Guida per gli sviluppatori

AWS SDK per Go v2

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

AWS SDK per Go v2: Guida per gli sviluppatori

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

I marchi e l'immagine commerciale di Amazon non possono essere utilizzati in relazione a prodotti o servizi che non siano di Amazon, in una qualsiasi modalità che possa causare confusione tra i clienti o in una qualsiasi modalità che denigri o discrediti Amazon. Tutti gli altri marchi non di proprietà di Amazon sono di proprietà dei rispettivi proprietari, che possono o meno essere affiliati, collegati o sponsorizzati da Amazon.

Table of Contents

Cos'è la AWS SDK per Go v2? 1
Manutenzione e supporto per le versioni principali dell'SDK 1
Nozioni di base 2
Crea un account Amazon 2
Installa la AWS SDK per Go v2 2
Ottieni le tue AWS chiavi di accesso 3
Per ottenere l'ID della chiave di accesso e la chiave di accesso segreta. 3
Argomenti correlati 4
Richiama un'operazione 4
Configurare l'SDK 6
Caricamento di file di AWS configurazione condivisi 6
Specificare la regione AWS 7
Configura la regione con una variabile d'ambiente 7
Specificare la regione a livello di codice 7
Specifica delle credenziali 7
Ruoli IAM per i processi 9
Ruoli IAM per istanze Amazon EC2 9
Credenziali e configurazione condivise 9
Variabili di ambiente 12
Specificare le credenziali a livello di codice 13
Autenticazione 17
Definizioni 17
Flusso di lavoro per la risoluzione dello schema di autenticazione 20
Supportato nativamente s `AuthScheme` 21
Endpoint client 24
Personalizzazione 24
V2: + `EndpointResolverV2` `BaseEndpoint` 25
V1: `EndpointResolver` 29
Migrazione 31
Client HTTP 35
Sovrascrivere durante il caricamento della configurazione 36
Timeout 36
Dialer 36
Trasporto 37

Intercettori 40
Interceptors vs. middleware 40
Ganci intercettori disponibili 41
Registrazione Interceptor 42
Configurazione globale degli intercettori 43
Registrazione 43
Logger 44
ClientLogMode 44
Tracce e metriche 45
Tracciamento 45
Metriche 46
Tentativi e timeout 48
Retrizzatore standard 49
NopRetryer 49
Personalizzazione del comportamento 49
Timeout 54
Utilizzo di SDK 56
Creazione di un client di servizio 56
NewFromConfig 56
Novità 57
Operazioni di servizio di chiamata 59
Passaggio di parametri a un'operazione di servizio 59
Ignorare le opzioni del client per Operation Call 60
Gestione delle risposte operative 61
Utilizzo simultaneo dei client di servizio 63
Utilizzo di Operation Paginators 65
Usare Waiters 67
Ignorare la configurazione del cameriere 68
Sostituzioni della configurazione avanzata del cameriere 70
Gestione degli errori nell'SDK 71
Errori di registrazione 71
Errori del client di servizio 72
Recupero degli identificatori della richiesta 73
Usa i servizi AWS 75
Checksum di Amazon S3 75
Caricamento di un oggetto 76

Donwload di un oggetto ... 79
Utilità SDK ... 80
Utilità Amazon RDS ... 80
Autenticazione IAM ... 80
Amazon CloudFront Utility ... 81
Firmatario CloudFront URL Amazon ... 81
Servizio di metadati di Amazon EC2 Instance ... 82
Utilità Amazon S3 ... 83
Gestori di trasferimenti Amazon S3 ... 83
Input di streaming non ricercabile ... 93
Middleware ... 95
Scrittura di un middleware personalizzato ... 96
Collegamento del middleware a tutti i client ... 98
Collegamento del middleware a un'operazione specifica ... 98
Trasmissione dei metadati nella pila ... 99
Metadati forniti dall'SDK ... 100
Passare i metadati allo stack ... 101
Domande frequenti ... 104
Come posso configurare il client HTTP del mio SDK? Esistono linee guida o best practice? ... 104
Come devo configurare i timeout operativi? ... 104
Le richieste effettuate dall'SDK scadono o impiegano troppo tempo. Come posso risolvere il problema? ... 105
Come posso correggere un errore? `read: connection reset` ... 105
Perché ricevo errori di «firma non valida» quando utilizzo un proxy HTTP con l'SDK? ... 106
Timing delle operazioni SDK ... 106
Test unitari ... 113
Mocking Client Operations ... 113
Mocking Impaginatori ... 115
Esempi di codice ... 118
Gateway API ... 119
AWS contributi della comunità ... 119
Aurora ... 120
Nozioni di base ... 120
Nozioni di base ... 122
Azioni ... 140
Amazon Bedrock ... 160

Nozioni di base 120
Azioni 140
API Runtime per Amazon Bedrock 163
Nozioni di base 120
Scenari 166
Amazon Nova 170
Generatore di immagini Amazon Titan 171
Anthropic Claude 174
CloudFormation 181
Azioni 140
CloudWatch Registri 183
Azioni 140
Gestore dell'identità per Amazon Cognito 185
Nozioni di base 120
Azioni 140
Scenari 166
Amazon DocumentDB 267
Esempi serverless 267
DynamoDB 269
Nozioni di base 122
Azioni 140
Scenari 166
Esempi serverless 267
AWS contributi della comunità 119
IAM 342
Nozioni di base 120
Nozioni di base 122
Azioni 140
Kinesis 403
Esempi serverless 267
Lambda 406
Nozioni di base 120
Nozioni di base 122
Azioni 140
Scenari 166
Esempi serverless 267

AWS contributi della comunità ... 119
MSK Amazon ... 518
Esempi serverless ... 267
Centrale Partner ... 519
Azioni ... 140
Amazon RDS ... 522
Nozioni di base ... 120
Nozioni di base ... 122
Azioni ... 140
Esempi serverless ... 267
Amazon Redshift ... 558
Nozioni di base ... 120
Nozioni di base ... 122
Azioni ... 140
Simple Storage Service (Amazon S3) ... 577
Nozioni di base ... 120
Nozioni di base ... 122
Azioni ... 140
Scenari ... 166
Esempi serverless ... 267
Amazon SNS ... 666
Nozioni di base ... 120
Azioni ... 140
Scenari ... 166
Esempi serverless ... 267
Amazon SQS ... 694
Nozioni di base ... 120
Azioni ... 140
Scenari ... 166
Esempi serverless ... 267
Migrare alla v2 ... 727
Versione minima Go ... 727
Modularizzazione ... 727
Caricamento della configurazione ... 728
Esempi ... 26
Mocking e `*iface` ... 731

Credenziali e fornitori di credenziali 732
Credenziali statiche 14
Credenziali del ruolo Amazon EC2 IAM 734
Credenziali degli endpoint 735
Credenziali di processo 735
AWS Security Token Service Credenziali 736
Client di servizio 738
Costruzione del cliente 739
Endpoints 741
Autenticazione 741
Richiamo delle operazioni API 742
Tipi di dati di servizio 743
Parametri del puntatore 743
Tipi di errori 744
Impaginatori 746
Waiter 747
Richieste predefinite 747
Richiedi la personalizzazione 749
Ingresso/uscita dell'operazione 750
richiesta/risposta HTTP 754
fasi del gestore 756
Funzionalità 758
Servizio di metadati Amazon EC2 Instance 758
Amazon S3 Transfer Manager 759
Utilità di CloudFront firma Amazon 759
Client di crittografia Amazon S3 760
Modifiche alle personalizzazioni del servizio 760
Simple Storage Service (Amazon S3) 760
Sicurezza 763
Protezione dei dati 763
Convalida della conformità 764
Resilienza 765
Cronologia dei documenti 766
..... dcclxvii

Cos'è la AWS SDK per Go v2?

La AWS SDK per Go versione 2 fornisce APIs utilità che gli sviluppatori possono utilizzare per creare applicazioni Go che utilizzano AWS servizi, come Amazon Elastic Compute Cloud (Amazon) EC2 e Amazon Simple Storage Service (Amazon S3).

L'SDK rimuove la complessità della codifica direttamente sull'interfaccia di un servizio Web. Nasconde molti elementi idraulici di livello inferiore, come l'autenticazione, i nuovi tentativi di richiesta e la gestione degli errori.

L'SDK include anche utili utilità. Ad esempio, il gestore di download e upload di Amazon S3 può suddividere automaticamente oggetti di grandi dimensioni in più parti e trasferirli in parallelo.

Usa la AWS SDK per Go Developer Guide per aiutarti a installare, configurare e utilizzare l'SDK. Questa guida fornisce informazioni sulla configurazione, codice di esempio e un'introduzione alle utilità SDK.

Manutenzione e supporto per le versioni principali dell'SDK

Per informazioni sulla manutenzione e il supporto per le versioni principali dell'SDK e le relative dipendenze sottostanti, consulta quanto segue nella Guida di riferimento AWS SDKs e agli strumenti:

- AWS SDKs e politica di manutenzione degli strumenti
- AWS SDKs e matrice di supporto delle versioni degli strumenti

Inizia con AWS SDK per Go

AWS SDK per Go Richiede una versione minima di Go 1.23. È possibile visualizzare la versione corrente di Go eseguendo il seguente comando:

```
go version
```

Per informazioni sull'installazione o l'aggiornamento della tua versione di Go, consulta Download e installazione nella documentazione di Go.

Crea un account Amazon

Prima di poter utilizzare la AWS SDK per Go versione 2, devi disporre di un account Amazon. Per ulteriori informazioni, consulta Creare un AWS account.

Installa la AWS SDK per Go v2

La AWS SDK per Go v2 utilizza i moduli Go, una funzionalità introdotta in Go 1.11. Inizializza il tuo progetto locale eseguendo il seguente comando Go.

```
go mod init example
```

Dopo aver inizializzato il progetto del modulo Go, sarai in grado di recuperare l'SDK e le sue dipendenze richieste utilizzando il comando. `go get` Queste dipendenze verranno registrate nel `go.mod` file creato dal comando precedente.

I comandi seguenti mostrano come recuperare il set standard di moduli SDK da utilizzare nell'applicazione.

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

Questo recupererà il modulo SDK principale e il modulo di configurazione utilizzato per caricare la configurazione condivisa. AWS

Successivamente puoi installare uno o più client API AWS di servizio richiesti dall'applicazione. Tutti i client API si trovano nella gerarchia di `github.com/aws/aws-sdk-go-v2/service`

importazione. Un set completo di client API attualmente supportati è disponibile qui. Per installare un client di servizio, esegui il seguente comando per recuperare il modulo e registrare la dipendenza nel filego.mod. In questo esempio recuperiamo il client API Amazon S3.

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

Ottieni le tue AWS chiavi di accesso

Le chiavi di accesso sono composte da un ID chiave di accesso e una chiave di accesso segreta che sono utilizzati per firmare le richieste programmatiche eseguite verso AWS. Se non disponi delle chiavi di accesso, puoi crearle utilizzando la Console di AWS gestione. Ti consigliamo di utilizzare le chiavi di accesso IAM anziché le chiavi di accesso all'account AWS root. IAM ti consente di controllare in modo sicuro l'accesso ai AWS servizi e alle risorse del tuo AWS account.

> **Note**
>
> Per creare le chiavi delle credenziali d'accesso, bisogna disporre delle autorizzazioni per effettuare le operazioni richieste dalle azioni IAM. Per ulteriori informazioni, consulta Autorizzazioni necessarie per accedere alle risorse IAM nella Guida per l'utente IAM.

Per ottenere l'ID della chiave di accesso e la chiave di accesso segreta.

1. Apri la console IAM.
2. Nel menu di navigazione, scegliere Users (Utenti).
3. Selezionare il tuo nome utente IAM; (non la casella di spunta).
4. Aprire la scheda Security credentials (Credenziali di sicurezza) e quindi scegliere Create access key (Crea chiave di accesso).
5. Per visualizzare la nuova chiave di accesso, seleziona Show (Mostra). Le credenziali sono simili ai seguenti:
   - ID chiave di accesso: `AKIAIOSFODNN7EXAMPLE`
   - Chiave di accesso segreta `wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY`
6. Per fare il download della coppia di chiavi, scegliere Download .csv file. Conserva le chiavi in un posto sicuro.

> **Warning**
>
> Mantieni riservate le chiavi per proteggere il tuo AWS account e non condividerle mai con nessuno al di fuori dell'organizzazione.

Argomenti correlati

- Che cos'è IAM? nella Guida per l'utente di IAM.
- AWS Credenziali di sicurezza in Amazon Web Services General Reference.

Richiama un'operazione

Dopo aver installato l'SDK, importi AWS i pacchetti nelle tue applicazioni Go per utilizzare l'SDK, come mostrato nell'esempio seguente, che importa le librerie, AWS Config e Amazon S3. Dopo aver importato i pacchetti SDK, viene caricata la configurazione condivisa dell' AWS SDK, viene creato un client e viene richiamata un'operazione API.

```
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)
    }
}
```

Configurare l'SDK

Nella AWS SDK per Go V2, è possibile configurare impostazioni comuni per i client di servizio, come il logger, il livello di registro e la configurazione Retry. La maggior parte delle impostazioni è facoltativa. Tuttavia, per ogni client del servizio, è necessario specificare una AWS regione e le proprie credenziali. L'SDK utilizza questi valori per inviare richieste alla regione corretta e firmare le richieste con le credenziali corrette. È possibile specificare questi valori in modo programmatico nel codice o tramite l'ambiente di esecuzione.

Caricamento di file di AWS configurazione condivisi

Esistono diversi modi per inizializzare un client API di servizio, ma il seguente è lo schema più comune consigliato agli utenti.

Per configurare l'SDK per l'utilizzo dei file di configurazione AWS condivisi, utilizzate il codice seguente:

```
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())`costruirà un AWS.config utilizzando le fonti di configurazione condivise. AWS Ciò include la configurazione di un provider di credenziali, la configurazione della AWS regione e il caricamento della configurazione specifica del servizio. I client di servizio possono essere costruiti utilizzando il file loaded`aws.Config`, che fornisce uno schema coerente per la creazione dei client.

Per ulteriori informazioni sui file di configurazione AWS condivisi, vedere Configuration nella AWS SDKs and Tools Reference Guide.

Specificare la regione AWS

Quando si specifica la regione, si specifica dove inviare le richieste, ad esempio `us-west-2` o `us-east-2`. Per un elenco delle regioni per ogni servizio, consulta Endpoint e quote del servizio in. Riferimenti generali di Amazon Web Services

L'SDK non dispone di una regione predefinita. Per specificare una regione:

- Imposta la variabile di `AWS_REGION` ambiente sulla regione predefinita.
- Imposta la regione in modo esplicito utilizzando config. WithRegioncome argomento per il `config.LoadDefaultConfig` caricamento della configurazione.

RECENSIONE: se imposti una regione utilizzando tutte queste tecniche, l'SDK utilizza la regione specificata in modo esplicito.

Configura la regione con una variabile d'ambiente

Linux, macOS o Unix

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

Windows

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

Specificare la regione a livello di codice

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

Specifica delle credenziali

AWS SDK per Go Richiede le credenziali (una chiave di accesso e una chiave di accesso segreta) a cui firmare le richieste. AWSÈ possibile specificare le credenziali in diverse posizioni, a seconda del caso d'uso specifico. Per informazioni su come ottenere le credenziali, vedere. Inizia con AWS SDK per Go

Quando inizializzate un'`aws.Config`istanza utilizzando`config.LoadDefaultConfig`, l'SDK utilizza la catena di credenziali predefinita per trovare le credenziali. AWS Questa catena di credenziali predefinita cerca le credenziali nel seguente ordine:

1. Variabili di ambiente.
    1. Credenziali statiche (,,) `AWS_ACCESS_KEY_ID` `AWS_SECRET_ACCESS_KEY` `AWS_SESSION_TOKEN`
    2. Token di identità Web () `AWS_WEB_IDENTITY_TOKEN_FILE`
2. File di configurazione condivisi.
    1. L'SDK utilizza per impostazione predefinita `credentials` il file `.aws` nella cartella principale del computer.
    2. L'SDK utilizza come impostazione predefinita `config` il file `.aws` nella cartella principale del computer.
3. Se la tua applicazione utilizza una definizione di attività Amazon ECS o un'operazione RunTask API, ruolo IAM per le attività.
4. Se la tua applicazione è in esecuzione su un'istanza Amazon EC2, ruolo IAM per Amazon EC2.

L'SDK rileva e utilizza automaticamente i provider integrati, senza richiedere configurazioni manuali. Ad esempio, se utilizzi i ruoli IAM per le istanze Amazon EC2, le tue applicazioni utilizzano automaticamente le credenziali dell'istanza. Non è necessario configurare manualmente le credenziali nell'applicazione.

Come procedura ottimale, si AWS consiglia di specificare le credenziali nell'ordine seguente:

1. Usa i ruoli IAM per le attività se la tua applicazione utilizza una definizione di attività Amazon ECS o un'operazione RunTask API.
2. Usa i ruoli IAM per Amazon EC2 (se la tua applicazione è in esecuzione su un'istanza Amazon EC2).

   I ruoli IAM forniscono alle applicazioni sull'istanza credenziali di sicurezza temporanee per effettuare chiamate. AWS I ruoli IAM forniscono un modo semplice per distribuire e gestire le credenziali su più istanze Amazon EC2.
3. Usa credenziali o file di configurazione condivisi.

   Le credenziali e i file di configurazione sono condivisi tra altri e. AWS SDKs AWS CLI Come best practice di sicurezza, consigliamo di utilizzare il file di credenziali per impostare valori

sensibili come la chiave di accesso IDs e le chiavi segrete. Di seguito sono riportati i requisiti di formattazione per ciascuno di questi file.

4. Usa le variabili di ambiente.

   L'impostazione delle variabili di ambiente è utile se stai eseguendo lavori di sviluppo su una macchina diversa da un'istanza Amazon EC2.

Ruoli IAM per i processi

Se la tua applicazione utilizza una definizione o un'`RunTask`operazione di Amazon ECS, utilizza IAM Roles for Tasks per specificare un ruolo IAM che può essere utilizzato dai contenitori in un'attività.

Ruoli IAM per istanze Amazon EC2

Se esegui l'applicazione su un'istanza Amazon EC2, utilizza il ruolo IAM dell'istanza per ottenere credenziali di sicurezza temporanee a cui effettuare chiamate. AWS

Se hai configurato l'istanza per utilizzare i ruoli IAM, l'SDK utilizza automaticamente queste credenziali per l'applicazione. Non è necessario specificare manualmente queste credenziali.

Credenziali e configurazione condivise

Le credenziali condivise e i file di configurazione possono essere utilizzati per condividere configurazioni comuni tra altri strumenti AWS SDKs . Se utilizzi credenziali differenti per diversi strumenti o applicazioni, puoi utilizzare i profili per configurare più chiavi di accesso nello stesso file di configurazione.

È possibile fornire più posizioni di credenziali o file di configurazione utilizzando`config.LoadOptions`, per impostazione predefinita, l'SDK carica i file archiviati nelle posizioni predefinite menzionate in. Specifica delle credenziali

```
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"},
    )
)
```

Quando si utilizzano credenziali e file di configurazione condivisi, se vengono specificati profili duplicati, questi vengono uniti per risolvere un profilo. In caso di conflitto di fusione,

1. Se vengono specificati profili duplicati all'interno dello stesso credentials/config file, le proprietà del profilo specificate in quest'ultimo profilo hanno la precedenza.
2. Se vengono specificati profili duplicati su più file di credenziali o su più file di configurazione, le proprietà del profilo vengono risolte secondo l'ordine di immissione del file in. `config.LoadOptions` Le proprietà del profilo in questi ultimi file hanno la precedenza.
3. Se un profilo esiste sia nel file delle credenziali che nel file di configurazione, le proprietà del file delle credenziali hanno la precedenza.

Se necessario, è possibile attivare `config.LoadOptions` e registrare i `LogConfigurationWarnings` passaggi di risoluzione del profilo.

Creazione del file delle credenziali

Se non disponi di un file di credenziali condiviso (`.aws/credentials`), puoi utilizzare qualsiasi editor di testo per crearne uno nella tua home directory. Aggiungi il seguente contenuto al file delle credenziali, sostituendo *<YOUR_ACCESS_KEY_ID>* e *<YOUR_SECRET_ACCESS_KEY>* con le tue credenziali.

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

L'`[default]`intestazione definisce le credenziali per il profilo predefinito, che verrà utilizzato dall'SDK a meno che non venga configurato per utilizzare un altro profilo.

Puoi anche utilizzare credenziali di sicurezza temporanee aggiungendo i token di sessione al tuo profilo, come mostrato nell'esempio seguente:

```
[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>
```

Il nome di sezione per un profilo non predefinito all'interno di un file di credenziali non deve iniziare con la parola. `profile` Per saperne di più, consulta la Guida AWS SDKs di riferimento agli strumenti.

Creazione del file Config

Se non disponi di un file di credenziali condiviso (`.aws/config`), puoi utilizzare qualsiasi editor di testo per crearne uno nella tua home directory. Aggiungi il seguente contenuto al tuo file di configurazione, sostituendolo *<REGION>* con la regione desiderata.

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

L'`[default]`intestazione definisce la configurazione per il profilo predefinito, che verrà utilizzato dall'SDK a meno che non venga configurato per utilizzare un altro profilo.

È possibile utilizzare profili denominati come illustrato nell'esempio seguente:

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

Il nome di sezione per un profilo non predefinito all'interno di un file di configurazione deve sempre iniziare con la parola`profile`, seguita dal nome del profilo desiderato. Puoi leggere ulteriori informazioni nella AWS SDKs and Tools Reference Guide.

Specificazione dei profili

È possibile includere più chiavi di accesso nello stesso file di configurazione associando ogni set di chiavi di accesso a un profilo. Ad esempio, nel file delle credenziali, puoi dichiarare più profili, come segue.

