`,`int8`,`int16`,,`int32`, `float32float64`,`bool`. In ähnlicher Weise wurden die Elementtypen Slice und Map entsprechend aktualisiert, um zu berücksichtigen, ob ihre Elemente als Zeigerreferenzen übergeben werden müssen.

Das aws-Paket enthält Hilfsfunktionen zum Erstellen von Zeigern für die integrierten Go-Typen. Diese Helfer sollten verwendet werden, um die Erstellung von Zeigertypen für diese Go-Typen einfacher zu handhaben. In ähnlicher Weise werden Hilfsmethoden bereitgestellt, mit denen Zeigerwerte für diese Typen sicher dereferenziert werden können. Die AWS.String-Funktion konvertiert beispielsweise aus ⇒. `string` `*string` Umgekehrt die aws. ToStringkonvertiert von `*string` ⇒`string`. Wenn Sie Ihre Anwendung von Version AWS SDK für Go 1 auf Version 2 aktualisieren, müssen Sie die Verwendung der Hilfsprogramme für die Konvertierung von Zeigertypen auf Varianten ohne Zeiger migrieren. Zum Beispiel aws. StringValuemuss aktualisiert werden auf`aws.ToString`.

Arten von Fehlern

Das AWS SDK für Go nutzt die in Go 1.13 eingeführte Error-Wrapping-Funktionalität voll aus. Dienste, die Fehlerantworten modellieren, verfügen im `types` Paket ihres Clients über generierte Typen, mit denen getestet werden kann, ob ein Client-Betriebsfehler durch einen dieser Typen

verursacht wurde. Beispielsweise kann der Amazon S3 `GetObject` S3-Vorgang einen `NoSuchKey` Fehler zurückgeben, wenn versucht wird, einen Objektschlüssel abzurufen, der nicht existiert. Sie können Errors.as verwenden, um zu testen, ob es sich bei dem zurückgegebenen Operationsfehler um einen Typ handelt. NoSuchKeyFehler. Falls ein Dienst keinen bestimmten Typ für einen Fehler modelliert, können Sie den Smithy verwenden. APIErrorSchnittstellentyp zur Überprüfung des zurückgegebenen Fehlercodes und der Meldung des Dienstes. Diese Funktionalität ersetzt awserr.Error und die anderen awserr-Funktionen aus Version 1. AWS SDK für Go Weitere Informationen zur Behandlung von Fehlern finden Sie unter. Umgang mit Fehlern in der AWS SDK für Go V2

Beispiel

```
// V1

import "github.com/aws/aws-sdk-go/aws/awserr"
import "github.com/aws/aws-sdk-go/service/s3"

// ...

client := s3.New(sess)

output, err := s3.GetObject(&s3.GetObjectInput{
    // input parameters
})
if err != nil {
    if awsErr, ok := err.(awserr.Error); ok {
        if awsErr.Code() == "NoSuchKey" {
            // handle NoSuchKey
        } else {
            // handle other codes
        }
        return
    }
    // handle a error
}
```

```
// V2

import "context"
import "github.com/aws/aws-sdk-go-v2/service/s3"
import "github.com/aws/aws-sdk-go-v2/service/s3/types"
```

```
import "github.com/aws/smithy-go"

// ...

client := s3.NewFromConfig(cfg)

output, err := client.GetObject(context.TODO(), &s3.GetObjectInput{
    // input parameters
})
if err != nil {
    var nsk *types.NoSuchKey
    if errors.As(err, &nsk) {
        // handle NoSuchKey error
        return
    }
    var apiErr smithy.APIError
    if errors.As(err, &apiErr) {
        code := apiErr.ErrorCode()
        message := apiErr.ErrorMessage()
        // handle error code
        return
    }
    // handle error
    return
}
```

Paginatoren

Paginatoren für Dienstoperationen werden nicht mehr als Methoden auf dem Service-Client aufgerufen. Um einen Paginator für eine Operation zu verwenden, müssen Sie mit einer der Paginator-Konstruktormethoden einen Paginator für eine Operation erstellen. Um beispielsweise Paginate über die Amazon S3 `ListObjectsV2` S3-Operation zu verwenden, müssen Sie den zugehörigen Paginator mithilfe der S3 erstellen. NewListObjectsV2-Paginator. Dieser Konstruktor gibt einen ListObjectsV2Paginator zurück, der die Methoden bereitstellt und bestimmt`HasMorePages`, ob weitere Seiten abgerufen werden müssen, und `NextPage` der die Operation zum Abrufen der nächsten Seite aufruft. Weitere Informationen zur Verwendung der SDK-Paginatoren finden Sie unter. Verwendung von Operation Paginators

Schauen wir uns ein Beispiel für die Migration von einem AWS SDK für Go v1-Paginator zum v2-Äquivalent an. AWS SDK für Go

Beispiel

```
// V1

import "fmt"
import "github.com/aws/aws-sdk-go/service/s3"

// ...

client := s3.New(sess)

params := &s3.ListObjectsV2Input{
    // input parameters
}

totalObjects := 0
err := client.ListObjectsV2Pages(params, func(output *s3.ListObjectsV2Output, lastPage
 bool) bool {
    totalObjects += len(output.Contents)
    return !lastPage
})
if err != nil {
    // handle error
}
fmt.Println("total objects:", totalObjects)
```

```
// V2

import "context"
import "fmt"
import "github.com/aws/aws-sdk-go-v2/service/s3"

// ...

client := s3.NewFromConfig(cfg)

params := &s3.ListObjectsV2Input{
    // input parameters
}

totalObjects := 0
paginator := s3.NewListObjectsV2Paginator(client, params)
for paginator.HasMorePages() {
```

```
    output, err := paginator.NextPage(context.TODO())
    if err != nil {
        // handle error
    }
    totalObjects += len(output.Contents)
}
fmt.Println("total objects:", totalObjects)
```

Waiter

Kellner für Serviceoperationen werden nicht mehr als Methoden auf dem Service-Client aufgerufen. Um einen Kellner zu verwenden, erstellen Sie zunächst den gewünschten Kellnertyp und rufen dann die Methode wait auf. Um beispielsweise darauf zu warten, dass ein Amazon S3 S3-Bucket existiert, müssen Sie einen `BucketExists` Kellner einrichten. Verwenden Sie den S3. NewBucketExistsWaiterKonstruktor, um ein S3 zu erstellen. BucketExistsWaiter. Das `s3.BucketExistsWaiter` bietet eine `Wait` Methode, mit der Sie warten können, bis ein Bucket verfügbar ist.

Vorsignierte Anfragen

Das V1-SDK unterstützte technisch gesehen die Vorsignierung aller AWS SDK-Operationen, dies entspricht jedoch nicht genau dem, was tatsächlich auf der Serviceebene unterstützt wird (und in der Realität unterstützen die meisten AWS Dienstoperationen das Vorsignieren nicht).

AWS SDK für Go behebt dieses Problem, indem es spezifische `PresignClient` Implementierungen in Servicepaketen mit spezifischen APIs , für die unterstützten vorsignierbaren Vorgängen verfügbar macht.

Hinweis: Wenn bei einem Dienst die Unterstützung für die Vorsignierung eines Vorgangs fehlt, den Sie erfolgreich in SDK v1 verwendet haben, teilen Sie uns dies bitte mit, indem Sie ein Problem unter melden. GitHub

Wird von Presign verwendet und PresignRequestmuss auf die Verwendung von dienstspezifischen Presigning-Clients umgestellt werden.

Das folgende Beispiel zeigt, wie die Vorsignierung einer S3-Anfrage migriert wird: GetObject

```
// V1

import (
```

```
    "fmt"
    "time"

    "github.com/aws/aws-sdk-go/aws"
    "github.com/aws/aws-sdk-go/aws/session"
    "github.com/aws/aws-sdk-go/service/s3"
)

func main() {
    sess := session.Must(session.NewSessionWithOptions(session.Options{
        SharedConfigState: session.SharedConfigEnable,
    }))

    svc := s3.New(sess)
    req, _ := svc.GetObjectRequest(&s3.GetObjectInput{
        Bucket: aws.String("amzn-s3-demo-bucket"),
        Key:    aws.String("key"),
    })

    // pattern 1
    url1, err := req.Presign(20 * time.Minute)
    if err != nil {
        panic(err)
    }
    fmt.Println(url1)

    // pattern 2
    url2, header, err := req.PresignRequest(20 * time.Minute)
    if err != nil {
        panic(err)
    }
    fmt.Println(url2, header)
}
```

```
// V2

import (
    "context"
    "fmt"
    "time"

    "github.com/aws/aws-sdk-go-v2/aws"
    "github.com/aws/aws-sdk-go-v2/config"
```

```
    "github.com/aws/aws-sdk-go-v2/service/s3"
)

func main() {
    cfg, err := config.LoadDefaultConfig(context.Background())
    if err != nil {
        panic(err)
    }

    svc := s3.NewPresignClient(s3.NewFromConfig(cfg))
    req, err := svc.PresignGetObject(context.Background(), &s3.GetObjectInput{
        Bucket: aws.String("amzn-s3-demo-bucket"),
        Key:    aws.String("key"),
    }, func(o *s3.PresignOptions) {
        o.Expires = 20 * time.Minute
    })
    if err != nil {
        panic(err)
    }

    fmt.Println(req.Method, req.URL, req.SignedHeader)
}
```

Anpassung der Anfrage

Die monolithische Request.Request-API wurde neu unterteilt.