```
[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>
```

Per impostazione predefinita, l'SDK verifica la variabile di ambiente `AWS_PROFILE` per determinare quale profilo utilizzare. Se non è impostata alcuna `AWS_PROFILE` variabile, l'SDK utilizza il profilo. `default`

A volte, potresti voler usare un profilo diverso con la tua applicazione. Ad esempio, desideri utilizzare `test-account` le credenziali con la tua `myapp` applicazione. È possibile utilizzare questo profilo utilizzando il seguente comando:

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

Puoi anche utilizzare istruisci l'SDK per selezionare un profilo chiamando `os.Setenv("AWS_PROFILE", "test-account")` prima della chiamata o passando un profilo esplicito come argomento`config.LoadDefaultConfig`, come mostrato nell'esempio seguente:

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

> **Note**
>
> Se specificate credenziali nelle variabili di ambiente, l'SDK utilizza sempre tali credenziali, indipendentemente dal profilo specificato.

Variabili di ambiente

Per impostazione predefinita, l'SDK rileva le AWS credenziali impostate nell'ambiente e le utilizza per firmare le richieste. AWS In questo modo non è necessario gestire le credenziali nelle applicazioni.

L'SDK cerca le credenziali nelle seguenti variabili di ambiente:

- `AWS_ACCESS_KEY_ID`
- `AWS_SECRET_ACCESS_KEY`
- `AWS_SESSION_TOKEN` (facoltativo)

Gli esempi seguenti mostrano come configurare le variabili di ambiente.

Linux, OS X o 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
```

Specificare le credenziali a livello di codice

`config.LoadDefaultConfig`consente di fornire un aws esplicito. CredentialProviderdurante il caricamento delle fonti di configurazione condivise. Per passare un provider di credenziali esplicito durante il caricamento della configurazione condivisa, usa config. WithCredentialsProvider. Ad esempio, se `customProvider` fa riferimento a un'istanza di `aws.CredentialProvider` implementazione, può essere passato durante il caricamento della configurazione in questo modo:

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

Se fornite in modo esplicito le credenziali, come in questo esempio, l'SDK utilizza solo quelle credenziali.

> **Note**
>
> Tutti i provider di credenziali passati o restituiti da `LoadDefaultConfig` vengono inseriti automaticamente in un file. CredentialsCache Ciò consente la memorizzazione nella cache e la rotazione delle credenziali in modo sicuro dalla concorrenza. Se configuri `aws.Config`

direttamente un provider in modo esplicito, devi anche inserire in modo esplicito questo tipo di provider utilizzando. NewCredentialsCache

Credenziali statiche

È possibile codificare le credenziali nell'applicazione utilizzando le credenziali. NewStaticCredentialsProviderfornitore di credenziali per impostare in modo esplicito le chiavi di accesso da utilizzare. Esempio:

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

Warning

Non incorporare credenziali all'interno di un'applicazione. Utilizzate questo metodo solo per scopi di test.

Credenziali Single Sign-on

L'SDK fornisce un provider di credenziali per il recupero di credenziali temporanee utilizzando. AWS AWS IAM Identity Center Utilizzando AWS CLI, ci si autentica con il portale di accesso e si autorizza l' AWS accesso alle credenziali temporanee. AWS Quindi configuri l'applicazione per caricare il profilo Single Sign-On (SSO) e l'SDK utilizza le tue credenziali SSO per recuperare le credenziali temporanee che verranno rinnovate automaticamente se scadute. AWS Se le tue credenziali SSO scadono, devi rinnovarle esplicitamente accedendo nuovamente al tuo account IAM Identity Center utilizzando il. AWS CLI

Ad esempio, puoi creare un profilo`dev-profile`, autenticarlo e autorizzarlo utilizzando e configurare l'applicazione come illustrato di seguito AWS CLI.

1. Per prima cosa, crea e `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. Effettua il login utilizzando AWS CLI per autenticare e autorizzare il profilo SSO.

```
$ 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. Quindi configura l'applicazione per utilizzare il profilo SSO.

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

// ...

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

Per ulteriori informazioni sulla configurazione dei profili SSO e sull'autenticazione tramite, AWS CLI consulta la sezione Configurazione dell'uso nella Guida AWS CLI per l'utente. AWS IAM Identity

Center AWS CLI Per ulteriori informazioni sulla costruzione programmatica del provider di credenziali SSO, consulta la documentazione di riferimento dell'API ssocreds.

Credenziali di accesso

È possibile utilizzare le credenziali di accesso esistenti AWS della Console di gestione per l'accesso programmatico ai servizi. AWS Dopo un flusso di autenticazione basato su browser, AWS genera credenziali temporanee che funzionano con strumenti di sviluppo locali come la CLI AWS e. AWS Strumenti per PowerShell AWS SDKs Questa funzionalità semplifica il processo di configurazione e gestione delle credenziali AWS CLI, soprattutto se si preferisce l'autenticazione interattiva alla gestione delle chiavi di accesso a lungo termine.

1. Avvia il flusso di accesso utilizzando la AWS CLI e segui le istruzioni del browser. In questo esempio, salviamo la sessione di accesso su un nuovo profilodev-profile, ma questo è facoltativo.

```
$ aws --profile dev-profile login
```

2. (Facoltativo) Ispeziona il file di configurazione AWS condiviso per verificare che la sessione sia stata stabilita.

```
[profile dev-profile]
login_session = arn:aws:sts::account id>:role
```

3. Quindi configura l'applicazione per utilizzare il profilo di accesso.

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

// ...

cfg, err := config.LoadDefaultConfig(
    context.Background(),
    // only necessary if login session is saved to a non-default profile
    config.WithSharedConfigProfile("dev-profile"),
)
if err != nil {
    return err
}
```

Per ulteriori informazioni sulla configurazione dei profili di accesso e sull'autenticazione tramite la AWS CLI, consulta Accesso per lo sviluppo AWS locale utilizzando le credenziali della console.

Altri fornitori di credenziali

L'SDK fornisce altri metodi per recuperare le credenziali nel modulo delle credenziali. Ad esempio, è possibile recuperare credenziali di sicurezza temporanee o credenziali dall'archivio crittografato. AWS Security Token Service

Fornitori di credenziali disponibili:

- ec2rolecreds — Recupera le credenziali dai ruoli delle istanze Amazon EC2 tramite Amazon EC2 IMDS.
- endpointcreds — Recupera le credenziali da un endpoint HTTP arbitrario.
- processcreds — Recupera le credenziali da un processo esterno che verrà richiamato dalla shell dell'ambiente host.
- stscreds — Recupera credenziali da AWS STS

Configurazione dell'autenticazione

AWS SDK per Go Fornisce la possibilità di configurare il servizio di comportamento di autenticazione. Nella maggior parte dei casi, la configurazione predefinita è sufficiente, ma la configurazione dell'autenticazione personalizzata consente di adottare comportamenti aggiuntivi, ad esempio l'utilizzo di funzionalità di servizio precedenti al rilascio.

Definizioni

Questa sezione fornisce una descrizione di alto livello dei componenti di autenticazione di. AWS SDK per Go

AuthScheme

An AuthSchemeè l'interfaccia che definisce il flusso di lavoro attraverso il quale l'SDK recupera l'identità del chiamante e la allega a una richiesta di operazione.

Uno schema di autenticazione utilizza i seguenti componenti, descritti in dettaglio più avanti:

- Un ID univoco che identifica lo schema

- Un risolutore di identità, che restituisce l'identità del chiamante utilizzata nel processo di firma (ad esempio le credenziali) AWS
- Un firmatario, che esegue l'inserimento effettivo dell'identità del chiamante nella richiesta di trasporto dell'operazione (ad esempio l'intestazione HTTP) `Authorization`

Ogni opzione del client di servizio include un `AuthSchemes` campo, che per impostazione predefinita viene compilato con l'elenco degli schemi di autenticazione supportati da quel servizio.

AuthSchemeResolver

Ogni opzione del client di servizio include un `AuthSchemeResolver` campo. Questa interfaccia, definita per servizio, è l'API chiamata dall'SDK per determinare le possibili opzioni di autenticazione per ogni operazione.

> **Important**
>
> Il resolver dello schema di autenticazione NON impone quale schema di autenticazione viene utilizzato. Restituisce un elenco di schemi che possono essere utilizzati («opzioni»), lo schema finale viene selezionato tramite un algoritmo fisso descritto qui.

Opzione

Restituita da una chiamata a`ResolverAuthSchemes`, un'opzione rappresenta una possibile opzione di autenticazione.

Un'opzione è composta da tre set di informazioni:

- Un ID che rappresenta lo schema possibile
- Un insieme opaco di proprietà da fornire al risolutore di identità dello schema
- Un insieme opaco di proprietà da fornire al firmatario dello schema

Una nota sulle proprietà

Nel 99% dei casi d'uso, i chiamanti non devono preoccuparsi delle proprietà opache per la risoluzione e la firma delle identità. L'SDK estrarrà le proprietà necessarie per ogni schema e le passerà alle interfacce fortemente tipizzate esposte nell'SDK. Ad esempio, il resolver di autenticazione predefinito

per i servizi codifica l'opzione sigV4 in modo da avere proprietà del firmatario per il nome e l'area della firma, i cui valori vengono passati alla v4 configurata dal client. HTTPSignerimplementazione quando è selezionato SigV4.

Identità

Un'identità è una rappresentazione astratta di chi è il chiamante dell'SDK.

Il tipo di identità più comune utilizzato nell'SDK è un insieme di. `aws.Credentials` Nella maggior parte dei casi d'uso, il chiamante non deve preoccuparsi di qualcosa `Identity` di astratto e può lavorare direttamente con i tipi concreti.

> **Note**
>
> Per preservare la compatibilità con le versioni precedenti ed evitare confusione tra le API, il tipo di identità AWS specifico dell'SDK `aws.Credentials` non soddisfa direttamente l'interfaccia. `Identity` Questa mappatura viene gestita internamente.

IdentityResolver

IdentityResolverè l'interfaccia attraverso la quale viene recuperato un `Identity` file.

Nell'SDK `IdentityResolver` esistono versioni concrete in forma fortemente tipizzata (ad esempio aws. CredentialsProvider), l'SDK gestisce questa mappatura internamente.

Un chiamante dovrà solo implementare direttamente l'`IdentityResolver`interfaccia quando definisce uno schema di autenticazione esterno.

Signer

Signer è l'interfaccia attraverso la quale una richiesta viene integrata con il chiamante recuperato. `Identity`

Nell'SDK `Signer` esistono versioni concrete in forma fortemente tipizzata (ad esempio v4). HTTPSigner), l'SDK gestisce questa mappatura internamente.

Un chiamante dovrà solo implementare direttamente l'`Signer`interfaccia quando definisce uno schema di autenticazione esterno.

AuthResolverParameters

Ogni servizio accetta un set specifico di input che vengono passati alla sua funzione di risoluzione, definita in ogni pacchetto di servizi come. `AuthResolverParameters`

I parametri del resolver di base sono i seguenti:

nome	tipo	description
`Operation`	`string`	Il nome dell'operazione che viene richiamata.
`Region`	`string`	La AWS regione del cliente. Presente solo per i servizi che utilizzano SigV4 [A].

Se state implementando il vostro resolver, non dovreste mai aver bisogno di costruire una vostra istanza dei suoi parametri. L'SDK genererà questi valori per ogni richiesta e li passerà all'implementazione.

Flusso di lavoro per la risoluzione dello schema di autenticazione

Quando si chiama un'operazione AWS di servizio tramite l'SDK, dopo la serializzazione della richiesta si verifica la seguente sequenza di azioni:

1. L'SDK richiama l'`AuthSchemeResolver.ResolveAuthSchemes()`API del client, fornendo i parametri di input necessari, per ottenere un elenco di possibili opzioni per l'operazione.
2. L'SDK esegue un'iterazione su tale elenco e seleziona il primo schema che soddisfa le seguenti condizioni.
   - Uno schema con ID corrispondente è presente nell'elenco del client `AuthSchemes`
   - Il risolutore di identità dello schema esiste (non è presente`nil`) nelle Opzioni del client (verificato tramite il `GetIdentityResolver` metodo dello schema, la mappatura ai tipi di risolutori di identità concreti descritti sopra viene gestita internamente) (1)
3. Supponendo che sia stato selezionato uno schema valido, l'SDK richiama la sua API per recuperare l'identità del chiamante. `GetIdentityResolver()` Ad esempio, lo schema di autenticazione SigV4 integrato verrà mappato internamente al provider del client. `Credentials`
4. L'SDK chiama i risolutori di identità (ad esempio per SigV4). `GetIdentity()` `aws.CredentialProvider.Retrieve()`

5. L'SDK chiama i resolver degli endpoint per trovare l'endpoint per la richiesta. `ResolveEndpoint()` L'endpoint può includere metadati aggiuntivi che influenzano il processo di firma (ad esempio un nome di firma univoco per S3 Object Lambda).
6. L'SDK chiama l'API dello schema di autenticazione per recuperare il firmatario e utilizza l'`Signer()`API per firmare la richiesta con l'identità del chiamante recuperata in precedenza. `SignRequest()`

(1) Se l'SDK rileva l'opzione anonima (ID`smithy.api#noAuth`) nell'elenco, viene selezionata automaticamente, poiché non esiste un risolutore di identità corrispondente.

Supportato nativamente s **`AuthScheme`**

I seguenti schemi di autenticazione sono supportati nativamente da. AWS SDK per Go

Nome	ID dello schema	Risolutore di identità	Signer	Note
SIG V4	`aws.auth#sigv4`	leggi. CredentialsProvider	v4. HTTPSigner	L'impostazione predefinita corrente per la maggior parte delle operazioni AWS di servizio.
SigV4a	`aws.auth#sigv4a`	leggi. CredentialsProvider	N/A	L'utilizzo di SigV4a è limitato al momento, l'implementazione del firmatario è interna. Leggete questo annuncio per scoprire un nuovo modulo opt-in aws-http-authche illustra lo scopo

Nome	ID dello schema	Risolutore di identità	Signer	Note
				generale della firma delle richieste HTTP. APIs
SIGV4Express	`com.amazonaws.s3#sigv4express`	s3. ExpressCredentials Provider	v4. HTTPSigner	Usato per Express One Zone.
Portatore HTTP	`smithy.api#httpBearerAuth`	portatore di fucina. TokenProvider	Smithy Bearer. Firmatario	Usato da codecatalyst.
Anonimo	`smithy.api#noAuth`	N/A	n/a	Nessuna autenticazione: non è richiesta alcuna identità e la richiesta non è firmata o autenticata.

Configurazione dell'identità

In AWS SDK per Go, i componenti di identità di uno schema di autenticazione sono configurati nel client SDK. `Options` L'SDK raccoglierà e utilizzerà automaticamente i valori di questi componenti per lo schema selezionato quando viene chiamata un'operazione.

> **Note**
>
> Per motivi di compatibilità con le versioni precedenti, l'SDK consente implicitamente l'uso dello schema di autenticazione anonimo se non sono configurati risolutori di identità. Ciò può essere ottenuto manualmente impostando tutti i resolver di identità

su un client (il resolver di identità sigv4 può anche `Options` essere impostato su`nil`). `aws.AnonymousCredentials{}`

Configurazione del firmatario

In AWS SDK per Go, i componenti firmatari di uno schema di autenticazione sono configurati nel client SDK. `Options` L'SDK raccoglierà e utilizzerà automaticamente i valori di questi componenti per lo schema selezionato quando viene chiamata un'operazione. Non è necessaria alcuna configurazione aggiuntiva.

Schema di autenticazione personalizzato

Per definire uno schema di autenticazione personalizzato e configurarlo per l'uso, il chiamante deve eseguire le seguenti operazioni:

1. Definire un'implementazione AuthScheme
2. Registra lo schema nell'elenco dei client SDK `AuthSchemes`
3. Strumenta il client SDK `AuthSchemeResolver` a restituire un'autenticazione `Option` con l'ID dello schema, ove applicabile

Warning

I seguenti servizi hanno un comportamento di autenticazione unico o personalizzato. Ti consigliamo di delegare all'implementazione predefinita e di eseguire il wrap di conseguenza se hai bisogno di un comportamento di autenticazione personalizzato:

Servizio	Note
S3	Uso condizionale di SigV4a e SigV4Expr ess a seconda dell'input dell'operazione.
EventBridge	Uso condizionale di SigV4A in base all'input dell'operazione.
Cognito	Alcune operazioni sono solo anonime.

Servizio	Note
SSO	Alcune operazioni sono esclusivamente anonime.
STS	Alcune operazioni sono esclusivamente anonime.

Configurazione degli endpoint client

> **Warning**
>
> La risoluzione degli endpoint è un argomento SDK avanzato. Modificando queste impostazioni rischi di violare il codice. Le impostazioni predefinite dovrebbero essere applicabili alla maggior parte degli utenti negli ambienti di produzione.

AWS SDK per Go Offre la possibilità di configurare un endpoint personalizzato da utilizzare per un servizio. Nella maggior parte dei casi, la configurazione predefinita sarà sufficiente. La configurazione degli endpoint personalizzati consente comportamenti aggiuntivi, ad esempio l'utilizzo di versioni precedenti al rilascio di un servizio.

Personalizzazione

Esistono due «versioni» della configurazione della risoluzione degli endpoint all'interno dell'SDK.

- v2, rilasciata nel terzo trimestre del 2023, configurata tramite:
  - `EndpointResolverV2`
  - `BaseEndpoint`
- v1, rilasciato insieme all'SDK, configurato tramite:
  - `EndpointResolver`

Consigliamo agli utenti con risoluzione degli endpoint v1 di migrare alla versione 2 per ottenere l'accesso a nuove funzionalità di servizio relative agli endpoint.

V2: + `EndpointResolverV2` `BaseEndpoint`

Nella risoluzione v2, `EndpointResolverV2` è il meccanismo definitivo attraverso il quale avviene la risoluzione degli endpoint. Il `ResolveEndpoint` metodo del resolver viene richiamato come parte del flusso di lavoro per ogni richiesta effettuata nell'SDK. Il nome host del file `Endpoint` restituito dal resolver viene utilizzato così com'è quando si effettua la richiesta (i serializzatori operativi possono comunque essere aggiunti al percorso HTTP).

La risoluzione v2 include una configurazione aggiuntiva a livello di client, `BaseEndpoint` che viene utilizzata per specificare un nome host «base» per l'istanza del servizio. Il valore qui impostato non è definitivo: alla fine viene passato come parametro al client `EndpointResolverV2` quando si verifica la risoluzione finale (continua a leggere per ulteriori informazioni sui parametri). `EndpointResolverV2` L'implementazione del resolver ha quindi l'opportunità di ispezionare e potenzialmente modificare quel valore per determinare l'endpoint finale.

Ad esempio, se esegui una `GetObject` richiesta S3 su un determinato bucket con un client in cui hai specificato a`BaseEndpoint`, il resolver predefinito inietterà il bucket nel nome host se è compatibile con l'host virtuale (supponendo che tu non abbia disabilitato l'hosting virtuale nella configurazione del client).

In pratica, molto probabilmente `BaseEndpoint` verrà utilizzato per indirizzare il cliente verso un'istanza di sviluppo o di anteprima di un servizio.

Parametri di `EndpointResolverV2`

Ogni servizio accetta un set specifico di input che vengono passati alla sua funzione di risoluzione, definita in ogni pacchetto di servizi come`EndpointParameters`.

Ogni servizio include i seguenti parametri di base, utilizzati per facilitare la risoluzione generale degli endpoint all'interno di: AWS

nome	tipo	description
`Region`	`string`	La regione del cliente AWS
`Endpoint`	`string`	Il valore impostato per `BaseEndpoint` nella configurazione del client

nome	tipo	description
`UseFips`	`bool`	Se gli endpoint FIPS sono abilitati nella configurazione del client
`UseDualStack`	`bool`	Se gli endpoint dual-stack sono abilitati nella configura zione del client

I servizi possono specificare parametri aggiuntivi necessari per la risoluzione. Ad esempio, S3 `EndpointParameters` include il nome del bucket e diverse impostazioni di funzionalità specifiche di S3, ad esempio se l'indirizzamento dell'host virtuale è abilitato.