Operation Eingabe/Ausgabe

Die undurchsichtigen `Request` Felder `Params` und`Data`, die die Eingabe- bzw. Ausgabestrukturen der Operation enthalten, sind nun innerhalb bestimmter Middleware-Phasen als Eingabe/Ausgabe zugänglich:

Anforderungshandler, die auf Middleware verweisen `Request.Params` und `Request.Data` zu dieser migriert werden müssen.

migrieren **`Params`**

```
// V1

import (
```

```
    "github.com/aws/aws-sdk-go/aws"
    "github.com/aws/aws-sdk-go/aws/request"
    "github.com/aws/aws-sdk-go/aws/session"
    "github.com/aws/aws-sdk-go/service/s3"
)

func withPutObjectDefaultACL(acl string) request.Option {
    return func(r *request.Request) {
        in, ok := r.Params.(*s3.PutObjectInput)
        if !ok {
            return
        }

        if in.ACL == nil {
            in.ACL = aws.String(acl)
        }
        r.Params = in
    }
}

func main() {
    sess := session.Must(session.NewSession())

 sess.Handlers.Validate.PushBack(withPutObjectDefaultACL(s3.ObjectCannedACLBucketOwnerFullContr

    // ...
}
```

```
// V2

import (
    "context"

    "github.com/aws/aws-sdk-go-v2/service/s3"
    "github.com/aws/aws-sdk-go-v2/service/s3/types"
    "github.com/aws/smithy-go/middleware"
    smithyhttp "github.com/aws/smithy-go/transport/http"
)

type withPutObjectDefaultACL struct {
    acl types.ObjectCannedACL
}
```

```
// implements middleware.InitializeMiddleware, which runs BEFORE a request has
// been serialized and can act on the operation input
var _ middleware.InitializeMiddleware = (*withPutObjectDefaultACL)(nil)

func (*withPutObjectDefaultACL) ID() string {
    return "withPutObjectDefaultACL"
}

func (m *withPutObjectDefaultACL) HandleInitialize(ctx context.Context, in
 middleware.InitializeInput, next middleware.InitializeHandler) (
    out middleware.InitializeOutput, metadata middleware.Metadata, err error,
) {
    input, ok := in.Parameters.(*s3.PutObjectInput)
    if !ok {
        return next.HandleInitialize(ctx, in)
    }

    if len(input.ACL) == 0 {
        input.ACL = m.acl
    }
    in.Parameters = input
    return next.HandleInitialize(ctx, in)
}

// create a helper function to simplify instrumentation of our middleware
func WithPutObjectDefaultACL(acl types.ObjectCannedACL) func (*s3.Options) {
    return func(o *s3.Options) {
        o.APIOptions = append(o.APIOptions, func (s *middleware.Stack) error {
            return s.Initialize.Add(&withPutObjectDefaultACL{acl: acl},
 middleware.After)
        })
    }
}

func main() {
    cfg, err := config.LoadDefaultConfig(context.Background())
    if err != nil {
        // ...
    }

    svc := s3.NewFromConfig(cfg,
 WithPutObjectDefaultACL(types.ObjectCannedACLBucketOwnerFullControl))
    // ...
```

```
}
```

migrieren **`Data`**

```
// V1

import (
    "github.com/aws/aws-sdk-go/aws"
    "github.com/aws/aws-sdk-go/aws/request"
    "github.com/aws/aws-sdk-go/aws/session"
    "github.com/aws/aws-sdk-go/service/s3"
)

func readPutObjectOutput(r *request.Request) {
        output, ok := r.Data.(*s3.PutObjectOutput)
        if !ok {
            return
        }

        // ...
    }
}

func main() {
    sess := session.Must(session.NewSession())
    sess.Handlers.Unmarshal.PushBack(readPutObjectOutput)

    svc := s3.New(sess)
    // ...
}
```

```
// V2

import (
    "context"

    "github.com/aws/aws-sdk-go-v2/config"
    "github.com/aws/aws-sdk-go-v2/service/s3"
    "github.com/aws/smithy-go/middleware"
    smithyhttp "github.com/aws/smithy-go/transport/http"
)

type readPutObjectOutput struct{}
```

```
var _ middleware.DeserializeMiddleware = (*readPutObjectOutput)(nil)

func (*readPutObjectOutput) ID() string {
    return "readPutObjectOutput"
}

func (*readPutObjectOutput) HandleDeserialize(ctx context.Context, in
 middleware.DeserializeInput, next middleware.DeserializeHandler) (
    out middleware.DeserializeOutput, metadata middleware.Metadata, err error,
) {
    out, metadata, err = next.HandleDeserialize(ctx, in)
    if err != nil {
        // ...
    }

    output, ok := in.Parameters.(*s3.PutObjectOutput)
    if !ok {
        return out, metadata, err
    }

    // inspect output...

    return out, metadata, err
}

func WithReadPutObjectOutput(o *s3.Options) {
    o.APIOptions = append(o.APIOptions, func (s *middleware.Stack) error {
        return s.Initialize.Add(&withReadPutObjectOutput{}, middleware.Before)
    })
}

func main() {
    cfg, err := config.LoadDefaultConfig(context.Background())
    if err != nil {
        // ...
    }

    svc := s3.NewFromConfig(cfg, WithReadPutObjectOutput)
    // ...
}
```