Se ne stai implementando uno tuo`EndpointResolverV2`, non dovresti mai aver bisogno di creare la tua istanza di. `EndpointParameters` L'SDK genererà i valori per ogni richiesta e li passerà all'implementazione.

Una nota su Amazon S3

Amazon S3 è un servizio complesso con molte delle sue funzionalità modellate attraverso complesse personalizzazioni degli endpoint, come l'hosting virtuale di bucket, S3 MRAP e altro ancora.

Per questo motivo, ti consigliamo di non sostituire l'implementazione nel tuo client S3. `EndpointResolverV2` Se hai bisogno di estenderne il comportamento di risoluzione, magari inviando richieste a uno stack di sviluppo locale con considerazioni aggiuntive sugli endpoint, ti consigliamo di completare l'implementazione predefinita in modo da delegare a quella predefinita come riserva (mostrata negli esempi seguenti).

Esempi

Con **`BaseEndpoint`**

Il seguente frammento di codice mostra come indirizzare il client S3 verso un'istanza locale di un servizio, che in questo esempio è ospitato sul dispositivo di loopback sulla porta 8080.

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

Con **EndpointResolverV2**

Il seguente frammento di codice mostra come inserire un comportamento personalizzato nella risoluzione degli endpoint di S3 utilizzando. `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{
            // ...
        }
    })
}
```

Con entrambi

Il seguente programma di esempio dimostra l'interazione tra `BaseEndpoint` e`EndpointResolverV2`. Questo è un caso d'uso avanzato:

```
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)
}
```

Quando viene eseguito, il programma precedente emette quanto segue:

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

V1: `EndpointResolver`

> **Warning**
>
> La risoluzione degli endpoint v1 viene mantenuta per motivi di compatibilità con le versioni precedenti ed è isolata dal comportamento moderno della risoluzione degli endpoint v2. Verrà utilizzato solo se il `EndpointResolver` campo è impostato dal chiamante.
> L'uso della v1 probabilmente impedirà di accedere alle funzionalità del servizio relative agli endpoint introdotte con o dopo il rilascio della risoluzione v2. Vedi «Migrazione» per istruzioni su come eseguire l'aggiornamento.

A EndpointResolverpuò essere configurato per fornire una logica di risoluzione degli endpoint personalizzata per i client di servizio. È possibile utilizzare un resolver di endpoint personalizzato per sovrascrivere la logica di risoluzione degli endpoint di un servizio per tutti gli endpoint o solo per un endpoint regionale specifico. Un resolver di endpoint personalizzato può attivare la logica di risoluzione degli endpoint del servizio in fallback se un resolver personalizzato non desidera risolvere un endpoint richiesto. EndpointResolverWithOptionsFuncpuò essere usato per racchiudere facilmente le funzioni in modo da soddisfare l'interfaccia. `EndpointResolverWithOptions`

A `EndpointResolver` può essere facilmente configurato passando il resolver incluso nella confezione WithEndpointResolverWithOptionsa LoadDefaultConfig, dando la possibilità di sovrascrivere gli endpoint durante il caricamento delle credenziali e configurando il risultato con un resolver endpoint personalizzato. `aws.Config`

Al resolver dell'endpoint vengono assegnati il servizio e la regione sotto forma di stringa, che consente al resolver di determinare dinamicamente il proprio comportamento. Ogni pacchetto client di servizio contiene una `ServiceID` costante esportata che può essere utilizzata per determinare quale client di servizio sta richiamando il resolver dell'endpoint.

Un endpoint resolver può utilizzare il valore di errore EndpointNotFoundErrorsentinel per attivare la risoluzione di fallback sulla logica di risoluzione predefinita del client del servizio. Ciò consente di sovrascrivere in modo selettivo uno o più endpoint senza interruzioni senza dover gestire la logica di fallback.

Se l'implementazione del resolver per endpoint restituisce un errore diverso da`EndpointNotFoundError`, la risoluzione degli endpoint si interromperà e l'operazione del servizio restituirà un errore all'applicazione.

Esempi

Con fallback

Il seguente frammento di codice mostra come un singolo endpoint di servizio può essere sovrascritto per DynamoDB con un comportamento di fallback per altri endpoint:

```
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))
```

Senza fallback

Il seguente frammento di codice mostra come un singolo endpoint di servizio può essere sovrascritto per DynamoDB senza un comportamento di fallback per altri endpoint:

```
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))
```

Endpoint immutabili

> **Warning**
>
> L'impostazione di un endpoint come immutabile può impedire il corretto funzionamento di alcune funzionalità del client di servizio e potrebbe comportare un comportamento indefinito. È necessario prestare attenzione quando si definisce un endpoint come immutabile.

Alcuni client di servizio, come Amazon S3, possono modificare l'endpoint restituito dal resolver per determinate operazioni di servizio. Ad esempio, Amazon S3 gestirà automaticamente il Virtual Bucket Addressing modificando l'endpoint risolto. Puoi impedire che l'SDK modifichi i tuoi endpoint personalizzati impostando su. HostnameImmutable`true` Esempio:

```
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))
```

Migrazione

Durante la migrazione dalla versione v1 alla versione 2 della risoluzione degli endpoint, si applicano i seguenti principi generali:

- Restituire un Endpoint con HostnameImmutableset to `false` equivale all'incirca `BaseEndpoint` a impostare l'URL originariamente restituito dalla v1 e lasciarlo come predefinito. `EndpointResolverV2`
- Restituire un Endpoint con HostnameImmutable set to `true` equivale all'incirca a implementare un `EndpointResolverV2` che restituisce l'URL originariamente restituito dalla v1.
  - L'eccezione principale riguarda le operazioni con prefissi di endpoint modellati. Una nota al riguardo è riportata più avanti.

Di seguito sono riportati alcuni esempi di questi casi.

> **Warning**
>
> Gli endpoint immutabili V1 e la risoluzione V2 non hanno un comportamento equivalente. Ad esempio, le sostituzioni di firma per funzionalità personalizzate come S3 Object Lambda sarebbero comunque impostate per gli endpoint immutabili restituiti tramite il codice v1, ma lo stesso non verrà fatto per la v2.

Nota sui prefissi degli host

Alcune operazioni sono modellate con prefissi host da anteporre all'endpoint risolto. Questo comportamento deve funzionare insieme all'output di ResolveEndpoint V2 e pertanto il prefisso host verrà comunque applicato a quel risultato.

È possibile disabilitare manualmente il prefisso dell'host dell'endpoint applicando un middleware, vedere la sezione degli esempi.

Esempi

Endpoint mutabile

Il seguente esempio di codice mostra come migrare un endpoint resolver v1 di base che restituisce un endpoint modificabile:

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

Endpoint immutabile

```
// 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{}
})
```

Disabilita il prefisso dell'host

```
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)
    }
}
```

Personalizzazione del client HTTP

AWS SDK per Go Utilizza un client HTTP predefinito con valori di configurazione predefiniti. Sebbene sia possibile modificare alcuni di questi valori di configurazione, il client e il trasporto HTTP predefiniti non sono sufficientemente configurati per i clienti che li utilizzano AWS SDK per Go in un ambiente con requisiti di velocità effettiva elevata e bassa latenza. Per ulteriori informazioni, consulta la sezione in Domande frequenti quanto i consigli di configurazione variano in base a carichi di lavoro specifici. Questa sezione descrive come configurare un client HTTP personalizzato e utilizzarlo per creare AWS SDK per Go chiamate.

Per facilitare la creazione di un client HTTP personalizzato, questa sezione descrive come NewBuildableClientconfigurare impostazioni personalizzate e utilizzare tale client con un client di AWS SDK per Go servizio.

Definiamo cosa vogliamo personalizzare.

Sovrascrivere durante il caricamento della configurazione

È possibile fornire client HTTP personalizzati durante la chiamata LoadDefaultConfigavvolgendo il client utilizzando With HTTPClient e passando il valore risultante a. `LoadDefaultConfig` Ad esempio, per passare `customClient` come nostro cliente:

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

Timeout

`BuildableHTTPClient`Può essere configurato con un limite di timeout della richiesta. Questo timeout include il tempo necessario per la connessione, l'elaborazione di eventuali reindirizzamenti e la lettura del corpo completo della risposta. Ad esempio, per modificare il timeout del client:

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

// ...

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

Dialer

`BuildableHTTPClient`Fornisce una meccanica di costruzione per la creazione di client con opzioni Dialer modificate. L'esempio seguente mostra come configurare le impostazioni di un client. `Dialer`

```
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
})
```

Impostazioni

Dialer. KeepAlive

Questa impostazione rappresenta il periodo di mantenimento per una connessione di rete attiva.

Imposta su un valore negativo per disabilitare i keep-alive.

Imposta su 0 per abilitare i keep-alive se supportati dal protocollo e dal sistema operativo.

I protocolli di rete o i sistemi operativi che non supportano i keep-alive ignorano questo campo. Per impostazione predefinita, TCP abilita keep alive.

Vedi https://golang. org/pkg/net/#Dialer. KeepAlive

Imposta `KeepAlive` come Time.Duration.

Dialer.Timeout

Questa impostazione rappresenta il tempo massimo di attesa di una chiamata per la creazione di una connessione.

L'impostazione predefinita è 30 secondi.

Vedi https://golang. org/pkg/net/#Dialer .Timeout

Imposta **`Timeout`** come time.Duration.

Trasporto

`BuildableHTTPClient`Fornisce una meccanica di costruzione per la costruzione di client con opzioni di trasporto modificate.

Configurazione di un proxy

Se non riesci a connetterti direttamente a Internet, puoi utilizzare le variabili di ambiente supportate da Go (`HTTP_PROXY/HTTPS_PROXY`) o creare un client HTTP personalizzato per configurare il proxy. L'esempio seguente configura il client da utilizzare `PROXY_URL` come endpoint proxy:

```
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)
})
```

Altre impostazioni

Di seguito sono riportate alcune altre `Transport` impostazioni che possono essere modificate per ottimizzare il client HTTP. Tutte le impostazioni aggiuntive non descritte qui sono disponibili nella documentazione del tipo di trasporto. Queste impostazioni possono essere applicate come illustrato nell'esempio seguente:

```
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
})
```

Trasporto. ExpectContinueTimeout

Se la richiesta ha un'intestazione «Expect: 100-continue», questa impostazione rappresenta il tempo massimo di attesa per le intestazioni della prima risposta di un server dopo aver scritto completamente le intestazioni della richiesta. Questo tempo non include il tempo necessario per inviare l'intestazione della richiesta. Il client HTTP invia il suo payload dopo che questo timeout è scaduto.

Impostazione predefinita: 1 secondo.

Imposta su 0 per non richiedere alcun timeout e invia il payload della richiesta senza attese. Un caso d'uso è quando si verificano problemi con proxy o servizi di terze parti che richiedono una sessione simile all'uso di Amazon S3 nella funzione mostrata più avanti.

Vedi https://golang. org/pkg/net/http/#Transport. ExpectContinueTimeout

Imposta `ExpectContinue` come Time.Duration.

Trasporto. IdleConnTimeout

Questa impostazione rappresenta la quantità massima di tempo per mantenere attiva una connessione di rete inattiva tra le richieste HTTP.

Imposta su 0 per non avere limiti.

Vedi https://golang. org/pkg/net/http/#Transport. IdleConnTimeout

Imposta `IdleConnTimeout` come Time.Duration.

Trasporto. MaxIdleConns

Questa impostazione rappresenta il numero massimo di connessioni inattive (keep-alive) su tutti gli host. Un caso d'uso per aumentare questo valore è quando si vedono molte connessioni in un breve periodo dagli stessi client

0 significa nessun limite.

Vedi https://golang. org/pkg/net/http/#Transport. MaxIdleConns

Imposta `MaxIdleConns` come int.

Trasporto. MaxIdleConnsPerHost

Questa impostazione rappresenta il numero massimo di connessioni inattive (keep-alive) da mantenere per host. Un caso d'uso per aumentare questo valore è quando si vedono molte connessioni in un breve periodo dagli stessi client

L'impostazione predefinita è di due connessioni inattive per host.

Imposta su 0 per utilizzare DefaultMaxIdleConnsPerHost (2).

Vedi https://golang. org/pkg/net/http/#Transport. MaxIdleConnsPerHost

Imposta `MaxIdleConnsPerHost` come int.

Trasporto. ResponseHeaderTimeout

Questa impostazione rappresenta il tempo massimo di attesa che un client legga l'intestazione della risposta.

Se il client non è in grado di leggere l'intestazione della risposta entro questo periodo, la richiesta ha esito negativo e viene generato un errore di timeout.

Fai attenzione a impostare questo valore quando usi funzioni Lambda a esecuzione prolungata, poiché l'operazione non restituisce alcuna intestazione di risposta fino al termine o al timeout della funzione Lambda. Tuttavia, è ancora possibile utilizzare questa opzione con l'operazione API** **. InvokeAsync

L'impostazione predefinita è nessun timeout; attendi per sempre.

Vedi https://golang. org/pkg/net/http/#Transport. ResponseHeaderTimeout

Imposta `ResponseHeaderTimeout` come Time.Duration.

Trasporto. TLSHandshakeTimeout

Questa impostazione rappresenta il tempo massimo di attesa per il completamento di un handshake TLS.

Il valore predefinito è 10 secondi.

Zero significa nessun timeout.

Vedi https://golang. org/pkg/net/http/#Transport. TLSHandshakeTimeout

Imposta `TLSHandshakeTimeout` come Time.Duration.

Intercettori HTTP

È possibile utilizzare gli intercettori per agganciarsi all'esecuzione di richieste e risposte API. Gli intercettori sono meccanismi aperti in cui l'SDK richiama il codice scritto dall'utente per inserire il comportamento nel ciclo di vita. request/response In questo modo, puoi modificare una richiesta in corso, eseguire il debug dell'elaborazione delle richieste, visualizzare le eccezioni e altro ancora.

Interceptors vs. middleware

La versione AWS SDK per Go 2 fornisce sia intercettori che middleware per personalizzare l'elaborazione delle richieste. Sebbene entrambi abbiano scopi simili, sono progettati per destinatari e casi d'uso diversi:

- Gli Interceptors sono progettati per gli utenti SDK che desiderano personalizzare l' request/response elaborazione con una semplice API incentrata su HTTP. Forniscono punti di accesso specifici nel ciclo di vita della richiesta e funzionano direttamente con le richieste e le risposte HTTP.
- Il middleware è un sistema più avanzato, indipendente dal trasporto, utilizzato principalmente internamente dall'SDK. Sebbene potente, il middleware richiede una conoscenza più approfondita degli interni dell'SDK e coinvolge interfacce più complesse.

Principali vantaggi degli intercettori rispetto al middleware per i casi d'uso più comuni:

- Incentrato sull'HTTP: gli intercettori funzionano direttamente con le richieste e le risposte HTTP, eliminando la necessità di controllare il tipo di trasporto richiesto dal middleware.
- Interfacce più semplici: ogni hook interceptor ha un'interfaccia specifica e mirata anziché il modello middleware generico.
- Modello di esecuzione più chiaro: gli intercettori vengono eseguiti in punti ben definiti del ciclo di vita della richiesta senza la necessità di conoscere l'ordinamento degli stack del middleware.

> **Note**
>
> Gli interceptor sono basati sul sistema middleware esistente, quindi entrambi possono coesistere nella stessa applicazione. Il middleware rimane disponibile per casi d'uso avanzati che richiedono un comportamento indipendente dal trasporto o una manipolazione complessa dello stack.

Ganci intercettori disponibili

La AWS SDK per Go v2 fornisce ganci di intercettazione in varie fasi del ciclo di vita della richiesta. Ogni hook corrisponde a un'interfaccia specifica che puoi implementare:

- `BeforeExecution`- Primo hook chiamato durante l'esecuzione dell'operazione
- `BeforeSerialization`- Prima che il messaggio di input venga serializzato nella richiesta di trasporto
- `AfterSerialization`- Dopo che il messaggio di input è stato serializzato nella richiesta di trasporto
- `BeforeRetryLoop`- Prima di entrare nel ciclo di riprova

- `BeforeAttempt`- Primo hook chiamato inside retry loop
- `BeforeSigning`- Prima della firma della richiesta di trasporto
- `AfterSigning`- Dopo la firma della richiesta di trasporto
- `BeforeTransmit`- Prima dell'invio della richiesta di trasporto
- `AfterTransmit`- Dopo aver ricevuto la risposta di trasporto
- `BeforeDeserialization`- Prima che la risposta al trasporto venga deserializzata
- `AfterDeserialization`- Dopo aver riordinato la risposta di trasporto
- `AfterAttempt`- Ultimo hook chiamato inside retry loop
- `AfterExecution`- Ultimo hook chiamato durante l'esecuzione dell'operazione

È possibile implementare più interfacce in un unico intercettore per collegarsi a più fasi del ciclo di vita della richiesta.

Registrazione Interceptor

Gli intercettori vengono registrati quando si costruisce un client di servizio o quando si sostituisce la configurazione per un'operazione specifica. La registrazione varia a seconda che si desideri che l'interceptor si applichi a tutte le operazioni del client o solo a quelle specifiche.

Gli intercettori sono gestiti tramite un registro di intercettori che fornisce metodi per aggiungere e rimuovere gli intercettori. L'esempio seguente mostra un semplice interceptor che aggiunge un'intestazione Trace ID alle richieste in uscita prima del processo di firma: AWS X-Ray

```
type recursionDetection struct{}

func (recursionDetection) BeforeSigning(ctx context.Context, in
 *smithyhttp.InterceptorContext) error {
    if traceID := os.Getenv("_X_AMZN_TRACE_ID"); traceID != "" {
        in.Request.Header.Set("X-Amzn-Trace-Id", traceID)
    }
    return nil
}

// use it on the client
svc := s3.NewFromConfig(cfg, func(o *s3.Options) {
    o.Interceptors.AddBeforeSigning(recursionDetection{})
})
```

Il registro degli intercettori viene aggiunto a Client Options, che consente la configurazione degli intercettori per singola operazione:

```
// ... or use it per-operation
s3.ListBuckets(context.Background(), &s3.ListBucketsInput{
}, func(o *s3.Options) {
   o.Interceptors.AddBeforeSigning(recursionDetection{})
})
```

Configurazione globale degli intercettori

È inoltre possibile registrare gli intercettori a livello globale utilizzando la `config.LoadDefaultConfig` funzione con le opzioni appropriate `With*` per ogni tipo di intercettore. Questo applica l'interceptor a tutti i client di servizio creati da quella configurazione: AWS

```
type myExecutionInterceptor struct{}

func (*myExecutionInterceptor) AfterExecution(ctx context.Context, in
 *smithyhttp.InterceptorContext) error {
    // Add your custom logic here
    return nil
}

cfg, err := config.LoadDefaultConfig(context.Background(),
    config.WithAfterExecution(&myExecutionInterceptor{}))
if err != nil {
    panic(err)
}

// every service client created from the above config
// will include this interceptor
svc := s3.NewFromConfig(cfg)
```

Registrazione

AWS SDK per Go Dispone di funzionalità di registrazione che consentono all'applicazione di abilitare le informazioni di debug per il debug e la diagnosi di problemi o errori nelle richieste. L'interfaccia Logger e ClientLogModei componenti principali a vostra disposizione per determinare come e cosa devono essere registrati dai client.

Logger

Quando si costruisce un Config, LoadDefaultConfigl'utilizzo di un `Logger` valore predefinito è configurato per inviare messaggi di registro all'errore standard del processo (stderr). Un logger personalizzato che soddisfa l'interfaccia Logger può essere passato come argomento a racchiudendolo con config. `LoadDefaultConfig` WithLogger.

Ad esempio, per configurare i nostri clienti in modo che utilizzino i nostri`applicationLogger`:

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

Ora i client configurati utilizzando il build `aws.Config` invieranno messaggi di registro `aapplicationLogger`.

Logger sensibili al contesto

Un'implementazione di Logger può implementare l'interfaccia opzionale. ContextLogger I logger che implementano questa interfaccia avranno i loro `WithContext` metodi richiamati nel contesto corrente. Ciò consente alle implementazioni di registrazione di restituirne una nuova in `Logger` grado di scrivere metadati di registrazione aggiuntivi in base ai valori presenti nel contesto.

ClientLogMode

Per impostazione predefinita, i client di servizio non producono messaggi di registro. Per configurare i client per l'invio di messaggi di registro a scopo di debug, usa il ClientLogModemember on. `Config` `ClientLogMode`può essere impostato per abilitare la messaggistica di debug per:

- Firma versione 4 (SigV4) Firma
- Richiedi nuovi tentativi
- Richieste HTTP
- Risposte HTTP

Ad esempio, per abilitare la registrazione delle richieste e dei tentativi HTTP:

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

Vedi ClientLogModele diverse modalità di registro del client disponibili.

Tracce e metriche

AWS SDK per Go È progettato per registrare gli intervalli di traccia e le metriche lato client per le operazioni SDK. Per impostazione predefinita, i client utilizzano implementazioni no-op sia per il tracciamento che per le metriche, il che significa che non viene raccolto alcun dato a meno che non si configuri un provider.

I client di servizio hanno due opzioni di configurazione per l'osservabilità:

- TracerProvider— Punto di ingresso per la creazione di traccianti e la registrazione degli intervalli di traccia dei client.
- MeterProvider— Punto di ingresso per la creazione di contatori e la registrazione delle metriche lato client.

Sebbene ispirate alle specifiche OpenTelemetry (OTel), queste APIs sono definite indipendentemente in. `smithy-go` L'SDK fornisce moduli adattatori per collegare implementazioni OTel SDK concrete alle interfacce del provider dell'SDK.

Tracciamento

Utilizzate il modulo adattatore smithyoteltracing per connettere un client di servizio. OTel `trace.TracerProvider`

L'esempio seguente mostra come configurare il tracciamento per un client Amazon S3:

```
import (
    "github.com/aws/aws-sdk-go-v2/service/s3"
    "github.com/aws/smithy-go/tracing/smithyoteltracing"
    "go.opentelemetry.io/otel/trace"
)

// provider is an OTel trace.TracerProvider that you have configured.
var provider trace.TracerProvider

svc := s3.NewFromConfig(cfg, func(o *s3.Options) {
    o.TracerProvider = smithyoteltracing.Adapt(provider)
})
```

Le operazioni SDK sono strumentate con una gerarchia di intervalli che copre i componenti di alto livello del ciclo di vita dell'operazione, come la serializzazione delle richieste, la firma e il ciclo di riprova.

Metriche

Utilizzate il modulo adattatore smithyotelmetrics per connettere An a un client di servizio. OTel `metric.MeterProvider`

L'esempio seguente mostra come configurare i parametri per un client Amazon S3:

```
import (
    "github.com/aws/aws-sdk-go-v2/service/s3"
    "github.com/aws/smithy-go/metrics/smithyotelmetrics"
    "go.opentelemetry.io/otel/metric"
)

// provider is an OTel metric.MeterProvider that you have configured.
var provider metric.MeterProvider

svc := s3.NewFromConfig(cfg, func(o *s3.Options) {
    o.MeterProvider = smithyotelmetrics.Adapt(provider)
})
```

Parametri supportati

I client SDK raccolgono le seguenti metriche:

Nome parametro	Unità	Tipo	Description
`client.call.duration`	s	Istogramma	Durata complessiva della chiamata, compresi i nuovi tentativi e il tempo necessario per inviare o ricevere la richiesta e il messaggio di risposta.
`client.call.attempts`	{tentativo}	Monotonic Counter	Il numero di tentativi per una singola operazione.
`client.call.errors`	{errore	Monotonic Counter	Il numero di errori per un'operazione.

Nome parametro	Unità	Tipo	Description
`client.call.attempt_duration`	s	Istogramma	L'ora necessaria per connettersi al servizio, inviare la richiesta e recuperare il codice di stato HTTP e le intestazioni (incluso il tempo in coda in attesa di invio).
`client.call.resolve_endpoint_duration`	s	Istogramma	L'ora necessaria per risolvere un endpoint (endpoint resolver, non DNS) per la richiesta.
`client.call.deserialization_duration`	s	Istogramma	Il momento necessario per deserializzare il corpo del messaggio.
`client.call.auth.signing_duration`	s	Istogramma	L'ora in cui firmare una richiesta.
`client.call.auth.resolve_identity_duration`	s	Istogramma	Il momento necessario per acquisire un'identità (AWS credenziali, token al portatore, ecc.) da un provider di identità.

I seguenti attributi (dimensioni) sono inclusi in ogni metrica, ove applicabile:

- `rpc.service`— Il nome del servizio.
- `rpc.method`— Il nome dell'operazione.
- `exception.type`— Il tipo di errore (incluso in`client.call.errors`).
- `auth.scheme_id`— Lo schema di autenticazione (incluso nelle metriche relative all'autenticazione).

Metriche del client HTTP

Il client HTTP dell'SDK raccoglie le seguenti metriche aggiuntive relative al ciclo di vita della connessione HTTP sottostante:

Nome parametro	Unità	Tipo	Description
`client.http.connections.acquire_duration`	s	Istogramma	Il tempo impiegato da una richiesta per acquisire una connessione.
`client.http.connections.dns_lookup_duration`	s	Istogramma	Il tempo necessario per eseguire una ricerca DNS.
`client.http.connections.tls_handshake_duration`	s	Istogramma	Il tempo necessario per eseguire un handshake TLS.
`client.http.connections.usage`	{conne one}	UpDownCounter	Lo stato attuale delle connessioni nel pool. Utilizza una `state` dimensione con un valore di `idle` o `acquired`.
`client.http.do_request_duration`	s	Istogramma	Il tempo totale impiegato per eseguire la richiesta HTTP.
`client.http.time_to_first_byte`	s	Istogramma	Il tempo che intercorre tra l'invio della richiesta e la ricezione del primo byte di risposta.

Tentativi e timeout

AWS SDK per Go Consente di configurare il comportamento di ripetizione delle richieste ai servizi HTTP. Per impostazione predefinita, i client di servizio utilizzano Retry.Standard come retryer predefinito. Se la configurazione o il comportamento predefiniti non soddisfano i requisiti dell'applicazione, è possibile modificare la configurazione di retryer o fornire un'implementazione retryer personalizzata.

AWS SDK per Go Fornisce un'interfaccia AWS.Retryer che definisce l'insieme di metodi richiesti da un'implementazione Retry per l'implementazione. L'SDK fornisce due implementazioni per i tentativi: retry.standard e aws. NoOpRetryer.

Retrizzatore standard

Retry.Standard retryer è l'`aws.Retryer`implementazione predefinita utilizzata dai client SDK. Il retryer standard è un retryer a velocità limitata con un numero massimo di tentativi configurabile e la possibilità di ottimizzare la politica di annullamento della richiesta.

La tabella seguente definisce i valori predefiniti per questo retryer:

Proprietà	Predefinita
Numero massimo di tentativi	3
Ritardo massimo di back off	20 secondi

Quando si verifica un errore riutilizzabile durante l'invocazione della richiesta, il sistema standard utilizzerà la configurazione fornita per ritardare e successivamente riprovare la richiesta. I nuovi tentativi aumentano la latenza complessiva della richiesta ed è necessario configurare retryer se la configurazione predefinita non soddisfa i requisiti dell'applicazione.

Consultate la documentazione del pacchetto retry per i dettagli su quali errori sono considerati riutilizzabili dall'implementazione standard di retryer.

NopRetryer

Le leggi. NopRetryerè un'`aws.Retryer`implementazione che viene fornita se si desidera disabilitare tutti i tentativi di nuovo tentativo. Quando si richiama l'operazione di un client di servizio, questo retryer consente di tentare la richiesta una sola volta e qualsiasi errore risultante verrà restituito all'applicazione chiamante.

Personalizzazione del comportamento

L'SDK fornisce una serie di utilità di supporto che completano un'`aws.Retryer`implementazione e restituisce il retryer fornito con il comportamento di riprova desiderato. È possibile sostituire il retryer predefinito per tutti i client, per client o per operazione, a seconda dei requisiti delle applicazioni. Per vedere altri esempi che mostrano come eseguire questa operazione, consulta gli esempi di documentazione del pacchetto retry.

> **Warning**
>
> Se si specifica un'`aws.Retryer`implementazione globale utilizzando`config.WithRetryer`, è necessario assicurarsi di restituire una nuova istanza di `aws.Retryer` ogni chiamata. In questo modo non creerai un bucket di token di riprova globale su tutti i client di servizio.

Limitazione del numero massimo di tentativi

Si usa retry. AddWithMaxAttemptsper completare un'`aws.Retryer`implementazione per impostare il numero massimo di tentativi sul valore desiderato.

> **Warning**
>
> L'utilizzo `retry.AddWithMaxAttempts` con un valore pari a zero consentirà all'SDK di riprovare tutti gli errori riproducibili finché la richiesta non avrà esito positivo o non verrà restituito un errore non riproducibile. Consentire all'SDK di riprovare all'infinito può comportare carichi di lavoro inutili e cicli di fatturazione gonfiati.

```
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"

// ...

// MaxAttempts will be infinite (will retry indefinitely)
cfg, err := config.LoadDefaultConfig(context.TODO(), config.WithRetryer(func()
 aws.Retryer {
    return retry.AddWithMaxAttempts(retry.NewStandard(), 0)
}))
if err != nil {
    return err
}

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

Tieni presente che l'istanziazione di un retryer utilizzando opzioni funzionali per l'impostazione diretta avrà un comportamento leggermente diverso. MaxAttempts Più specificamente, l'impostazione di un valore inferiore o uguale a zero farà sì che il retryer utilizzi il massimo predefinito di 3 tentativi, anziché riprovare all'infinito:

```
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"

// ...

// MaxAttempts will default to 3
cfg, err := config.LoadDefaultConfig(context.TODO(), config.WithRetryer(func()
 aws.Retryer {
    return retry.NewStandard(func(o *retry.StandardOptions) {
        o.MaxAttempts = 0
    })
}))
if err != nil {
    return err
}

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

Limitazione del ritardo massimo di backoff

Si usa retry. AddWithMaxBackoffDelayper completare un'`aws.Retryer`implementazione e limitare il ritardo massimo consentito tra un tentativo e l'altro di una richiesta fallita.

Ad esempio, è possibile utilizzare il codice seguente per eseguire il wrapping del client retryer standard con un ritardo massimo desiderato di cinque secondi:

```
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)
```

Riprova i codici di errore API aggiuntivi

Si utilizza retry. AddWithErrorCodesper completare un'`aws.Retryer`implementazione e includere codici di errore API aggiuntivi che dovrebbero essere considerati riutilizzabili.

Ad esempio, puoi utilizzare il codice seguente per inserire il client retryer standard in modo da includere l'eccezione Amazon `NoSuchBucketException` S3 come riutilizzabile.

```
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)
```

Limitazione della tariffa lato client

AWS SDK per Go Introduce un nuovo meccanismo di limitazione della velocità lato client nella politica di ripetizione dei tentativi standard per allinearsi al comportamento moderno. SDKs Questo comportamento è controllato dal campo delle opzioni di un retryer. RateLimiter

A RateLimiter funziona come un token bucket con una capacità prestabilita, laddove i tentativi di operazione falliti consumano i token. Un nuovo tentativo che tenta di consumare più token di quelli disponibili comporta un fallimento dell'operazione con a. QuotaExceededError

L'implementazione predefinita è parametrizzata come segue (come modificare ogni impostazione):

- una capacità di 500 (imposta il valore di RateLimiter on StandardOptions using) NewTokenRateLimit
- un nuovo tentativo causato da un timeout costa 10 token (impostato su) RetryTimeoutCost StandardOptions
- un nuovo tentativo causato da altri errori costa 5 token (impostato su) RetryCost StandardOptions
- un'operazione che riesce al primo tentativo aggiunge 1 token (impostato su) NoRetryIncrement StandardOptions
  - le operazioni che hanno esito positivo al secondo o successivo tentativo non aggiungono nuovamente alcun token

Se ritieni che il comportamento predefinito non soddisfi le esigenze dell'applicazione, puoi disabilitarlo con RateLimit.None.

Esempio: limitatore di velocità modificato

```
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
    })
}))
```

Esempio: nessun limite di velocità utilizzando 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
    })
}))
```

Timeout

Si utilizza il pacchetto context per impostare timeout o scadenze quando si richiama un'operazione del client di servizio. Usa il contesto. WithDeadlineper riassumere il contesto dell'applicazione e fissare una scadenza a un'ora specifica entro la quale l'operazione richiamata deve essere completata. Per impostare un timeout dopo un determinato `time.Duration` contesto di utilizzo. WithTimeout. L'SDK passa il fornito `context.Context` al client di trasporto HTTP quando richiama un'API di servizio. Se il contesto passato all'SDK viene annullato o viene annullato durante l'invocazione dell'operazione, l'SDK non ritenterà ulteriormente la richiesta e tornerà all'applicazione chiamante. È necessario gestire l'annullamento del contesto in modo appropriato nell'applicazione nei casi in cui il contesto fornito all'SDK sia stato annullato.

Impostazione di un timeout

L'esempio seguente mostra come impostare un timeout per l'operazione di un client di servizio.

```
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
}
```

Usando il AWS SDK per Go

Scopri i metodi di programmazione più comuni e consigliati per AWS SDK per Go utilizzarli nelle tue applicazioni.

Argomenti

- Creazione di un client di servizio
- Operazioni di servizio di chiamata
- Utilizzo simultaneo dei client di servizio
- Utilizzo di Operation Paginators
- Usare Waiters
- Gestione degli errori nella AWS SDK per Go V2

Creazione di un client di servizio

I client di servizio possono essere creati utilizzando `NewFromConfig` le funzioni `New` o disponibili nel pacchetto Go del client di servizio. Ogni funzione restituirà un tipo di `Client` struttura contenente i metodi per richiamare il servizio APIs. `NewFromConfig`Ciascuna fornisce lo stesso set di opzioni configurabili per la creazione di un client di servizio, ma fornisce modelli di costruzione leggermente diversi che esamineremo nelle sezioni seguenti. `New`

NewFromConfig

`NewFromConfig`la funzione fornisce un'interfaccia coerente per la creazione di client di servizio utilizzando AWS.config. An `aws.Config` può essere caricato utilizzando il file di configurazione. LoadDefaultConfig. Per ulteriori informazioni sulla costruzione di un`aws.Config`, vedereConfigurare l'SDK. L'esempio seguente mostra come costruire un client di servizio Amazon S3 utilizzando `aws.Config` la funzione and`NewFromConfig`:

```
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)
```

Ignorare la configurazione

`NewFromConfig`può accettare uno o più argomenti funzionali che possono modificare la struttura di configurazione `Options` di un client. Ciò consente di effettuare sostituzioni specifiche, come la modifica della regione o la modifica di opzioni specifiche del servizio come l'opzione Amazon S3. `UseAccelerate` Per esempio:

```
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
    })
```

Le sostituzioni al `Options` valore del client sono determinate dall'ordine in cui vengono assegnati gli argomenti funzionali. `NewFromConfig`

Novità

> **Note**
>
> Newè considerata una forma più avanzata di costruzione del cliente. Si consiglia di utilizzarlo `NewFromConfig` per la costruzione del cliente, in quanto consente la costruzione utilizzando

la `aws.Config` struttura. Ciò elimina la necessità di creare un'istanza di `Options` struct per ogni client di servizio richiesto dall'applicazione.

Newfunction è un costruttore di client che fornisce un'interfaccia per la costruzione di client utilizzando solo la `Options` struttura dei pacchetti client per definire le opzioni di configurazione del client. Ad esempio, per creare un client Amazon S3 utilizzando: `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,
 "")),
    })
```

Ignorare la configurazione

Newpuò accettare uno o più argomenti funzionali che possono modificare la struttura di configurazione `Options` di un client. Ciò consente di effettuare sostituzioni specifiche, come la modifica della regione o la modifica di opzioni specifiche del servizio come l'opzione Amazon S3. `UseAccelerate` Per esempio:

```
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
    })
```

Le sostituzioni al `Options` valore del client sono determinate dall'ordine in cui vengono assegnati gli argomenti funzionali. `New`

Operazioni di servizio di chiamata

Dopo aver creato un'istanza del client di servizio, è possibile utilizzarla per richiamare le operazioni di un servizio. Ad esempio, per chiamare l'operazione Amazon S3: `GetObject`

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

Quando richiami un'operazione di servizio, l'SDK convalida in modo sincrono l'input, serializza la richiesta, la firma con le tue credenziali, la invia ad AWS e quindi deserializza una risposta o un errore. Nella maggior parte dei casi, puoi chiamare direttamente le operazioni di assistenza. Ogni metodo client per le operazioni di servizio restituirà una struttura di risposta all'operazione e un tipo di interfaccia di errore. È sempre necessario controllare il `error` tipo per determinare se si è verificato un errore prima di tentare di accedere alla struttura di risposta dell'operazione di servizio.

Passaggio di parametri a un'operazione di servizio

Ogni metodo di funzionamento del servizio accetta un valore Context.context che può essere utilizzato per impostare scadenze per le richieste che verranno rispettate dall'SDK. Inoltre, ogni operazione di servizio utilizzerà una `<OperationName>Input` struttura presente nel rispettivo pacchetto Go del servizio. Si passano i parametri di input dell'API utilizzando l'operazione input struct.

Le strutture di input delle operazioni possono avere parametri di input come i tipi standard Go numerici, booleani, stringhe, map ed elenchi. Nelle operazioni API più complesse, un servizio potrebbe avere una modellazione più complessa dei parametri di input. Questi altri tipi, come le strutture specifiche del servizio e i valori enum, si trovano nel pacchetto `types` Go del servizio.

Inoltre, i servizi potrebbero distinguere tra il valore predefinito di un tipo Go e se il valore è stato impostato o meno dall'utente. In questi casi, i parametri di input potrebbero richiedere il passaggio di un riferimento puntatore al tipo in questione. Per i tipi Go standard come numerici, booleani e stringhe, nell'aws sono `<Type>` disponibili `From<Type>` comode funzioni per facilitare questa conversione. Ad esempio, aws.String può essere utilizzato per convertire a in un `*string` tipo per `string` i parametri di input che richiedono un puntatore a una stringa. Al contrario, aws.ToStringpuò essere usato per trasformare a in un `string` while `*string` fornendo protezione dalla dereferenziazione di un puntatore nullo. Le `To<Type>` funzioni sono utili nella gestione delle risposte di servizio.

Diamo un'occhiata a un esempio di come possiamo usare un client Amazon S3 per chiamare l'`GetObject`API e costruire il nostro input utilizzando il `types` pacchetto e gli helper. `aws.<Type>`

```
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,
    })
```

Ignorare le opzioni del client per Operation Call

Analogamente a come è possibile modificare le opzioni operative del client durante la costruzione di un client utilizzando argomenti funzionali, le opzioni del client possono essere modificate nel momento in cui viene chiamato il metodo operativo fornendo uno o più argomenti funzionali al metodo

operativo del servizio. Questa azione è sicura dal punto di vista della concorrenza e non influirà su altre operazioni simultanee sul client.

Ad esempio, per sovrascrivere la regione del client da «us-west-2" a «us-east-1":

```
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"
    })
```

Gestione delle risposte operative

A ogni operazione di servizio è associata una struttura di output che contiene i membri della risposta operativa del servizio. La struttura di output segue il seguente schema di denominazione. `<OperationName>Output` Alcune operazioni potrebbero non avere membri definiti per l'output dell'operazione. Dopo aver chiamato un'operazione di servizio, è necessario controllare sempre il tipo di `error` argomento restituito per determinare se si è verificato un errore durante la chiamata dell'operazione di servizio. Gli errori restituiti possono variare da errori di convalida dell'input lato client a risposte di errore lato servizio restituite al client. Non è necessario accedere alla struttura di output dell'operazione nel caso in cui il client restituisca un errore diverso da zero.

Ad esempio, per registrare un errore operativo e tornare prematuramente dalla funzione chiamante:

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

Per ulteriori informazioni sulla gestione degli errori, incluso come verificare la presenza di tipi di errore specifici, consulta TODO

Risposte con `io.ReadCloser`

Alcune operazioni API restituiscono una struttura di risposta che contiene un membro di output che è un`io.ReadCloser`. Questo sarà il caso delle operazioni API che espongono alcuni elementi del loro output nel corpo della risposta HTTP stessa.

Ad esempio, l'`GetObject`operazione Amazon S3 restituisce una risposta il cui `Body` membro è un `io.ReadCloser` per accedere al payload dell'oggetto.

> **Warning**
>
> DEVI SEMPRE utilizzare `Close()` tutti i membri `io.ReadCloser` di output, indipendentemente dal fatto che tu ne abbia consumato il contenuto. In caso contrario, si possono verificare perdite di risorse e potenzialmente creare problemi con la lettura dei corpi di risposta per le operazioni richiamate in futuro.