HTTP-Anfrage/Antwort

Die `HTTPResponse` Felder `HTTPRequest` und von `Request` werden jetzt in bestimmten Middleware-Phasen verfügbar gemacht. Da Middleware transportunabhängig ist, müssen Sie eine Typzusicherung für die Middleware-Eingabe oder -Ausgabe durchführen, um die zugrunde liegende HTTP-Anfrage oder -Antwort aufzudecken.

Anforderungshandler, die auf Middleware verweisen und zu dieser migriert werden müssen. `Request.HTTPRequest` `Request.HTTPResponse`

migrieren **`HTTPRequest`**

```
// V1

import (
    "github.com/aws/aws-sdk-go/aws/request"
    "github.com/aws/aws-sdk-go/aws/session"
)

func withHeader(header, val string) request.Option {
    return func(r *request.Request) {
        request.HTTPRequest.Header.Set(header, val)
    }
}

func main() {
    sess := session.Must(session.NewSession())
    sess.Handlers.Build.PushBack(withHeader("x-user-header", "..."))

    svc := s3.New(sess)
    // ...
}
```

```
// V2

import (
    "context"
    "fmt"

    "github.com/aws/aws-sdk-go-v2/config"
    "github.com/aws/aws-sdk-go-v2/service/s3"
    "github.com/aws/smithy-go/middleware"
```

```
    smithyhttp "github.com/aws/smithy-go/transport/http"
)

type withHeader struct {
    header, val string
}

// implements middleware.BuildMiddleware, which runs AFTER a request has been
// serialized and can operate on the transport request
var _ middleware.BuildMiddleware = (*withHeader)(nil)

func (*withHeader) ID() string {
    return "withHeader"
}

func (m *withHeader) HandleBuild(ctx context.Context, in middleware.BuildInput, next
 middleware.BuildHandler) (
    out middleware.BuildOutput, metadata middleware.Metadata, err error,
) {
    req, ok := in.Request.(*smithyhttp.Request)
    if !ok {
        return out, metadata, fmt.Errorf("unrecognized transport type %T", in.Request)
    }

    req.Header.Set(m.header, m.val)
    return next.HandleBuild(ctx, in)
}

func WithHeader(header, val string) func (*s3.Options) {
    return func(o *s3.Options) {
        o.APIOptions = append(o.APIOptions, func (s *middleware.Stack) error {
            return s.Build.Add(&withHeader{
                header: header,
                val: val,
            }, middleware.After)
        })
    }
}

func main() {
    cfg, err := config.LoadDefaultConfig(context.Background())
    if err != nil {
        // ...
    }
```

```
    svc := s3.NewFromConfig(cfg, WithHeader("x-user-header", "..."))
    // ...
}
```

Handler-Phasen

SDK v2-Middleware-Phasen sind der Nachfolger der v1-Handler-Phasen.

Die folgende Tabelle enthält eine grobe Zuordnung der v1-Handler-Phasen zu ihrer entsprechenden Position innerhalb des V2-Middleware-Stacks:

Name des v1-Handlers	v2-Middleware-Phase
Validieren	Initialisieren
Entwicklung	Serialisieren
Sign	Finalisieren
Senden	n/a (1)
ValidateResponse	Deserialisieren
Unmarschall	Deserialisieren
UnmarshalMetadata	Deserialisieren
UnmarshalError	Deserialisieren
Erneut versuchen	Finalisieren, nach `"Retry"` Middleware (2)
AfterRetry	Finalisieren, vor der `"Retry"` Middleware, nach- (2,3) `next.HandleFinalize()`
CompleteAttempt	Finalisieren, Ende des Schritts
Complete	Initialisieren, Beginn des Schritts, Post- `next.HandleInitialize()` (3)

(1) Die `Send` Phase in Version 1 ist praktisch der umschlossene HTTP-Client-Round-Trip in Version 2. Dieses Verhalten wird durch das `HTTPClient` Feld in den Client-Optionen gesteuert.

(2) Jede Middleware nach der `"Retry"` Middleware im Schritt Finalize wird Teil der Wiederholungsschleife.

(3) Der Middleware-"Stack“ ist während des Betriebs in eine mehrfach dekorierte Handler-Funktion integriert. Jeder Handler ist dafür verantwortlich, den nächsten in der Kette aufzurufen. Dies bedeutet implizit, dass ein Middleware-Schritt auch Maßnahmen ergreifen kann, NACHDEM sein nächster Schritt aufgerufen wurde.

Für den Schritt Initialisieren, der sich ganz oben im Stack befindet, bedeutet dies, dass Middlewares initialisieren, die nach dem Aufrufen des nächsten Handlers eine Aktion ausführen, effektiv am Ende der Anfrage ausgeführt werden:

```
// V2

import (
    "context"

    "github.com/aws/smithy-go/middleware"
)

type onComplete struct{}

var _ middleware.InitializeMiddleware = (*onComplete)(nil)

func (*onComplete) ID() string {
    return "onComplete"
}

func (*onComplete) HandleInitialize(ctx context.Context, in middleware.InitializeInput,
 next middleware.InitializeHandler) (
    out middleware.InitializeOutput, metadata middleware.Metadata, err error,
) {
    out, metadata, err = next.HandleInitialize(ctx, in)

    // the entire operation was invoked above - the deserialized response is
    // available opaquely in out.Result, run post-op actions here...

    return out, metadata, err
}
```

Features

Metadatenservice für EC2 Amazon-Instances

Der AWS SDK für Go stellt einen Amazon EC2 Instance Metadata Service (IMDS) -Client bereit, mit dem Sie das lokale IMDS abfragen können, wenn Sie Ihre Anwendung auf einer EC2 Amazon-Instance ausführen. Der IMDS-Client ist ein separates Go-Modul, das Ihrer Anwendung hinzugefügt werden kann, indem Sie

```
go get github.com/aws/aws-sdk-go-v2/feature/ec2/imds
```

Der Client-Konstruktor und die Methodenoperationen wurden aktualisiert, sodass sie dem Design der anderen SDK-Dienstclients entsprechen.

Beispiel

```
// V1

import "github.com/aws/aws-sdk-go/aws/ec2metadata"

// ...

client := ec2metadata.New(sess)

region, err := client.Region()
if err != nil {
    // handle error
}
```

```
// V2

import "context"
import "github.com/aws/aws-sdk-go-v2/feature/ec2/imds"

// ...

client := imds.NewFromConfig(cfg)

region, err := client.GetRegion(context.TODO())
if err != nil {
```

```
    // handle error
}
```

Amazon S3 Transfer-Manager

Der Amazon S3 S3-Transfermanager ist für die gleichzeitige Verwaltung von Uploads und Downloads von Objekten verfügbar. Dieses Paket befindet sich in einem Go-Modul außerhalb des Importpfads des Service-Clients. Dieses Modul kann abgerufen werden, indem Sie `go get github.com/aws/aws-sdk-go-v2/feature/s3/manager`

s3. NewUploaderund s3. NewUploaderWithClientwurden durch den Konstruktor-Methodenmanager ersetzt. NewUploaderzum Erstellen eines Upload-Manager-Clients.

s3. NewDownloaderund s3. NewDownloaderWithClientwurden durch einen einzigen Konstruktor-Methodenmanager ersetzt. NewDownloaderzum Erstellen eines Download-Manager-Clients.

Amazon CloudFront Signing Utilities

Das AWS SDK für Go stellt CloudFront Amazon-Signaturdienstprogramme in einem Go-Modul außerhalb des Importpfads des Service-Clients bereit. Dieses Modul kann mithilfe von abgerufen werden`go get`.

```
go get github.com/aws/aws-sdk-go-v2/feature/cloudfront/sign
```

Amazon S3 Encryption Client

Ab sofort AWS SDK für Go ist der Amazon S3 S3-Verschlüsselungsclient ein separates Modul unter AWS Crypto Tools. Die neueste Version des S3-Verschlüsselungsclients für Go, 3.x, ist jetzt bei https://github.com/aws/amazon-s3 - verfügbar. encryption-client-go Dieses Modul kann wie folgt abgerufen werden: `go get`

```
go get github.com/aws/amazon-s3-encryption-client-go/v3
```

Die separaten Versionen `EncryptionClient` (v1, v2) und `DecryptionClient` (v1, v2) APIs wurden durch einen einzigen Client, S3 EncryptionClient V3, ersetzt, der sowohl Verschlüsselungs- als auch Entschlüsselungsfunktionen bietet.