```
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
    }
```

Metadati di risposta

Tutte le strutture di output delle operazioni di servizio includono un `ResultMetadata` membro di tipo Middleware.METADATA. `middleware.Metadata`viene utilizzato dal middleware SDK per fornire informazioni aggiuntive da una risposta di servizio non modellata dal servizio. Ciò include metadati

come. `RequestID` Ad esempio, per recuperare la risposta `RequestID` associata a un servizio e assistere AWS Support nella risoluzione dei problemi di una richiesta:

```
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)
```

Utilizzo simultaneo dei client di servizio

È possibile creare goroutine che utilizzano contemporaneamente lo stesso client di servizio per inviare più richieste. Puoi usare un client di servizio con tutte le goroutine che desideri.

Nell'esempio seguente, un client di servizio Amazon S3 viene utilizzato in più goroutine. Questo esempio carica contemporaneamente due oggetti in un bucket Amazon S3.

```
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))
    }
```

Utilizzo di Operation Paginators

In genere, quando si recupera un elenco di elementi, potrebbe essere necessario verificare la struttura di output di un token o di un marker per confermare se il AWS servizio ha restituito tutti i risultati della richiesta. Se il token o il marker è presente, lo si utilizza per richiedere la pagina successiva di risultati. Invece di gestire questi token o marker, puoi utilizzare i tipi di impaginatori disponibili nel pacchetto di servizi.

Gli helper Paginator sono disponibili per le operazioni di servizio supportate e sono disponibili nel pacchetto Go del client di servizio. Per costruire un impaginatore per un'operazione supportata, usa la funzione. `New<OperationName>Paginator` Le funzioni di costruzione di Paginator richiedono il servizio`Client`, i parametri di `<OperationName>Input` input dell'operazione e un set opzionale di argomenti funzionali che consentono di configurare altre impostazioni opzionali dell'impaginatore.

Il tipo di paginatore dell'operazione restituita offre un modo conveniente per ripetere un'operazione impaginata fino a raggiungere l'ultima pagina o trovare gli elementi che l'applicazione stava cercando. Un tipo di impaginatore ha due metodi: e. `HasMorePages` `NextPage` `HasMorePages`restituisce un valore booleano `true` se la prima pagina non è stata recuperata o se sono disponibili pagine aggiuntive da recuperare utilizzando l'operazione. Per recuperare la prima o le pagine successive dell'operazione, è necessario richiamare l'`NextPage`operazione. `NextPage`prende `context.Context` e restituisce l'output dell'operazione e l'eventuale errore corrispondente. Analogamente ai parametri di ritorno del metodo operativo client, l'errore restituito deve essere sempre verificato prima di tentare di utilizzare la struttura di risposta restituita. Consultare Gestione delle risposte operative.

L'esempio seguente utilizza l'`ListObjectsV2`impaginatore per elencare fino a tre pagine di chiavi oggetto relative all'operazione. `ListObjectV2` Ogni pagina è composta da un massimo di 10 chiavi, definite dall'opzione `Limit` paginator.

```
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++
    }
```

Analogamente al metodo operativo del client, le opzioni del client come la regione di richiesta possono essere modificate fornendo uno o più argomenti funzionali a. `NextPage` Per ulteriori informazioni sulla sovrascrittura delle opzioni del client quando si chiama un'operazione, vedereIgnorare le opzioni del client per Operation Call.

Usare Waiters

Quando si interagisce con una risorsa asincrona, spesso è necessario attendere AWS APIs che una particolare risorsa diventi disponibile per eseguire ulteriori azioni su di essa.

Ad esempio, l'API Amazon `CreateTable` DynamoDB ritorna immediatamente con TableStatus un di CREATING e non puoi richiamare operazioni di lettura o scrittura fino a quando lo stato della tabella non è passato a. `ACTIVE`

Scrivere una logica per controllare continuamente lo stato della tabella può essere complicato e soggetto a errori. I camerieri aiutano a semplificare la situazione e sono semplici APIs e gestiscono i sondaggi al posto tuo.

Ad esempio, è possibile utilizzare i camerieri per verificare se una tabella DynamoDB è stata creata ed è pronta per un'operazione di scrittura.

```
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")
```

Ignorare la configurazione del cameriere

Per impostazione predefinita, l'SDK utilizza il valore di ritardo minimo e massimo configurati con valori ottimali definiti da AWS services for different. APIs È possibile ignorare la configurazione del cameriere fornendo opzioni funzionali durante la costruzione del cameriere o quando si richiama l'operazione di cameriere.

Ad esempio, per ignorare la configurazione del cameriere durante la costruzione del cameriere

```
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
    })
```

La `Wait` funzione di ogni cameriere include anche opzioni funzionali. Analogamente all'esempio precedente, è possibile sovrascrivere la configurazione del cameriere per richiesta. `Wait`

```
// 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")
```

Sostituzioni della configurazione avanzata del cameriere

È inoltre possibile personalizzare il comportamento predefinito del cameriere fornendo una funzione riutilizzabile personalizzata. Le opzioni specifiche per il cameriere consentono inoltre di personalizzare i middleware operativi. `APIOptions`

Ad esempio, per configurare le sostituzioni avanzate dei camerieri.

```
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
    })
```

Gestione degli errori nella AWS SDK per Go V2

AWS SDK per Go Restituisce errori che soddisfano il tipo di `error` interfaccia Go. È possibile utilizzare il `Error()` metodo per ottenere una stringa formattata del messaggio di errore SDK senza alcuna gestione speciale. Gli errori restituiti dall'SDK possono implementare un metodo. `Unwrap` Il `Unwrap` metodo viene utilizzato dall'SDK per fornire informazioni contestuali aggiuntive sugli errori, fornendo al contempo l'accesso all'errore o alla catena di errori sottostante. Il `Unwrap` metodo deve essere usato con Errors.as per gestire l'unwrapping delle catene di errori.

È importante che l'applicazione verifichi se si è verificato un errore dopo aver richiamato una funzione o un metodo in grado di restituire un tipo di interfaccia. `error` La forma più semplice di gestione degli errori è simile al seguente esempio:

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

Errori di registrazione

La forma più semplice di gestione degli errori consiste tradizionalmente nel registrare o stampare il messaggio di errore prima di tornare o uscire dall'applicazione.

```
import "log"

// ...

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

Errori del client di servizio

L'SDK include tutti gli errori restituiti dai client di servizio con la fucina. OperationErrortipo di errore. `OperationError`fornisce informazioni contestuali sul nome e sull'operazione del servizio associati a un errore sottostante. Queste informazioni possono essere utili per le applicazioni che eseguono batch di operazioni su uno o più servizi, con un meccanismo centralizzato di gestione degli errori. L'applicazione può utilizzare `errors.As` per accedere a questi `OperationError` metadati.

```
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
}
```

Risposte di errore API

Le operazioni di servizio possono restituire tipi di errore modellati per indicare errori specifici. Questi tipi modellati possono essere utilizzati con `errors.As` per decomprimere e determinare se l'errore dell'operazione è dovuto a un errore specifico. Ad esempio, Amazon S3 `CreateBucket` può restituire un BucketAlreadyExistserrore se esiste già un bucket con lo stesso nome.

Ad esempio, per verificare se un errore era un `BucketAlreadyExists` errore:

```
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
}
```

Tutti gli errori di risposta dell'API di servizio implementano la fucina. APIErrortipo di interfaccia. Questa interfaccia può essere utilizzata per gestire risposte di errore di servizio sia modellate che non modellate. Questo tipo fornisce l'accesso al codice di errore e al messaggio restituiti dal servizio. Inoltre, questo tipo indica se l'errore è dovuto al client o al server, se noto.

```
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
}
```

Recupero degli identificatori della richiesta

Quando si collabora con AWS Support, è possibile che venga richiesto di fornire l'identificatore della richiesta che identifica la richiesta che si sta tentando di risolvere. Puoi usare http. ResponseErrore usa il `ServiceRequestID()` metodo per recuperare l'identificatore della richiesta associato alla risposta all'errore.

```
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
}
```

Identificatori di richiesta Amazon S3

Le richieste Amazon S3 contengono identificatori aggiuntivi che possono essere utilizzati per assistere il AWS Supporto nella risoluzione dei problemi della richiesta. Puoi usare s3.ResponseErrore chiama `ServiceRequestID()` e `ServiceHostID()` recupera l'ID della richiesta e l'ID dell'host.

```
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
}
```

Usa la AWS SDK per Go v2 con i servizi AWS

Per effettuare chiamate a un AWS servizio, è necessario innanzitutto creare un'istanza del client di servizio. Un client di servizio fornisce un accesso di basso livello a ogni azione API per quel servizio. Ad esempio, crei un client di servizio Amazon S3 per effettuare chiamate verso Amazon S3. APIs

Quando chiami le operazioni di servizio, trasmetti i parametri di input come struttura. Una chiamata riuscita produrrà una struttura di output contenente la risposta dell'API del servizio. Ad esempio, dopo aver chiamato con successo un'azione di creazione del bucket di Amazon S3, l'azione restituisce una struttura di output con la posizione del bucket.

Per l'elenco dei client del servizio, compresi i relativi metodi e parametri, consulta l'AWS SDK per Go API Reference.

Protezione dell'integrità dei dati con checksum

Amazon Simple Storage Service (Amazon S3) Simple Storage Service (Amazon S3) offre la possibilità di specificare un checksum quando carichi un oggetto. Quando specifichi un checksum, questo viene memorizzato con l'oggetto e può essere convalidato quando l'oggetto viene scaricato.

I checksum forniscono un ulteriore livello di integrità dei dati durante il trasferimento dei file. Con i checksum, è possibile verificare la coerenza dei dati confermando che il file ricevuto corrisponde al file originale. Per ulteriori informazioni sui checksum con Amazon S3, consulta la Guida per l'utente di Amazon Simple Storage Service, che include gli algoritmi supportati.

Hai la flessibilità di scegliere l'algoritmo più adatto alle tue esigenze e lasciare che sia l'SDK a calcolare il checksum. In alternativa, puoi fornire un valore di checksum precalcolato utilizzando uno degli algoritmi supportati.

> **Note**
>
> A partire dalla versione v1.74.1 del modulo Amazon S3, l'SDK fornisce protezioni di integrità predefinite calcolando automaticamente un checksum per i caricamenti. `CRC32` L'SDK calcola questo checksum se non fornisci un valore di checksum precalcolato o se non specifichi un algoritmo che l'SDK deve utilizzare per calcolare un checksum.
> L'SDK fornisce anche impostazioni globali per la protezione dell'integrità dei dati che puoi impostare esternamente, come puoi leggere nella Guida di riferimento agli strumenti e agli strumenti.AWS SDKs

Discutiamo i checksum in due fasi di richiesta: caricamento di un oggetto e download di un oggetto.

Caricamento di un oggetto

Quando carichi un oggetto con il `putObject` metodo e fornisci un algoritmo di checksum, l'SDK calcola il checksum per l'algoritmo specificato.

Il seguente frammento di codice mostra una richiesta di caricamento di un oggetto con un checksum. `CRC32` Quando l'SDK invia la richiesta, calcola il `CRC32` checksum e carica l'oggetto. Amazon S3 convalida l'integrità del contenuto calcolando il checksum e confrontandolo con il checksum fornito dall'SDK. Amazon S3 memorizza quindi il checksum con l'oggetto.

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

Se non fornisci un algoritmo di checksum con la richiesta, il comportamento del checksum varia a seconda della versione dell'SDK che usi, come mostrato nella tabella seguente.

Comportamento del checksum quando non viene fornito alcun algoritmo di checksum

Versione del modulo Amazon S3 di AWS SDK per Go	Comportamento del checksum
Precedente alla v1.74.1	L'SDK non calcola automaticamente un checksum basato su CRC e lo fornisce nella richiesta.
v1.74.1 o versione successiva	L'SDK utilizza l'`CRC32`algoritmo per calcolare il checksum e lo fornisce nella richiesta. Amazon S3 convalida l'integrità del trasferimento calcolando il proprio `CRC32` checksum e lo confronta con il checksum fornito dall'SDK. Se i checksum corrispondono, il checksum viene salvato con l'oggetto.

Utilizza un valore di checksum precalcolato

Un valore di checksum precalcolato fornito con la richiesta disabilita il calcolo automatico da parte dell'SDK e utilizza invece il valore fornito.

L'esempio seguente mostra una richiesta con un checksum precalcolato. 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"),
 })
```

Se Amazon S3 determina che il valore del checksum non è corretto per l'algoritmo specificato, il servizio restituisce una risposta di errore.

Caricamenti in più parti

Puoi anche utilizzare i checksum con caricamenti in più parti.

AWS SDK per Go Fornisce due opzioni per utilizzare i checksum con caricamenti in più parti. La prima opzione utilizza il gestore dei trasferimenti che specifica l'algoritmo per il `CRC32` caricamento.

```
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,
     })
```

Se non fornisci un algoritmo di checksum quando utilizzi il gestore di trasferimento per i caricamenti, l'SDK calcola automaticamente un checksum in base all'algoritmo. `CRC32` L'SDK esegue questo calcolo per tutte le versioni dell'SDK.

La seconda opzione utilizza il client Amazon S3 per eseguire il caricamento in più parti. Se specifichi un checksum con questo approccio, devi specificare l'algoritmo da utilizzare all'avvio del caricamento. Inoltre, devi specificare l'algoritmo per ogni richiesta parte e fornire il checksum calcolato per ciascuna parte dopo che è stata caricata.

```
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)
    }
```

Donwload di un oggetto

Quando utilizzate il GetObjectmetodo per scaricare un oggetto, l'SDK convalida automaticamente il checksum quando il `ChecksumMode` campo di è impostato su. `GetObjectInput` `types.ChecksumModeEnabled`

La richiesta nel seguente frammento indica all'SDK di convalidare il checksum nella risposta calcolando il checksum e confrontando i valori.

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

Se l'oggetto non è stato caricato con un checksum, non viene effettuata alcuna convalida.

Utilizzo delle AWS SDK per Go utilità

AWS SDK per Go Include le seguenti utilità per aiutarti a utilizzare più facilmente i servizi. AWS Trova le utilità SDK nel relativo AWS pacchetto di servizi.

Utilità Amazon RDS

Autenticazione IAM

Il pacchetto auth fornisce utilità per la generazione di token di autenticazione per la connessione a istanze di database Amazon RDS MySQL e PostgreSQL. Utilizzando il BuildAuthTokenmetodo, si genera un token di autorizzazione del database fornendo l'endpoint del database, la regione, il nome utente e un aws. AWS CredentialProviderimplementazione che restituisce le credenziali IAM con l'autorizzazione a connettersi al database utilizzando l'autenticazione del database IAM. Per ulteriori informazioni sulla configurazione di Amazon RDS con l'autenticazione IAM, consulta le seguenti risorse della Amazon RDS Developer Guide:

- Abilitazione e disabilitazione dell'autenticazione del database IAM
- Creazione e utilizzo di una policy IAM per l'accesso al database IAM
- Creazione di un account di database utilizzando l'autenticazione IAM

L'esempio seguente mostra come generare un token di autenticazione per connettersi a un database Amazon RDS:

```
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 Utility

Firmatario CloudFront URL Amazon

Il firmatario CloudFront URL di Amazon semplifica il processo di creazione dei firmatari. URLs Un URL firmato include informazioni, come la data e l'ora di scadenza, che ti consentono di controllare l'accesso ai tuoi contenuti. URLs I firmati sono utili quando desideri distribuire contenuti tramite Internet, ma desideri limitare l'accesso a determinati utenti (ad esempio, agli utenti che hanno pagato una tariffa).

Per firmare un URL, crea un'`URLSigner`istanza con il tuo ID della coppia di CloudFront chiavi e la chiave privata associata. Quindi chiama il `SignWithPolicy` metodo `Sign` or e includi l'URL da firmare. Per ulteriori informazioni sulle coppie di CloudFront chiavi Amazon, consulta Creating CloudFront Key Pairs for Your Trusted Signers nella CloudFront Developer Guide.

L'esempio seguente crea un URL firmato valido per un'ora dopo la creazione.

```
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
}
```

Per ulteriori informazioni sull'utilità di firma, consulta il pacchetto sign nell' AWS SDK per Go API Reference.

Servizio di metadati di Amazon EC2 Instance

Puoi utilizzare il AWS SDK per Go per accedere ad Amazon EC2 Instance Metadata Service. Il pacchetto feature/ec2/imdsGo fornisce un tipo di client che può essere utilizzato per accedere ad Amazon EC2 Instance Metadata Service. Le `Client` operazioni associate possono essere utilizzate in modo simile agli altri client AWS di servizio forniti dall'SDK. Per ulteriori informazioni su come configurare l'SDK e utilizzare i client di servizio, consulta Configurare l'SDK e. Usa la AWS SDK per Go v2 con i servizi AWS

Il client può aiutarti a recuperare facilmente informazioni sulle istanze su cui vengono eseguite le tue applicazioni, come la AWS regione o l'indirizzo IP locale. In genere, è necessario creare e inviare richieste HTTP per recuperare i metadati dell'istanza. Invece, crea un servizio `imds.Client` per accedere ad Amazon EC2 Instance Metadata Service utilizzando un client programmatico come altri AWS servizi.

Ad esempio, per creare un client:

```
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)
```

Quindi utilizza il client di servizio per recuperare informazioni da una categoria di metadati come `local-ipv4` (l'indirizzo IP privato dell'istanza).

```
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))
```

Per un elenco di tutte le categorie di metadati, consulta le categorie di metadati delle istanze nella Amazon EC2 User Guide.

Utilità Amazon S3

Gestori di trasferimenti Amazon S3

I gestori di upload e download di Amazon S3 possono suddividere oggetti di grandi dimensioni, in modo che possano essere trasferiti in più parti, in parallelo. In questo modo è facile riprendere i trasferimenti interrotti.

Gestione caricamenti Amazon S3

Il gestore di caricamento di Amazon S3 determina se un file può essere suddiviso in parti più piccole e caricato in parallelo. Puoi personalizzare il numero di caricamenti paralleli e la dimensione delle parti caricate.

L'esempio seguente utilizza Amazon S3 `Uploader` per caricare un file. L'utilizzo `Uploader` è simile all'`s3.PutObject()`operazione.

```
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,
})
```

Opzioni di configurazione

Quando si crea un'`Uploader`istanza utilizzando NewUploader, è possibile specificare diverse opzioni di configurazione per personalizzare il modo in cui gli oggetti vengono caricati. Le opzioni vengono sovrascritte fornendo uno o più argomenti a. `NewUploader` Le opzioni includono:

- `PartSize`— specifica la dimensione del buffer, in byte, di ogni parte da caricare. La dimensione minima per parte è di 5 MiB.
- `Concurrency`— specifica il numero di parti da caricare in parallelo.
- `LeavePartsOnError`— Indica se lasciare le parti caricate correttamente in Amazon S3.

Il `Concurrency` valore limita il numero simultaneo di caricamenti di parti che possono verificarsi per una determinata chiamata. `Upload` Non si tratta di un limite globale di concorrenza tra client. Modifica i valori `PartSize` e di `Concurrency` configurazione per trovare la configurazione ottimale. Ad esempio, i sistemi con connessioni a larghezza di banda elevata possono inviare parti più grandi e più caricamenti in parallelo.

Ad esempio, l'applicazione viene configurata `Uploader` con un'`Concurrency`impostazione di. `5` Se l'applicazione chiama poi `Upload` da due goroutine diverse, il risultato sono caricamenti `10` simultanei di parti (2 goroutine * 5). `Concurrency`

> **Warning**
>
> Si prevede che l'applicazione limiti le chiamate simultanee per evitare l'esaurimento delle risorse dell'applicazione. `Upload`

Di seguito è riportato un esempio per impostare la dimensione predefinita della parte durante la `Uploader` creazione:

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

Per ulteriori informazioni `Uploader` e sulle relative configurazioni, consulta Uploader nell' AWS SDK per Go API Reference.

PutObjectInput Body Field (io. ReadSeeker rispetto a IO.Reader)

Il `Body` campo della `s3.PutObjectInput` struttura è un tipo. `io.Reader` Tuttavia, questo campo può essere compilato con un tipo che soddisfi sia l'interfaccia che l'`io.ReadSeeker`io.ReaderAtinterfaccia per migliorare l'utilizzo delle risorse applicative dell'ambiente host. L'esempio seguente crea il tipo `ReadSeekerAt` che soddisfa entrambe le interfacce:

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

Per `io.Reader` i tipi, i byte del lettore devono essere memorizzati nel buffer prima di poter caricare la parte. Quando si aumenta il `Concurrency` valore `PartSize` o, la memoria (RAM) richiesta per il valore `Uploader` aumenta in modo significativo. La memoria richiesta è di circa *`PartSize`* `Concurrency`*. Ad esempio, se si specificano 100 MB per `PartSize` e 10 per`Concurrency`, è necessario almeno 1 GB.

Poiché un `io.Reader` tipo non può determinarne le dimensioni prima di leggerne i byte, `Uploader` non può calcolare quante parti verranno caricate. Di conseguenza, `Uploader` puoi raggiungere il limite di caricamento di Amazon S3 di 10.000 parti per file di grandi dimensioni se imposti un valore `PartSize` troppo basso. Se tenti di caricare più di 10.000 parti, il caricamento si interrompe e restituisce un errore.