Wie bei anderen Service-Clients wurde AWS SDK für Go der Vorgang zusammengefasst APIs :

- Die `GetObjectGetObjectRequest`, und die `GetObjectWithContext` Entschlüsselung APIs werden durch GetObjectersetzt.
- Die `PutObjectWithContext` Verschlüsselung `PutObjectPutObjectRequest`, und APIs werden durch PutObjectersetzt.

Informationen zur Migration auf die 3.x-Hauptversion des Verschlüsselungsclients finden Sie in diesem Handbuch.

Änderungen an den Serviceanpassungen

Amazon S3

Eine wichtige Änderung, die Sie bei der Migration von Version AWS SDK für Go 1 auf Version 2 beachten sollten, betrifft den Umgang mit der serverseitigen Verschlüsselung mit vom Kunden bereitgestellten Schlüsseln (SSE-C). `SSECustomerKey` In AWS SDK für Go Version 1 wurde die Kodierung von `SSECustomerKey` zu Base64 intern vom SDK übernommen. In SDK v2 wurde diese automatische Kodierung entfernt, und es ist nun erforderlich, die Kodierung manuell nach Base64 zu codieren`SSECustomerKey`, bevor sie an das SDK übergeben wird.

Beispiel für eine Anpassung:

```
// V1

import (
  "context"
  "encoding/base64"
  "github.com/aws/aws-sdk-go-v2/config"
  "github.com/aws/aws-sdk-go-v2/service/s3"
)
// ... more code

plainTextKey := "12345678901234567890123456789012" // 32 bytes in length

// calculate md5..

_, err = client.PutObjectWithContext(context.Background(), &s3.PutObjectInput{
    Bucket: aws.String("amzn-s3-demo-bucket"),
    Key:    aws.String("your-object-key"),
    Body:                 strings.NewReader("hello-world"),
    SSECustomerKey:       &plainTextKey,
```

```
    SSECustomerKeyMD5:    &base64Md5,
    SSECustomerAlgorithm: aws.String("AES256"),
})

// ... more code
```

```
// V2

import (
  "github.com/aws/aws-sdk-go/aws"
  "github.com/aws/aws-sdk-go/aws/session"
  "github.com/aws/aws-sdk-go/service/s3"
)

// ... more code

plainTextKey := "12345678901234567890123456789012" // 32 bytes in length
base64EncodedKey := base64.StdEncoding.EncodeToString([]byte(plainTextKey))

// calculate md5..

_, err = client.PutObject(context.Background(), &s3.PutObjectInput{
    Bucket: aws.String("amzn-s3-demo-bucket"),
    Key:    aws.String("your-object-key"),
    Body:                 strings.NewReader("hello-world"),
    SSECustomerKey:       &base64EncodedKey,
    SSECustomerKeyMD5:    &base64Md5,
    SSECustomerAlgorithm: aws.String("AES256"),
})

// ... more code
```

Sicherheit in AWS SDK für Go

Cloud-Sicherheit AWS hat höchste Priorität. Als AWS Kunde profitieren Sie von Rechenzentren und Netzwerkarchitekturen, die darauf ausgelegt sind, die Anforderungen der sicherheitssensibelsten Unternehmen zu erfüllen.

Sicherheit ist eine gemeinsame AWS Verantwortung von Ihnen und Ihnen. Das Modell der geteilten Verantwortung beschreibt dies als Sicherheit der Cloud selbst und Sicherheit in der Cloud:

- Sicherheit der Cloud — AWS ist verantwortlich für den Schutz der Infrastruktur, auf der AWS Dienste in der ausgeführt AWS Cloud werden. AWS bietet Ihnen auch Dienste, die Sie sicher nutzen können. Externe Prüfer testen und verifizieren regelmäßig die Wirksamkeit unserer Sicherheitsmaßnahmen im Rahmen der AWS . Weitere Informationen zu den Compliance-Programmen, die für gelten AWS SDK für Go, finden Sie unter AWS Services im Umfang nach Compliance-Programmen AWS .
- Sicherheit in der Cloud — Ihre Verantwortung richtet sich nach dem AWS Dienst, den Sie nutzen. Sie sind auch für andere Faktoren verantwortlich, etwa für die Vertraulichkeit Ihrer Daten, für die Anforderungen Ihres Unternehmens und für die geltenden Gesetze und Vorschriften.