Per `body` i valori che implementano il `ReadSeekerAt` tipo, `Uploader` non memorizza nel buffer il contenuto del corpo in memoria prima di inviarlo ad Amazon S3. `Uploader`calcola il numero previsto di parti prima di caricare il file su Amazon S3. Se il valore corrente di `PartSize` richiede più di 10.000 parti per caricare il file, `Uploader` aumenta il valore della dimensione della parte in modo che siano necessarie meno parti.

Gestione dei caricamenti non riusciti

Se un caricamento su Amazon S3 fallisce, per impostazione predefinita, `Uploader` utilizza l'operazione Amazon `AbortMultipartUpload` S3 per rimuovere le parti caricate. Questa funzionalità garantisce che i caricamenti non riusciti non consumino lo storage Amazon S3.

Puoi impostare su `LeavePartsOnError` true in modo che `Uploader` non elimini le parti caricate correttamente. Ciò è utile per riprendere i caricamenti parzialmente completati. Per operare sulle parti caricate, è necessario ottenere l'indicazione `UploadID` del caricamento non riuscito. L'esempio seguente mostra come utilizzare il tipo di interfaccia `manager.MultiUploadFailure` di errore per ottenere il`UploadID`.

```
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
}
```

Sovrascrivere le opzioni di caricamento per caricamento

È possibile sovrascrivere le `Uploader` opzioni durante la chiamata `Upload` fornendo uno o più argomenti al metodo. Queste sostituzioni sono modifiche sicure per la concorrenza e non influiscono sui caricamenti in corso o sulle successive chiamate al gestore. `Upload` Ad esempio, per sovrascrivere la configurazione per una richiesta di caricamento specifica: `PartSize`

```
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
})
```

Esempi

Caricare una cartella su Amazon S3

L'esempio seguente utilizza il `path/filepath` pacchetto per raccogliere in modo ricorsivo un elenco di file e caricarli nel bucket Amazon S3 specificato. Le chiavi degli oggetti Amazon S3 sono precedute dal percorso relativo del file.

```
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
```

```
}
```

Download Manager

Il gestore di Amazon S3 Downloader determina se un file può essere suddiviso in parti più piccole e scaricato in parallelo. È possibile personalizzare il numero di download paralleli e la dimensione delle parti scaricate.

Esempio: scaricare un file

L'esempio seguente utilizza Amazon S3 `Downloader` per scaricare un file. Using `Downloader` è simile a s3. GetObjectoperazione.

```
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"),
})
```

Il `downloadFile` parametro è un `io.WriterAt` tipo. L'`WriterAt`interfaccia consente di `Downloader` scrivere più parti del file in parallelo.

Opzioni di configurazione

Quando crei un'`Downloader`istanza, puoi specificare le opzioni di configurazione per personalizzare il modo in cui gli oggetti vengono scaricati:

- `PartSize`— specifica la dimensione del buffer, in byte, di ogni parte da scaricare. La dimensione minima per parte è di 5 MB.
- `Concurrency`— specifica il numero di parti da scaricare in parallelo.

Il `Concurrency` valore limita il numero simultaneo di download di parti che possono verificarsi per una determinata `Download` chiamata. Non si tratta di un limite globale di concorrenza tra client. Modifica i valori `PartSize` e di `Concurrency` configurazione per trovare la configurazione ottimale. Ad esempio, i sistemi con connessioni a larghezza di banda elevata possono ricevere parti più grandi e più download in parallelo.

Ad esempio, l'applicazione viene configurata `Downloader` con un di. `Concurrency` `5` L'applicazione chiama quindi `Download` da due goroutine diverse, il risultato saranno download `10` simultanei di parti (2 goroutine * 5). `Concurrency`

> **Warning**
>
> Si prevede che l'applicazione limiti le chiamate simultanee per evitare l'esaurimento delle risorse dell'applicazione. `Download`

Per ulteriori informazioni sulle `Downloader` altre opzioni di configurazione, consulta Manager.downloader nell'API Reference. AWS SDK per Go

Ignorare le opzioni del downloader per download

È possibile sovrascrivere le `Downloader` opzioni durante la chiamata `Download` fornendo uno o più argomenti funzionali al metodo. Queste sostituzioni sono modifiche sicure dal punto di vista della concorrenza e non influiscono sui caricamenti in corso o sulle successive chiamate al gestore. `Download` Ad esempio, per sovrascrivere la `PartSize` configurazione per una richiesta di caricamento specifica:

```
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
})
```

Esempi

Scarica tutti gli oggetti in un secchio

L'esempio seguente utilizza la paginazione per raccogliere un elenco di oggetti da un bucket Amazon S3. Quindi scarica ogni oggetto in un file locale.

```
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

GetBucketRegionè una funzione di utilità per determinare la posizione AWS della regione di un bucket Amazon S3. `GetBucketRegion`prende un client Amazon S3 e lo utilizza per determinare la posizione del bucket richiesto all'interno della AWS partizione associata alla regione configurata del client.

Ad esempio, per trovare la regione per il bucket: *`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)
```

Se non `GetBucketRegion` è in grado di risolvere la posizione di un Bucket, la funzione restituisce un tipo di BucketNotFounderrore come mostrato nell'esempio.

Input di streaming non ricercabile

Per operazioni API come `PutObject` and`UploadPart`, il client Amazon S3 si aspetta che il valore del parametro di `Body` input implementi l'interfaccia IO.Seeker per impostazione predefinita. L'`io.Seeker`interfaccia viene utilizzata dal client per determinare la lunghezza del valore da caricare e per calcolare l'hash del payload per la firma della richiesta. Se il valore del parametro `Body` di input non viene implementato`io.Seeker`, l'applicazione riceverà un errore.

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

È possibile modificare questo comportamento modificando il metodo operativo Middleware utilizzando le opzioni funzionali. L'APIOptionshelper With restituisce un'opzione funzionale per zero o più mutatori del middleware. Per disabilitare il client che calcola l'hash del payload e utilizzare Unsigned Payload, richiedi la firma aggiungi v4. 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

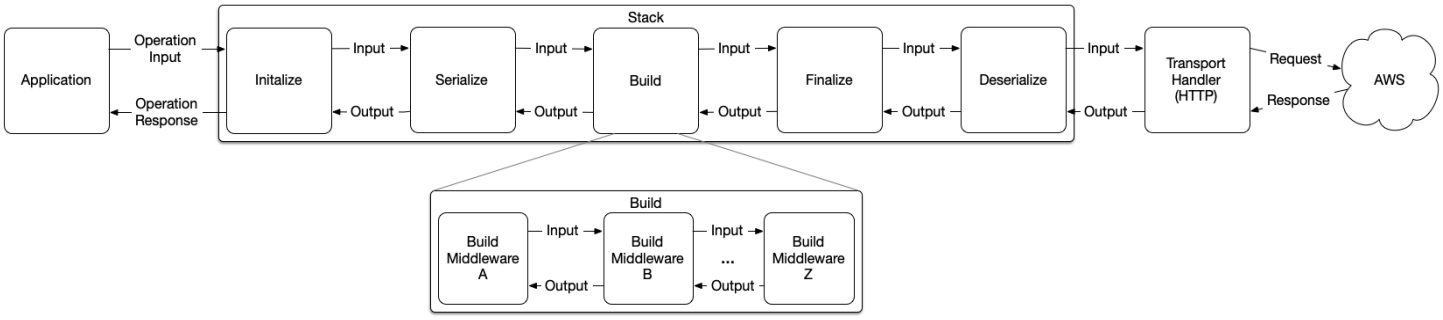
Amazon S3 richiede che venga fornita la lunghezza del contenuto per tutti gli oggetti caricati in un bucket. Poiché il parametro `Body` di input non implementa l'`io.Seeker`interfaccia, il client non sarà in grado di calcolare il `ContentLength` parametro per la richiesta. Il parametro deve essere fornito dall'applicazione. La richiesta avrà esito negativo se il `ContentLength` parametro non viene fornito.

Utilizza gli SDK Gestione caricamenti Amazon S3 per i caricamenti che non sono ricercabili e che non hanno una lunghezza nota.

Personalizzazione delle richieste del client AWS SDK per Go v2 con Middleware

> **Warning**
>
> La modifica della pipeline delle richieste del client può causare richieste non corrette/non valide o errori di applicazione imprevisti. Questa funzionalità è pensata per casi d'uso avanzati non forniti dall'interfaccia SDK per impostazione predefinita.

È possibile personalizzare le richieste dei AWS SDK per Go client registrando uno o più middleware nello stack di un'operazione di servizio. Lo stack è composto da una serie di passaggi: inizializzazione, serializzazione, compilazione, finalizzazione e deserializzazione. Ogni passaggio contiene zero o più middleware che operano sui tipi di input e output di quella fase. Il diagramma e la tabella seguenti forniscono una panoramica del modo in cui la richiesta e la risposta di un'operazione attraversano lo stack.



Fase dello stack	Descrizione
Inizializzazione	Prepara l'input e imposta i parametri predefiniti in base alle esigenze.
Serializza	Serializza l'input in un formato di protocollo adatto al livello di trasporto di destinazione.
Creazione	Allega metadati aggiuntivi all'input serializzato, ad esempio HTTP Content-Length.

Fase dello stack	Descrizione
Finalizza	Preparazione finale del messaggio, inclusi nuovi tentativi e autenticazione (firma SigV4).
Deserializza	Deserializza le risposte dal formato del protocollo in un tipo o errore strutturato.

Ogni middleware all'interno di un determinato passaggio deve avere un identificatore univoco, determinato dal metodo del middleware. `ID` Gli identificatori del middleware assicurano che solo un'istanza di un determinato middleware sia registrata in un passaggio e consentono l'inserimento di altri middleware relativi ad esso.

È possibile collegare il middleware degli step utilizzando uno o più metodi. `Insert` `Add` Si utilizza `Add` per collegare un middleware all'inizio di un passaggio specificando middleware.before come e middleware.after per collegarlo alla RelativePositionfine del passaggio. Si utilizza `Insert` per collegare un middleware a un passaggio inserendo il middleware relativo a un altro step middleware.

> **Warning**
>
> È necessario utilizzare il `Add` metodo per inserire in modo sicuro il middleware step personalizzato. L'utilizzo `Insert` crea una dipendenza tra il middleware personalizzato e il middleware a cui si sta inserendo rispetto. Il middleware all'interno di una fase di stack deve essere considerato opaco per evitare di interrompere le modifiche apportate all'applicazione.

Scrittura di un middleware personalizzato

Ogni fase dello stack ha un'interfaccia che è necessario soddisfare per collegare un middleware a un determinato passaggio. È possibile utilizzare una delle *`Step`*`MiddlewareFunc` funzioni fornite per soddisfare rapidamente questa interfaccia. La tabella seguente descrive i passaggi, la relativa interfaccia e la funzione di supporto che può essere utilizzata per soddisfare l'interfaccia.

Fase	Interfaccia	Funzione Helper
Inizializzazione	InitializeMiddleware	InitializeMiddlewareFunc

Fase	Interfaccia	Funzione Helper
Creazione	BuildMiddleware	BuildMiddlewareFunc
Serializza	SerializeMiddleware	SerializeMiddlewareFunc
Finalizzare	FinalizeMiddleware	FinalizeMiddlewareFunc
Deserializza	DeserializeMiddleware	DeserializeMiddlewareFunc

Gli esempi seguenti mostrano come scrivere un middleware personalizzato per popolare il membro Bucket delle chiamate `GetObject` API Amazon S3 se non ne viene fornito uno. Questo middleware verrà citato negli esempi seguenti per mostrare come collegare il middleware step allo stack.

```
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)
})
```

Collegamento del middleware a tutti i client

Puoi collegare il tuo middleware step personalizzato a ogni client aggiungendo il middleware utilizzando il membro del tipo AWS.config. `APIOptions` Gli esempi seguenti collegano il `defaultBucket` middleware a ogni client creato utilizzando l'oggetto dell'applicazione: `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)
```

Collegamento del middleware a un'operazione specifica

È possibile collegare il middleware Step personalizzato a una specifica operazione client modificando il `APIOptions` membro del client utilizzando l'elenco degli argomenti variadici per un'operazione. Gli esempi seguenti collegano il `defaultBucket` middleware a una chiamata operativa specifica di Amazon S3: `GetObject`

```
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)
})
```

Trasmissione dei metadati nella pila

In alcune situazioni, potresti scoprire che sono necessari due o più middleware per funzionare in tandem condividendo informazioni o stati. È possibile utilizzare Context.context per passare questi metadati utilizzando il middleware. WithStackValue. `middleware.WithStackValue`collega la coppia chiave-valore specificata al contesto fornito e limita in modo sicuro l'ambito allo stack attualmente in esecuzione. Questi valori con ambito dello stack possono essere recuperati da un contesto utilizzando il middleware. GetStackValuee fornendo la chiave utilizzata per memorizzare il valore corrispondente. Le chiavi devono essere comparabili ed è necessario definire i propri tipi come chiavi di contesto per evitare collisioni. Gli esempi seguenti mostrano come due middleware possono essere utilizzati `context.Context` per trasmettere informazioni all'interno dello stack.

```
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)
})
```

Metadati forniti dall'SDK

AWS SDK per Go Fornisce diversi valori di metadati che possono essere recuperati dal contesto fornito. Questi valori possono essere utilizzati per abilitare un middleware più dinamico che ne modifichi il comportamento in base al servizio, all'operazione o alla regione di destinazione in esecuzione. Alcune delle chiavi disponibili sono fornite nella tabella seguente:

Chiave	cane da riporto	Descrizione
ID del servizio	GetServiceID	Recupera l'identificatore del servizio per lo stack di esecuzione. Questo può essere confrontato con la costante del pacchetto client del servizio. `ServiceID`
OperationName	GetOperationName	Recupera il nome dell'oper azione per lo stack di esecuzione.
Logger	GetLogger	Recupera il logger che può essere utilizzato per registrare i messaggi dal middleware.

Passare i metadati allo stack

Puoi passare i metadati attraverso lo stack aggiungendo coppie di chiavi e valori di metadati utilizzando Middleware.metadata. Ogni passaggio del middleware restituisce una struttura di output, metadati e un errore. Il middleware personalizzato deve restituire i metadati ricevuti dalla chiamata al gestore successivo della fase. Ciò garantisce che i metadati aggiunti dal middleware downstream si propaghino all'applicazione che richiama l'operazione del servizio. I metadati risultanti sono accessibili all'applicazione richiamante tramite la forma di output dell'operazione tramite il membro della struttura. `ResultMetadata`

Gli esempi seguenti mostrano come un middleware personalizzato può aggiungere metadati restituiti come parte dell'output dell'operazione.

```
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)
```

Domande frequenti

Come posso configurare il client HTTP del mio SDK? Esistono linee guida o best practice?

Non siamo in grado di fornire indicazioni ai clienti su come configurare il flusso di lavoro HTTP nel modo più efficace per il loro particolare carico di lavoro. La risposta a questa domanda è il prodotto di un'equazione multivariata, con fattori di input tra cui, a titolo esemplificativo ma non esaustivo:

- l'impronta di rete dell'applicazione (TPS, throughput, ecc.)
- i servizi utilizzati
- le caratteristiche di calcolo della distribuzione
- la natura geografica della distribuzione
- il comportamento o le esigenze dell'applicazione stessa desiderati (SLAstempistiche, ecc.)

Come devo configurare i timeout operativi?

Proprio come la domanda precedente, dipende. Gli elementi da considerare qui includono quanto segue:

- Tutti i fattori sopra indicati relativi alla configurazione del client HTTP
- vincoli relativi alla tempistica delle applicazioni o agli SLA (ad esempio se sei tu stesso a fornire traffico ad altri consumatori)

La risposta a questa domanda non dovrebbe quasi MAI basarsi sulla pura osservazione empirica del comportamento a monte, ad esempio «Ho effettuato 1000 chiamate per questa operazione, ci sono voluti al massimo 5 secondi, quindi imposterò il timeout in base a quello con un fattore di sicurezza compreso tra 2 e 10 secondi». Le condizioni ambientali possono cambiare, i servizi possono peggiorare temporaneamente e questi tipi di ipotesi possono diventare errate senza preavviso.

Le richieste effettuate dall'SDK scadono o impiegano troppo tempo. Come posso risolvere il problema?

Non siamo in grado di fornire assistenza in caso di chiamate operative prolungate o scadute a causa del prolungato tempo trascorso in rete. Il «tempo di trasmissione» nell'SDK è definito come uno dei seguenti:

- Tempo impiegato nel metodo di un client SDK `HTTPClient.Do()`
- Tempo trascorso in `Read()` s su un corpo di risposta HTTP che è stato inoltrato al chiamante (ad es.) `GetObject`

In caso di problemi dovuti alla latenza o ai timeout delle operazioni, la prima cosa da fare è ottenere la telemetria del ciclo di vita delle operazioni dell'SDK per determinare la ripartizione temporale tra il tempo trascorso sul cavo e il sovraccarico circostante dell'operazione. Consulta la guida sulla temporizzazione delle operazioni dell'SDK, che contiene un frammento di codice riutilizzabile per raggiungere questo obiettivo.

Come posso correggere un errore? `read: connection reset`

Per impostazione predefinita, l'SDK riprova qualsiasi errore corrispondente al `connection reset` pattern. Ciò riguarderà la gestione degli errori per la maggior parte delle operazioni, in cui la risposta HTTP dell'operazione viene completamente utilizzata e deserializzata nel tipo di risultato modellato.

Tuttavia, questo errore può ancora verificarsi in un contesto al di fuori del ciclo di ripetizione: alcune operazioni di servizio inoltrano direttamente il corpo della risposta HTTP dell'API al chiamante per essere utilizzato direttamente dal cavo `io.ReadCloser` (ad esempio`GetObject`, il payload dell'oggetto). È possibile che si verifichi questo errore quando si esegue un comando `Read` sul corpo della risposta.

Questo errore indica che l'host, il servizio o qualsiasi intermediario (ad esempio gateway NAT, proxy, sistemi di bilanciamento del carico) ha chiuso la connessione durante il tentativo di leggere la risposta.

Questo può verificarsi per diversi motivi:

- Il corpo della risposta non è stato utilizzato per qualche tempo dopo la ricezione della risposta stessa (dopo la chiamata dell'operazione di servizio). Ti consigliamo di utilizzare il corpo della risposta HTTP il prima possibile per questi tipi di operazioni.

- Non hai chiuso un corpo di risposta ricevuto in precedenza. Ciò può causare il ripristino della connessione su determinate piattaforme. È NECESSARIO chiudere tutte **`io.ReadCloser`** le istanze fornite nella risposta di un'operazione, indipendentemente dal fatto che se ne utilizzi o meno il contenuto.

Oltre a ciò, prova a eseguire una `tcpdump` connessione interessata ai margini della rete (ad esempio dopo tutti i proxy che controlli). Se notate che l' AWS endpoint sembra inviare un RST TCP, dovreste usare la console di AWS supporto per avviare una causa contro il servizio incriminato. Preparati a fornire richieste IDs e orari specifici in cui si è verificato il problema.

Perché ricevo errori di «firma non valida» quando utilizzo un proxy HTTP con l'SDK?

L'algoritmo di firma per AWS i servizi (generalmente sigv4) è legato alle intestazioni della richiesta serializzata, in particolare alla maggior parte delle intestazioni con il prefisso. `X-` I proxy tendono a modificare la richiesta in uscita aggiungendo ulteriori informazioni di inoltro (spesso tramite un'intestazione), il che di fatto infrange la firma calcolata dall'SDK. `X-Forwarded-For`

Se utilizzi un proxy HTTP e riscontri errori di firma, dovresti cercare di acquisire la richiesta così come appare in uscita dal proxy e determinare se è diversa.

Timing delle operazioni SDK

Quando si eseguono il debug di problemi di timeout/latenza nell'SDK, è fondamentale identificare i componenti del ciclo di vita dell'operazione che richiedono più tempo del previsto per l'esecuzione. Come punto di partenza, in genere è necessario esaminare la ripartizione temporale tra la chiamata operativa complessiva e la chiamata HTTP stessa.

Il seguente programma di esempio implementa una sonda di strumentazione di base in termini di `smithy-go` middleware per i client SQS e ne dimostra l'utilizzo. La sonda emette le seguenti informazioni per ogni chiamata operativa:

- AWS ID della richiesta
- ID del servizio
- nome dell'operazione
- tempo di invocazione dell'operazione
- tempo di chiamata http

Ogni messaggio emesso è preceduto da un «ID di invocazione» univoco (per una singola operazione) impostato all'inizio dello stack del gestore.

Il punto di ingresso per la strumentazione è esposto come`WithOperationTiming`, che è parametrizzato per accettare una funzione di gestione dei messaggi che riceverà gli «eventi» della strumentazione come stringhe formattate. `PrintfMSGHandler`è fornito per comodità e consente semplicemente di scaricare i messaggi su stdout.

Il servizio qui utilizzato è intercambiabile: TUTTE le opzioni del client di servizio sono accettate `APIOptions` e vengono visualizzate come configurazione. `HTTPClient` Ad esempio, `WithOperationTiming` potrebbe invece essere dichiarato come:

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

Se lo modifichi, assicurati di cambiare anche la firma della funzione che restituisce.