Diese Dokumentation hilft Ihnen zu verstehen, wie Sie das Modell der gemeinsamen Verantwortung bei der Nutzung anwenden können AWS SDK für Go. In den folgenden Themen erfahren Sie, wie Sie die Konfiguration vornehmen AWS SDK für Go , um Ihre Sicherheits- und Compliance-Ziele zu erreichen. Sie erfahren auch, wie Sie andere AWS Dienste nutzen können, die Sie bei der Überwachung und Sicherung Ihrer AWS SDK für Go Ressourcen unterstützen.

Themen

- Datenschutz in AWS SDK für Go
- Konformitätsvalidierung für AWS SDK für Go
- Resilienz in AWS SDK für Go

Datenschutz in AWS SDK für Go

Das Modell der AWS gemeinsamen Verantwortung und geteilter Verantwortung gilt für den Datenschutz in AWS SDK für Go. Wie in diesem Modell beschrieben, AWS ist verantwortlich für den Schutz der globalen Infrastruktur, auf der alle Systeme laufen AWS Cloud. Sie sind dafür

verantwortlich, die Kontrolle über Ihre in dieser Infrastruktur gehosteten Inhalte zu behalten. Sie sind auch für die Sicherheitskonfiguration und die Verwaltungsaufgaben für die von Ihnen verwendeten AWS-Services verantwortlich. Weitere Informationen zum Datenschutz finden Sie unter Häufig gestellte Fragen zum Datenschutz. Informationen zum Datenschutz in Europa finden Sie im Blog-Beitrag AWS -Modell der geteilten Verantwortung und in der DSGVO im AWS -Sicherheitsblog.

Aus Datenschutzgründen empfehlen wir, dass Sie AWS-Konto Anmeldeinformationen schützen und einzelne Benutzer mit AWS IAM Identity Center oder AWS Identity and Access Management (IAM) einrichten. So erhält jeder Benutzer nur die Berechtigungen, die zum Durchführen seiner Aufgaben erforderlich sind. Außerdem empfehlen wir, die Daten mit folgenden Methoden schützen:

- Verwenden Sie für jedes Konto die Multi-Faktor-Authentifizierung (MFA).
- Verwenden Sie SSL/TLS, um mit Ressourcen zu kommunizieren. AWS Wir benötigen TLS 1.2 und empfehlen TLS 1.3.
- Richten Sie die API und die Protokollierung von Benutzeraktivitäten mit ein. AWS CloudTrail Informationen zur Verwendung von CloudTrail Pfaden zur Erfassung von AWS Aktivitäten finden Sie unter Arbeiten mit CloudTrail Pfaden im AWS CloudTrail Benutzerhandbuch.
- Verwenden Sie AWS Verschlüsselungslösungen zusammen mit allen darin enthaltenen Standardsicherheitskontrollen AWS-Services.
- Verwenden Sie erweiterte verwaltete Sicherheitsservices wie Amazon Macie, die dabei helfen, in Amazon S3 gespeicherte persönliche Daten zu erkennen und zu schützen.
- Wenn Sie für den Zugriff AWS über eine Befehlszeilenschnittstelle oder eine API FIPS 140-3-validierte kryptografische Module benötigen, verwenden Sie einen FIPS-Endpunkt. Weitere Informationen über verfügbare FIPS-Endpunkte finden Sie unter Federal Information Processing Standard (FIPS) 140-3.

Wir empfehlen dringend, in Freitextfeldern, z. B. im Feld Name, keine vertraulichen oder sensiblen Informationen wie die E-Mail-Adressen Ihrer Kunden einzugeben. Dies gilt auch, wenn Sie mit der Konsole, der AWS SDK für Go API oder auf andere AWS-Services Weise arbeiten oder diese verwenden. AWS CLI AWS SDKs Alle Daten, die Sie in Tags oder Freitextfelder eingeben, die für Namen verwendet werden, können für Abrechnungs- oder Diagnoseprotokolle verwendet werden. Wenn Sie eine URL für einen externen Server bereitstellen, empfehlen wir dringend, keine Anmeldeinformationen zur Validierung Ihrer Anforderung an den betreffenden Server in die URL einzuschließen.

Konformitätsvalidierung für AWS SDK für Go

Informationen darüber, ob AWS-Service ein AWS-Services in den Geltungsbereich bestimmter Compliance-Programme fällt, finden Sie unter Umfang nach Compliance-Programm AWS-Services unter . Wählen Sie dort das Compliance-Programm aus, an dem Sie interessiert sind. Allgemeine Informationen finden Sie unter AWS Compliance-Programme AWS .

Sie können Prüfberichte von Drittanbietern unter herunterladen AWS Artifact. Weitere Informationen finden Sie unter Berichte herunterladen unter .