```
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()
}
```

Un esempio di output di questo programma:

```
[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
```

Test unitario con la AWS SDK per Go v2

Quando utilizzi l'SDK nella tua applicazione, ti consigliamo di simularlo per il test unitario dell'applicazione. La simulazione dell'SDK consente al test di concentrarsi su ciò che si desidera testare, non sui componenti interni dell'SDK.

Per supportare il mocking, utilizza le interfacce Go anziché tipi di client di servizio concreti, impaginatori e camerieri, ad esempio. `s3.Client` Ciò consente all'applicazione di utilizzare modelli come l'iniezione di dipendenza per testare la logica dell'applicazione.

Mocking Client Operations

In questo esempio, `S3GetObjectAPI` è un'interfaccia che definisce l'insieme di operazioni API Amazon S3 richieste dalla `GetObjectFromS3` funzione. `S3GetObjectAPI`è soddisfatto del metodo del client Amazon S3. GetObject

```
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)
}
```

Per testare la `GetObjectFromS3` funzione, usa `mockGetObjectAPI` per soddisfare la definizione dell'`S3GetObjectAPI`interfaccia. Quindi utilizzate il `mockGetObjectAPI` tipo per simulare l'output e le risposte di errore restituite dal client del servizio.

```
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)
            }
        })
    }
}
```

Mocking Impaginatori

Analogamente ai client di servizio, gli impaginatori possono essere presi in giro definendo un'interfaccia Go per l'impaginatore. Tale interfaccia verrebbe utilizzata dal codice dell'applicazione. Ciò consente di utilizzare l'implementazione dell'SDK quando l'applicazione è in esecuzione e un'implementazione simulata per i test.

Nell'esempio seguente, `ListObjectsV2Pager` è un'interfaccia che definisce i comportamenti per Amazon ListObjectsS3 V2Paginator richiesti dalla funzione. `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
}
```

Per eseguire il test`CountObjects`, crea il `mockListObjectsV2Pager` tipo che soddisfa la definizione dell'interfaccia. `ListObjectsV2Pager` Utilizzatelo quindi `mockListObjectsV2Pager` per replicare il comportamento di paginazione delle risposte di output e di errore provenienti dall'impaginatore per le operazioni di servizio.

```
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)
    }
}
```

Esempi di codice SDK for Go V2

Gli esempi di codice in questo argomento mostrano come utilizzare la AWS SDK per Go V2 con. AWS

Nozioni di base: esempi di codice che mostrano come eseguire le operazioni essenziali all'interno di un servizio.

Le azioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le azioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati.

Scenari: esempi di codice che mostrano come eseguire un'attività specifica chiamando più funzioni all'interno dello stesso servizio o combinate con altri Servizi AWS.

Alcuni servizi contengono ulteriori categorie di esempi che mostrano come sfruttare le librerie o le funzioni specifiche del servizio.

Servizi

- Esempi per Gateway API con SDK per Go V2
- Esempi per Aurora con SDK per Go V2
- Esempi per Amazon Bedrock con SDK per Go V2
- Esempi per l'API Runtime per Amazon Bedrock con SDK per Go V2
- CloudFormation esempi di utilizzo di SDK for Go V2
- CloudWatch Esempi di log utilizzando SDK for Go V2
- Esempi per il gestore dell'identità per Amazon Cognito con SDK per Go V2
- Esempi per Amazon DocumentDB con SDK per Go V2
- Esempi per DynamoDB con SDK per Go V2
- Esempi per IAM con SDK per Go V2
- Esempi per Kinesis con SDK per Go V2
- Esempi per Lambda con SDK per Go V2
- Esempi per Amazon MSK con SDK per Go V2
- Esempi per Centrale Partner con SDK per Go V2
- Esempi per Amazon RDS con SDK per Go V2
- Esempi per Amazon Redshift con SDK per Go V2

- Esempi per Amazon S3 con SDK per Go V2
- Esempi per Amazon SNS con SDK per Go V2
- Esempi per Amazon SQS con SDK per Go V2

Esempi per Gateway API con SDK per Go V2

I seguenti esempi di codice mostrano come eseguire azioni e implementare scenari comuni utilizzando AWS SDK per Go V2 con API Gateway.

AWS i contributi della community sono esempi che sono stati creati e gestiti da più team AWS. Per fornire un feedback, utilizza il meccanismo disponibile nei repository collegati.

Ogni esempio include un link al codice sorgente completo, in cui vengono fornite le istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

- AWS contributi della comunità

AWS contributi della comunità

Creare e testare un'applicazione serverless

L'esempio di codice seguente mostra come creare e testare un'applicazione serverless utilizzando Gateway API con Lambda e DynamoDB

SDK per Go V2

> Mostra come creare e testare un'applicazione serverless composta da Gateway API con Lambda e DynamoDB utilizzando l'SDK per Go.
>
> Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, vedi l'esempio completo su GitHub.
>
> Servizi utilizzati in questo esempio
>
> - Gateway API
> - DynamoDB
> - Lambda

Esempi per Aurora con SDK per Go V2

I seguenti esempi di codice mostrano come eseguire azioni e implementare scenari comuni utilizzando AWS SDK per Go V2 con Aurora.

Nozioni di base: esempi di codice che mostrano come eseguire le operazioni essenziali all'interno di un servizio.

Le azioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le azioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati.

Ogni esempio include un link al codice sorgente completo, in cui vengono fornite le istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

- Nozioni di base
- Nozioni di base
- Azioni

Nozioni di base

Hello Aurora

L'esempio di codice seguente mostra come iniziare a utilizzare Aurora.

SDK per Go V2

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
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)
  }
 }
}
```

- Per i dettagli sull'API, consulta Descrivi DBClusters in AWS SDK per Go API Reference.

Nozioni di base

Informazioni di base

L'esempio di codice seguente mostra come:

- Crea un gruppo di parametri del cluster di database Aurora personalizzati e imposta i relativi valori.
- Crea un cluster di database che utilizza il gruppo di parametri.
- Crea un'istanza database che contiene un database.
- Acquisisci uno snapshot del cluster di database, quindi elimina le risorse.

SDK per Go V2

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Esegui uno scenario interattivo al prompt dei comandi.

```
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()
}
```

Definisci le funzioni richiamate dallo scenario per gestire le operazioni Aurora.

```
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
}
```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella documentazione di riferimento dell'API AWS SDK per Go .
  - CreaDBCluster
  - CreaDBClusterParameterGroup
  - Crea DBCluster istantanea
  - CreaDBInstance
  - EliminaDBCluster
  - EliminaDBClusterParameterGroup
  - EliminaDBInstance
  - Descriva DBCluster ParameterGroups
  - DBClusterDescrivi parametri
  - Descrivi le DBCluster istantanee
  - Descriva DBClusters
  - Descrivi DBEngine versioni
  - Descriva DBInstances
  - DescribeOrderableDBInstanceOpzioni
  - ModificaDBClusterParameterGroup

Azioni

`CreateDBCluster`

Il seguente esempio di codice mostra come usare`CreateDBCluster`.

SDK per Go V2

> **Note**
>
> C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
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
 }
}
```

- Per i dettagli sull'API, consulta Create DBCluster in AWS SDK per Go API Reference.

`CreateDBClusterParameterGroup`

Il seguente esempio di codice mostra come utilizzare`CreateDBClusterParameterGroup`.

SDK per Go V2

> **Note**
>
> C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
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
 }
}
```

- Per i dettagli sull'API, consulta Create DBCluster ParameterGroup in AWS SDK per Go API Reference.

`CreateDBClusterSnapshot`

Il seguente esempio di codice mostra come utilizzare`CreateDBClusterSnapshot`.

SDK per Go V2

> **Note**
>
> C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
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
 }
}
```

- Per i dettagli sull'API, consulta Create DBCluster Snapshot in AWS SDK per Go API Reference.

`CreateDBInstance`

Il seguente esempio di codice mostra come utilizzare`CreateDBInstance`.

SDK per Go V2

> **Note**
>
> C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
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
 }
}
```

- Per i dettagli sull'API, consulta Create DBInstance in AWS SDK per Go API Reference.

`DeleteDBCluster`

Il seguente esempio di codice mostra come utilizzare`DeleteDBCluster`.

SDK per Go V2

> **Note**
>
> C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
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
 }
}
```

- Per i dettagli sull'API, consulta Delete DBCluster in AWS SDK per Go API Reference.

`DeleteDBClusterParameterGroup`

Il seguente esempio di codice mostra come utilizzare`DeleteDBClusterParameterGroup`.

SDK per Go V2

> **Note**
>
> C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
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
 }
}
```

- Per i dettagli sull'API, consulta Delete DBCluster ParameterGroup in AWS SDK per Go API Reference.

`DeleteDBInstance`

Il seguente esempio di codice mostra come utilizzare`DeleteDBInstance`.

SDK per Go V2

> **Note**
>
> C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
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
 }
}
```

- Per i dettagli sull'API, consulta Delete DBInstance in AWS SDK per Go API Reference.

`DescribeDBClusterParameterGroups`

Il seguente esempio di codice mostra come utilizzare`DescribeDBClusterParameterGroups`.

SDK per Go V2

> **Note**
>
> C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
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
 }
}
```

- Per i dettagli sull'API, consulta Descrivi DBCluster ParameterGroups in AWS SDK per Go API Reference.

`DescribeDBClusterParameters`

Il seguente esempio di codice mostra come utilizzare`DescribeDBClusterParameters`.

SDK per Go V2

> **Note**
>
> C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
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
}
```

- Per i dettagli sull'API, consulta DBClusterDescrivi i parametri in AWS SDK per Go API Reference.

`DescribeDBClusterSnapshots`

Il seguente esempio di codice mostra come utilizzare`DescribeDBClusterSnapshots`.

SDK per Go V2

> **Note**
>
> C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
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
 }
}
```

- Per i dettagli sull'API, consulta Descrivi le DBCluster istantanee in AWS SDK per Go API Reference.

`DescribeDBClusters`

Il seguente esempio di codice mostra come utilizzare`DescribeDBClusters`.

SDK per Go V2

> **Note**
>
> C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
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
 }
}
```

- Per i dettagli sull'API, consulta Descrivi DBClusters in AWS SDK per Go API Reference.

`DescribeDBEngineVersions`

Il seguente esempio di codice mostra come utilizzare`DescribeDBEngineVersions`.

SDK per Go V2

> **Note**
>
> C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
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
 }
```

```
}
```

- Per i dettagli sull'API, consulta Descrivi DBEngine le versioni in AWS SDK per Go API Reference.

`DescribeDBInstances`

Il seguente esempio di codice mostra come utilizzare`DescribeDBInstances`.

SDK per Go V2

> **Note**
>
> C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
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
  }
}
```

- Per i dettagli sull'API, consulta Descrivi DBInstances in AWS SDK per Go API Reference.

`DescribeOrderableDBInstanceOptions`

Il seguente esempio di codice mostra come utilizzare`DescribeOrderableDBInstanceOptions`.

SDK per Go V2

> **Note**
>
> C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
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
}
```

- Per i dettagli sull'API, consulta DescribeOrderableDBInstanceOpzioni in AWS SDK per Go API Reference.

`ModifyDBClusterParameterGroup`

Il seguente esempio di codice mostra come utilizzare`ModifyDBClusterParameterGroup`.

SDK per Go V2

> **Note**
>
> C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
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
 }
}
```

- Per i dettagli sull'API, consulta Modify DBCluster ParameterGroup in AWS SDK per Go API Reference.

Esempi per Amazon Bedrock con SDK per Go V2

I seguenti esempi di codice mostrano come eseguire azioni e implementare scenari comuni utilizzando la versione AWS SDK per Go 2 con Amazon Bedrock.

Le azioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le azioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati.

Ogni esempio include un link al codice sorgente completo, in cui vengono fornite le istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

- Nozioni di base
- Azioni

Nozioni di base

Hello Amazon Bedrock

Il seguente esempio di codice mostra come iniziare a usare Amazon Bedrock.

SDK per Go V2

> **Note**
>
> C'è altro su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
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)
 }
}
```

- Per i dettagli sull'API, consulta la ListFoundationModelssezione AWS SDK per Go API Reference.

Azioni

`ListFoundationModels`

Il seguente esempio di codice mostra come utilizzare`ListFoundationModels`.

SDK per Go V2

> **Note**
>
> C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Elenca i modelli di fondazione di Bedrock.

```
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
}
```

- Per i dettagli sull'API, consulta la ListFoundationModelssezione AWS SDK per Go API Reference.

Esempi per l'API Runtime per Amazon Bedrock con SDK per Go V2

I seguenti esempi di codice mostrano come eseguire azioni e implementare scenari comuni utilizzando AWS SDK per Go V2 con Amazon Bedrock Runtime.

Scenari: esempi di codice che mostrano come eseguire un'attività specifica chiamando più funzioni all'interno dello stesso servizio o combinate con altri Servizi AWS.

Ogni esempio include un link al codice sorgente completo, in cui vengono fornite le istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

- Nozioni di base
- Scenari
- Amazon Nova
- Generatore di immagini Amazon Titan
- Anthropic Claude

Nozioni di base

Hello Amazon Bedrock

Il seguente esempio di codice mostra come iniziare a usare Amazon Bedrock.

SDK per Go V2

> **Note**
>
> C'è altro su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
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)
}
```

- Per i dettagli sull'API, consulta la InvokeModelsezione AWS SDK per Go API Reference.

Scenari

Invocare più modelli di fondazione in Amazon Bedrock

Il seguente esempio di codice mostra come preparare e inviare un prompt a una varietà di modelli in grandi lingue (LLMs) su Amazon Bedrock

SDK per Go V2

> **Note**
>
> C'è altro su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Invoca più modelli di fondazione in Amazon Bedrock.

```
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 Meta Llama 2 Chat
// 3. Generate text and asynchronously process the response stream with Anthropic
 Claude 2
// 4. Generate an image with the Amazon Titan image generation 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("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.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) 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-v2")
 fmt.Printf("The generated image has been saved to %s\n", imagePath)
}
```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella documentazione di riferimento dell'API AWS SDK per Go .
  - InvokeModel
  - InvokeModelWithResponseStream

Amazon Nova

Converse

L'esempio di codice seguente mostra come inviare un messaggio di testo ad Amazon Nova utilizzando l'API Converse di Bedrock.

SDK per Go V2

> **Note**
>
> C'è dell'altro GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Invia un messaggio di testo ad Amazon Nova utilizzando l'API Converse di Bedrock.

```
func (wrapper ConverseWrapper) ConverseNova(ctx context.Context, prompt string)
 (string, error) {
 var content = types.ContentBlockMemberText{
  Value: prompt,
 }
 var message = types.Message{
  Content: []types.ContentBlock{&content},
  Role:    "user",
 }
 modelId := "amazon.nova-lite-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
}
```

- Per informazioni dettagliate sull'API, consulta Converse nella documentazione di riferimento dell'API AWS SDK per Go .

Generatore di immagini Amazon Titan

InvokeModel

L'esempio di codice seguente mostra come invocare Generatore di immagini Amazon Titan in Amazon Bedrock per generare un'immagine.

SDK per Go V2

> **Note**
>
> C'è dell'altro GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Crea un'immagine con Generatore di immagini Amazon Titan.

```
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-v2:0"

 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

}
```

- Per i dettagli sull'API, consulta la [InvokeModel](#)sezione AWS SDK per Go API Reference.

Anthropic Claude

Converse

L'esempio di codice seguente mostra come inviare un messaggio di testo ad Anthropic Claude utilizzando l'API Converse di Bedrock.

SDK per Go V2

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Invia un messaggio di testo ad Anthropic Claude utilizzando l'API Converse di Bedrock.

```
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

}
```

- Per informazioni dettagliate sull'API, consulta Converse nella documentazione di riferimento dell'API AWS SDK per Go .

InvokeModel

L'esempio di codice seguente mostra come inviare un messaggio di testo a Anthropic Claude, utilizzando l'API Invoke Model.

SDK per Go V2

> **Note**
>
> C'è dell'altro GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Invoca il modello di fondazione Anthropic Claude 2 per generare testo.

```
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
}
```

- Per i dettagli sull'API, consulta la InvokeModelsezione AWS SDK per Go API Reference.

InvokeModelWithResponseStream

L'esempio di codice seguente mostra come inviare un messaggio di testo ai modelli Anthropic Claude utilizzando l'API Invoke Model e stampare il flusso di risposta.

SDK per Go V2

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Utilizza l'API Invoke Model per inviare un messaggio di testo ed elaborare il flusso di risposta in tempo reale.

```
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
}
```

- Per i dettagli sull'API, consulta la InvokeModelWithResponseStreamsezione AWS SDK per Go API Reference.

CloudFormation esempi di utilizzo di SDK for Go V2

I seguenti esempi di codice mostrano come eseguire azioni e implementare scenari comuni utilizzando AWS SDK per Go V2 con. CloudFormation

Le azioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le azioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati.

Ogni esempio include un link al codice sorgente completo, in cui vengono fornite le istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

- Azioni

Azioni

`DescribeStacks`

Il seguente esempio di codice mostra come utilizzare`DescribeStacks`.

SDK per Go V2

> **Note**
>
> C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
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
}
```

- Per i dettagli sull'API, consulta la DescribeStackssezione AWS SDK per Go API Reference.

CloudWatch Esempi di log utilizzando SDK for Go V2

I seguenti esempi di codice mostrano come eseguire azioni e implementare scenari comuni utilizzando AWS SDK per Go V2 con Logs. CloudWatch

Le azioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le azioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati.

Ogni esempio include un link al codice sorgente completo, in cui vengono fornite le istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

- Azioni

Azioni

`StartLiveTail`

Il seguente esempio di codice mostra come utilizzare. `StartLiveTail`

SDK per Go V2

Includere i file richiesti.

```
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"
)
```

Gestisce gli eventi della sessione Live Tail.

```
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)
   }
  }
 }
}
```

Avvia la sessione Live Tail.

```
 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)
```

Arresta la sessione Live Tail dopo un determinato periodo di tempo.

```
// Close the stream (which ends the session) after a timeout
time.Sleep(10 * time.Second)
stream.Close()
log.Println("Event stream closed")
```

- Per i dettagli sull'API, consulta la StartLiveTailsezione AWS SDK per Go API Reference.

Esempi per il gestore dell’identità per Amazon Cognito con SDK per Go V2

I seguenti esempi di codice mostrano come eseguire azioni e implementare scenari comuni utilizzando la versione AWS SDK per Go 2 con Amazon Cognito Identity Provider.

Le azioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le azioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati.

Scenari: esempi di codice che mostrano come eseguire un’attività specifica chiamando più funzioni all’interno dello stesso servizio o combinate con altri Servizi AWS.

Ogni esempio include un link al codice sorgente completo, in cui vengono fornite le istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

- Nozioni di base
- Azioni

- Scenari

Nozioni di base

Ciao Amazon Cognito

L'esempio di codice seguente mostra come iniziare a utilizzare Amazon Cognito.

SDK per Go V2

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
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)
  }
 }
}
```

- Per i dettagli sull'API, consulta la ListUserPoolssezione AWS SDK per Go API Reference.

Azioni

`AdminCreateUser`

Il seguente esempio di codice mostra come utilizzare`AdminCreateUser`.

SDK per Go V2

> **Note**
>
> C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
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
}
```

- Per i dettagli sull'API, consulta la AdminCreateUsersezione AWS SDK per Go API Reference.

`AdminSetUserPassword`

Il seguente esempio di codice mostra come utilizzare`AdminSetUserPassword`.

SDK per Go V2

> **Note**
>
> C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
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
}
```

- Per i dettagli sull'API, consulta la AdminSetUserPasswordsezione AWS SDK per Go API Reference.

`ConfirmForgotPassword`

Il seguente esempio di codice mostra come utilizzare`ConfirmForgotPassword`.

SDK per Go V2

> **Note**
>
> C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
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
}
```

- Per i dettagli sull'API, consulta la ConfirmForgotPasswordsezione AWS SDK per Go API Reference.

`DeleteUser`

Il seguente esempio di codice mostra come utilizzare`DeleteUser`.

SDK per Go V2

> **Note**
>
> C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
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
}
```

- Per i dettagli sull'API, consulta la DeleteUsersezione AWS SDK per Go API Reference.

`ForgotPassword`

Il seguente esempio di codice mostra come utilizzare`ForgotPassword`.