Ihre Verantwortung für die Einhaltung der Vorschriften bei der Nutzung AWS-Services hängt von der Vertraulichkeit Ihrer Daten, den Compliance-Zielen Ihres Unternehmens und den geltenden Gesetzen und Vorschriften ab. AWS stellt die folgenden Ressourcen zur Verfügung, die Sie bei der Einhaltung der Vorschriften unterstützen:

- Compliance und Governance im Bereich Sicherheit – In diesen Anleitungen für die Lösungsimplementierung werden Überlegungen zur Architektur behandelt. Außerdem werden Schritte für die Bereitstellung von Sicherheits- und Compliance-Features beschrieben.
- Referenz für berechtigte HIPAA-Services – Listet berechtigte HIPAA-Services auf. Nicht alle AWS-Services sind HIPAA-fähig.
- AWS Compliance-Ressourcen — Diese Sammlung von Arbeitsmappen und Leitfäden gilt möglicherweise für Ihre Branche und Ihren Standort.
- AWS Leitfäden zur Einhaltung von Vorschriften für Kunden — Verstehen Sie das Modell der gemeinsamen Verantwortung aus dem Blickwinkel der Einhaltung von Vorschriften. In den Leitfäden werden die bewährten Verfahren zur Sicherung zusammengefasst AWS-Services und die Leitlinien den Sicherheitskontrollen in verschiedenen Frameworks (einschließlich des National Institute of Standards and Technology (NIST), des Payment Card Industry Security Standards Council (PCI) und der International Organization for Standardization (ISO)) zugeordnet.
- Evaluierung von Ressourcen anhand von Regeln im AWS Config Entwicklerhandbuch — Der AWS Config Service bewertet, wie gut Ihre Ressourcenkonfigurationen den internen Praktiken, Branchenrichtlinien und Vorschriften entsprechen.
- AWS Security Hub— Dies AWS-Service bietet einen umfassenden Überblick über Ihren internen Sicherheitsstatus. AWS Security Hub verwendet Sicherheitskontrollen, um Ihre AWS -Ressourcen zu bewerten und Ihre Einhaltung von Sicherheitsstandards und bewährten Methoden zu überprüfen. Die Liste der unterstützten Services und Kontrollen finden Sie in der Security-Hub-Steuerelementreferenz.

- Amazon GuardDuty — Dies AWS-Service erkennt potenzielle Bedrohungen für Ihre Workloads AWS-Konten, Container und Daten, indem es Ihre Umgebung auf verdächtige und böswillige Aktivitäten überwacht. GuardDuty kann Ihnen helfen, verschiedene Compliance-Anforderungen wie PCI DSS zu erfüllen, indem es die in bestimmten Compliance-Frameworks vorgeschriebenen Anforderungen zur Erkennung von Eindringlingen erfüllt.
- AWS Audit Manager— Auf diese AWS-Service Weise können Sie Ihre AWS Nutzung kontinuierlich überprüfen, um das Risikomanagement und die Einhaltung von Vorschriften und Industriestandards zu vereinfachen.

Resilienz in AWS SDK für Go

Die AWS globale Infrastruktur basiert auf Availability AWS-Regionen Zones. AWS-Regionen bieten mehrere physisch getrennte und isolierte Availability Zones, die über Netzwerke mit niedriger Latenz, hohem Durchsatz und hoher Redundanz miteinander verbunden sind. Mithilfe von Availability Zones können Sie Anwendungen und Datenbanken erstellen und ausführen, die automatisch Failover zwischen Zonen ausführen, ohne dass es zu Unterbrechungen kommt. Availability Zones sind besser verfügbar, fehlertoleranter und skalierbarer als herkömmliche Infrastrukturen mit einem oder mehreren Rechenzentren.

Weitere Informationen zu Availability Zones AWS-Regionen und Availability Zones finden Sie unter AWS Globale Infrastruktur.

Dokumentenverlauf für das AWS SDK für Go v2 Developer Guide

In der folgenden Tabelle werden die Dokumentationsversionen für AWS SDK für Go Version 2 beschrieben.

Änderung	Beschreibung	Datum
Schutz der Datenintegrität mit Prüfsummen	Der Inhalt wurde mit Einzelheiten zur automatischen Prüfsummenberechnung aktualisiert.	16. Januar 2025
Erstversion	Erste Version des AWS SDK für Go v2-Entwicklerhandbuchs	31. Oktober 2024

Die vorliegende Übersetzung wurde maschinell erstellt. Im Falle eines Konflikts oder eines Widerspruchs zwischen dieser übersetzten Fassung und der englischen Fassung (einschließlich infolge von Verzögerungen bei der Übersetzung) ist die englische Fassung maßgeblich.