SDK per Go V2

> **Note**
>
> C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
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
}
```

- Per i dettagli sull'API, consulta la ForgotPasswordsezione AWS SDK per Go API Reference.

`InitiateAuth`

Il seguente esempio di codice mostra come utilizzare`InitiateAuth`.

SDK per Go V2

> **Note**
>
> C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
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
}
```

- Per i dettagli sull'API, consulta la InitiateAuthsezione AWS SDK per Go API Reference.

`ListUserPools`

Il seguente esempio di codice mostra come utilizzare`ListUserPools`.

SDK per Go V2

> **Note**
>
> C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
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)
  }
 }
}
```

- Per i dettagli sull'API, consulta la ListUserPoolssezione AWS SDK per Go API Reference.

`SignUp`

Il seguente esempio di codice mostra come utilizzare`SignUp`.

SDK per Go V2

> **Note**
>
> C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
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
}
```

- Per i dettagli sull'API, consulta la SignUpsezione AWS SDK per Go API Reference.

`UpdateUserPool`

Il seguente esempio di codice mostra come utilizzare`UpdateUserPool`.

SDK per Go V2

> **Note**
>
> C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
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
}
```

- Per i dettagli sull'API, consulta la UpdateUserPoolsezione AWS SDK per Go API Reference.

Scenari

Confermare automaticamente gli utenti noti con una funzione Lambda

L'esempio di codice seguente mostra come confermare automaticamente gli utenti noti di Amazon Cognito con una funzione Lambda.

- Configura un pool di utenti per chiamare una funzione Lambda per il trigger `PreSignUp`.
- Registra un utente con Amazon Cognito.
- La funzione Lambda esegue la scansione di una tabella DynamoDB e conferma automaticamente gli utenti noti.
- Accedi come nuovo utente, quindi elimina le risorse.

SDK per Go V2

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Esegui uno scenario interattivo al prompt dei comandi.

```
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))
}
```

Gestisci il trigger `PreSignUp` con una funzione Lambda.

```
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)
}
```

Crea una struttura che esegue operazioni comuni.

```
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))
}
```

Crea una struttura che racchiude le azioni di Amazon Cognito.

```
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
}
```

Crea una struttura che racchiude le azioni di DynamoDB.

```
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
}
```

Crea una struttura che racchiuda le azioni di CloudWatch Logs.

```
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
}
```

Crea una struttura che racchiuda le azioni. 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
}
```

Eliminare le risorse.

```
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!")
 }
}
```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella documentazione di riferimento dell'API AWS SDK per Go .
  - DeleteUser
  - InitiateAuth
  - SignUp
  - UpdateUserPool

Migrare automaticamente gli utenti noti con una funzione Lambda

L'esempio di codice seguente mostra come migrare automaticamente gli utenti noti di Amazon Cognito con una funzione Lambda.

- Configura un pool di utenti per chiamare una funzione Lambda per il trigger `MigrateUser`.
- Accedi ad Amazon Cognito con un nome utente e un indirizzo e-mail non incluso nel pool di utenti.
- La funzione Lambda esegue la scansione di una tabella DynamoDB e migra automaticamente gli utenti noti al pool di utenti.
- Esegui il flusso della password dimenticata per reimpostare la password per l'utente migrato.
- Accedi come nuovo utente, quindi elimina le risorse.

SDK per Go V2

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Esegui uno scenario interattivo al prompt dei comandi.

```
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))
}
```

Gestisci il trigger `MigrateUser` con una funzione Lambda.

```
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)
}
```

Crea una struttura che esegue operazioni comuni.

```
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))
}
```

Crea una struttura che racchiude le azioni di Amazon Cognito.

```
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
}
```

Crea una struttura che racchiude le azioni di DynamoDB.

```
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
}
```

Crea una struttura che racchiuda le azioni di CloudWatch Logs.

```
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
}
```

Crea una struttura che racchiuda le azioni. 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
}
```

Eliminare le risorse.

```
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!")
 }
}
```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella documentazione di riferimento dell'API AWS SDK per Go .
  - ConfirmForgotPassword
  - DeleteUser
  - ForgotPassword
  - InitiateAuth

- SignUp
- UpdateUserPool

Scrivere i dati di attività personalizzate con una funzione Lambda dopo l'autenticazione utente di Amazon Cognito tramite un SDK

L'esempio di codice seguente mostra come scrivere dati di attività personalizzate con una funzione Lambda dopo l'autenticazione utente di Amazon Cognito.

- Usa le funzioni di amministratore per aggiungere un utente a un pool di utenti.
- Configura un pool di utenti per chiamare una funzione Lambda per il trigger `PostAuthentication`.
- Accedere con il nuovo utente ad Amazon Cognito.
- La funzione Lambda scrive informazioni personalizzate nei CloudWatch log e in una tabella DynamoDB.
- Acquisisci e visualizza dati personalizzati dalla tabella DynamoDB, quindi elimina le risorse.

SDK per Go V2

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Esegui uno scenario interattivo al prompt dei comandi.

```
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))
}
```

Gestisci il trigger `PostAuthentication` con una funzione Lambda.

```
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)
}
```

Crea una struttura che esegue operazioni comuni.

```
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))
}
```

Crea una struttura che racchiude le azioni di Amazon Cognito.

```
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
}
```

Crea una struttura che racchiude le azioni di DynamoDB.

```
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
}
```

Crea una struttura che racchiuda le azioni di CloudWatch Logs.

```
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
}
```

Crea una struttura che racchiuda le azioni. 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
}
```

Eliminare le risorse.

```
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!")
 }
}
```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella documentazione di riferimento dell'API AWS SDK per Go .
  - AdminCreateUser
  - AdminSetUserPassword
  - DeleteUser
  - InitiateAuth
  - UpdateUserPool

Esempi per Amazon DocumentDB con SDK per Go V2

I seguenti esempi di codice mostrano come eseguire azioni e implementare scenari comuni utilizzando la versione AWS SDK per Go 2 con Amazon DocumentDB.

Ogni esempio include un link al codice sorgente completo, in cui vengono fornite le istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

- Esempi serverless

Esempi serverless

Invocare una funzione Lambda da un trigger Amazon DocumentDB

L'esempio di codice seguente mostra come implementare una funzione Lambda che riceve un evento attivato dalla ricezione di record da un flusso di modifiche di DocumentDB. La funzione recupera il payload DocumentDB e registra il contenuto del record.

SDK per Go V2

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Utilizzo di un evento Amazon DocumentDB con Lambda tramite 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))
}
```

Esempi per DynamoDB con SDK per Go V2

I seguenti esempi di codice mostrano come eseguire azioni e implementare scenari comuni utilizzando la versione AWS SDK per Go V2 con DynamoDB.

Nozioni di base: esempi di codice che mostrano come eseguire le operazioni essenziali all'interno di un servizio.

Le azioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le azioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati.

Scenari: esempi di codice che mostrano come eseguire un'attività specifica chiamando più funzioni all'interno dello stesso servizio o combinate con altri Servizi AWS.

AWS i contributi della community sono esempi che sono stati creati e gestiti da diversi team. AWS Per fornire un feedback, utilizza il meccanismo disponibile nei repository collegati.

Ogni esempio include un link al codice sorgente completo, in cui vengono fornite le istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

- Nozioni di base
- Azioni
- Scenari

- Esempi serverless
- AWS contributi della comunità

Nozioni di base

Informazioni di base

L'esempio di codice seguente mostra come:

- Crea una tabella in grado di contenere i dati del filmato.
- Inserire, ottenere e aggiornare un singolo filmato nella tabella.
- Scrivere i dati del filmato nella tabella da un file JSON di esempio.
- Eseguire una query sui filmati che sono stati rilasciati in un dato anno.
- Cerca i filmati che sono stati distribuiti in diversi anni.
- Elimina un filmato dalla tabella, quindi elimina la tabella.

SDK per Go V2

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Esegui uno scenario interattivo per creare la tabella ed eseguire azioni su di essa.

```
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))
}
```

Definisce una struttura di filmato utilizzata in questo esempio.

```
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"])
}
```

Crea una struttura e metodi che chiamano azioni DynamoDB.

```
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
}
```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella documentazione di riferimento dell'API AWS SDK per Go .
  - BatchWriteItem
  - CreateTable
  - DeleteItem
  - DeleteTable
  - DescribeTable
  - GetItem
  - PutItem
  - Query
  - Scan

- UpdateItem

Azioni

`BatchExecuteStatement`

Il seguente esempio di codice mostra come usare`BatchExecuteStatement`.

SDK per Go V2

> **Note**
>
> C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Definisce una struttura di ricevitore di funzioni per l'esempio.

```
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
}
```

Utilizzo di batch di istruzioni INSERT per aggiungere elementi.

```
// 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
}
```

Utilizzo di batch di istruzioni SELECT per ottenere elementi.

```
// 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
}
```

Utilizzo di batch di istruzioni UPDATE per aggiornare elementi.

```
// 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
}
```

Utilizzo di batch di istruzioni DELETE per eliminare elementi.

```
// 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
}
```

Definisce una struttura di filmato utilizzata in questo esempio.

```
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"])
}
```

- Per i dettagli sull'API, consulta la BatchExecuteStatementsezione AWS SDK per Go API Reference.

`BatchWriteItem`

Il seguente esempio di codice mostra come utilizzare`BatchWriteItem`.

SDK per Go V2

> **Note**
>
> C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
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
}
```

Definisce una struttura di filmato utilizzata in questo esempio.

```
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"])
}
```

- Per i dettagli sull'API, consulta la BatchWriteItemsezione AWS SDK per Go API Reference.

`CreateTable`

Il seguente esempio di codice mostra come utilizzare`CreateTable`.

SDK per Go V2

> **Note**
>
> C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
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
}
```

- Per i dettagli sull'API, consulta la CreateTablesezione AWS SDK per Go API Reference.

`DeleteItem`

Il seguente esempio di codice mostra come utilizzare`DeleteItem`.

SDK per Go V2

> **Note**
>
> C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
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
```

```
}
```

Definisce una struttura di filmato utilizzata in questo esempio.

```
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"])
}
```

- Per i dettagli sull'API, consulta la DeleteItemsezione AWS SDK per Go API Reference.

`DeleteTable`

Il seguente esempio di codice mostra come utilizzare`DeleteTable`.

SDK per Go V2

> **Note**
>
> C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
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
}
```

- Per i dettagli sull'API, consulta la DeleteTablesezione AWS SDK per Go API Reference.

`DescribeTable`

Il seguente esempio di codice mostra come utilizzare`DescribeTable`.

SDK per Go V2

> **Note**
>
> C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
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
}
```

- Per i dettagli sull'API, consulta la DescribeTablesezione AWS SDK per Go API Reference.

`ExecuteStatement`

Il seguente esempio di codice mostra come utilizzare`ExecuteStatement`.

SDK per Go V2

> **Note**
>
> C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Definisce una struttura di ricevitore di funzioni per l'esempio.

```
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
}
```

Utilizzo di un'istruzione INSERT per aggiungere un elemento.

```
// 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
}
```

Utilizzo di un'istruzione SELECT per ottenere un elemento.

```
// 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
}
```

Utilizzo di un'istruzione SELECT per ottenere un elenco di elementi e proiettare i risultati.

```
// 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
}
```

Utilizzo di un'istruzione UPDATE per aggiornare un elemento.

```
// 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
}
```

Utilizzo di un'istruzione DELETE per eliminare un elemento.

```
// 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
}
```

Definisce una struttura di filmato utilizzata in questo esempio.

```
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"])
}
```

- Per i dettagli sull'API, consulta la ExecuteStatementsezione AWS SDK per Go API Reference.

`GetItem`

Il seguente esempio di codice mostra come utilizzare`GetItem`.

SDK per Go V2

> **Note**
>
> C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
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
}
```

Definisce una struttura di filmato utilizzata in questo esempio.

```
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"])
}
```

- Per i dettagli sull'API, consulta la GetItemsezione AWS SDK per Go API Reference.

`ListTables`

Il seguente esempio di codice mostra come utilizzare`ListTables`.

SDK per Go V2

> **Note**
>
> C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
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
}
```

- Per i dettagli sull'API, consulta la ListTablessezione AWS SDK per Go API Reference.

`PutItem`

Il seguente esempio di codice mostra come utilizzare`PutItem`.

SDK per Go V2

> **Note**
>
> C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
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
}
```

Definisce una struttura di filmato utilizzata in questo esempio.

```
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"])
}
```

- Per i dettagli sull'API, consulta la PutItemsezione AWS SDK per Go API Reference.

`Query`

Il seguente esempio di codice mostra come utilizzare`Query`.

SDK per Go V2

> **Note**
>
> C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
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
}
```

Definisce una struttura di filmato utilizzata in questo esempio.

```
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"])
}
```

- Per informazioni dettagliate sull'API, consulta Query nella documentazione di riferimento dell'API AWS SDK per Go .

`Scan`

Il seguente esempio di codice mostra come usare`Scan`.

SDK per Go V2

> **Note**
>
> C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
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
}
```

Definisce una struttura di filmato utilizzata in questo esempio.

```
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"])
}
```

- Per informazioni dettagliate sull'API, consulta Scan nella documentazione di riferimento dell'API AWS SDK per Go .

`UpdateItem`

Il seguente esempio di codice mostra come usare`UpdateItem`.

SDK per Go V2

> **Note**
>
> C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
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
}
```

Definisce una struttura di filmato utilizzata in questo esempio.

```
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"])
}
```

- Per i dettagli sull'API, consulta la UpdateItemsezione AWS SDK per Go API Reference.

Scenari

Esecuzione di una query su una tabella mediante batch di istruzioni PartiQL

L'esempio di codice seguente mostra come:

- Ricezione di un batch di elementi mediante più istruzioni SELECT.
- Aggiunta di un batch di articoli eseguendo più istruzioni INSERT.
- Aggiornamento di un batch di elementi mediante più istruzioni UPDATE.
- Eliminazione di un batch di elementi mediante più istruzioni DELETE.

SDK per Go V2

> **Note**
>
> C'è di più su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Esecuzione di uno scenario che crea una tabella ed esegue batch di query PartiQL.

```
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))
}
```

Definisce una struttura di filmato utilizzata in questo esempio.

```
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"])
}
```

Crea una struttura e i metodi che eseguono istruzioni PartiQL.

```
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
}
```

- Per i dettagli sull'API, consulta la BatchExecuteStatementsezione AWS SDK per Go API Reference.

Esecuzione di una query mediante PartiQL

L’esempio di codice seguente mostra come:

- Ricezione di un articolo eseguendo un’istruzione SELECT.
- Aggiunta di un elemento eseguendo un’istruzione INSERT.
- Aggiornamento di un elemento eseguendo un’istruzione UPDATE.
- Eliminazione di un elemento eseguendo un’istruzione DELETE.

SDK per Go V2

> **Note**
>
> C'è di più su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Esecuzione di uno scenario che crea una tabella ed esegue query PartiQL.

```
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))
}
```

Definisce una struttura di filmato utilizzata in questo esempio.

```
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"])
}
```

Crea una struttura e i metodi che eseguono istruzioni PartiQL.

```
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
}
```

- Per i dettagli sull'API, consulta la ExecuteStatementsezione AWS SDK per Go API Reference.

Esempi serverless

Invocare una funzione Lambda da un trigger DynamoDB

L'esempio di codice seguente mostra come implementare una funzione Lambda che riceve un evento attivato dalla ricezione di record da un flusso DynamoDB. La funzione recupera il payload DynamoDB e registra il contenuto del record.

SDK per Go V2

> **Note**
>
> C'è di più su GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Utilizzo di un evento DynamoDB con Lambda tramite 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)
}
```

Segnalazione di errori di elementi batch per funzioni Lambda con un trigger DynamoDB

L'esempio di codice seguente mostra come implementare una risposta batch parziale per funzioni Lambda che ricevono eventi da un flusso DynamoDB. La funzione riporta gli errori degli elementi batch nella risposta, segnalando a Lambda di riprovare tali messaggi in un secondo momento.

SDK per Go V2

> **Note**
>
> C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Segnalazione di errori di elementi batch di DynamoDB con Lambda tramite 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 contributi della comunità

Creare e testare un’applicazione serverless

L’esempio di codice seguente mostra come creare e testare un’applicazione serverless utilizzando Gateway API con Lambda e DynamoDB

SDK per Go V2

Mostra come creare e testare un'applicazione serverless composta da Gateway API con Lambda e DynamoDB utilizzando l'SDK per Go.

Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, vedi l'esempio completo su GitHub.

Servizi utilizzati in questo esempio

- Gateway API
- DynamoDB
- Lambda

Esempi per IAM con SDK per Go V2

I seguenti esempi di codice mostrano come eseguire azioni e implementare scenari comuni utilizzando la AWS SDK per Go V2 con IAM.

Nozioni di base: esempi di codice che mostrano come eseguire le operazioni essenziali all'interno di un servizio.

Le azioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le azioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati.

Ogni esempio include un link al codice sorgente completo, in cui vengono fornite le istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

- Nozioni di base
- Nozioni di base
- Azioni

Nozioni di base

Hello IAM

Il seguente esempio di codice mostra come iniziare a utilizzare IAM.

SDK per Go V2

> **Note**
>
> C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
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)
  }
 }
}
```

- Per i dettagli sull'API, consulta la ListPoliciessezione AWS SDK per Go API Reference.

Nozioni di base

Informazioni di base

L'esempio di codice seguente mostra come creare un utente e assumere un ruolo.

> **Warning**
>
> Per evitare rischi per la sicurezza, non utilizzare gli utenti IAM per l'autenticazione quando sviluppi software creato ad hoc o lavori con dati reali. Utilizza invece la federazione con un provider di identità come AWS IAM Identity Center.

- Crea un utente che non disponga di autorizzazioni.
- Crea un ruolo che conceda l'autorizzazione per elencare i bucket Amazon S3 per l'account.
- Aggiungi una policy per consentire all'utente di assumere il ruolo.
- Assumi il ruolo ed elenca i bucket S3 utilizzando le credenziali temporanee, quindi ripulisci le risorse.

SDK per Go V2

> **Note**
>
> C'è di più su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Esegui uno scenario interattivo al prompt dei comandi.

```
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)
}
```

Definisci una struttura che racchiude le azioni dell'account.

```
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
}
```

Definisci una struttura che racchiude le azioni della policy.

```
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
}
```

Definisci una struttura che racchiude le azioni del ruolo.

```
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
}
```

Definisci una struttura che racchiude le azioni dell'utente.

```
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
}
```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella documentazione di riferimento dell'API AWS SDK per Go .
  - AttachRolePolicy
  - CreateAccessKey
  - CreatePolicy
  - CreateRole
  - CreateUser
  - DeleteAccessKey
  - DeletePolicy
  - DeleteRole
  - DeleteUser
  - DeleteUserPolicy

- DetachRolePolicy
- PutUserPolicy

Azioni

`AttachRolePolicy`

Il seguente esempio di codice mostra come usare`AttachRolePolicy`.

SDK per Go V2

> **Note**
>
> C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
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
}
```

- Per i dettagli sull'API, consulta la AttachRolePolicysezione AWS SDK per Go API Reference.

`CreateAccessKey`

Il seguente esempio di codice mostra come utilizzare`CreateAccessKey`.

SDK per Go V2

> **Note**
>
> C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
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
}
```

- Per i dettagli sull'API, consulta la CreateAccessKeysezione AWS SDK per Go API Reference.

`CreatePolicy`

Il seguente esempio di codice mostra come utilizzare`CreatePolicy`.

SDK per Go V2

> **Note**
>
> C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
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
}
```

- Per i dettagli sull'API, consulta la CreatePolicysezione AWS SDK per Go API Reference.

`CreateRole`

Il seguente esempio di codice mostra come utilizzare`CreateRole`.

SDK per Go V2

> **Note**
>
> C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
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
}
```

- Per i dettagli sull'API, consulta la CreateRolesezione AWS SDK per Go API Reference.

`CreateServiceLinkedRole`

Il seguente esempio di codice mostra come utilizzare`CreateServiceLinkedRole`.

SDK per Go V2

> **Note**
>
> C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
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
}
```

- Per i dettagli sull'API, consulta la CreateServiceLinkedRolesezione AWS SDK per Go API Reference.

`CreateUser`

Il seguente esempio di codice mostra come utilizzare`CreateUser`.

SDK per Go V2

> **Note**
>
> C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
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
}
```

- Per i dettagli sull'API, consulta la CreateUsersezione AWS SDK per Go API Reference.

`DeleteAccessKey`

Il seguente esempio di codice mostra come utilizzare`DeleteAccessKey`.

SDK per Go V2

> **Note**
>
> C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
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
}
```

- Per i dettagli sull'API, consulta la DeleteAccessKeysezione AWS SDK per Go API Reference.

`DeletePolicy`

Il seguente esempio di codice mostra come utilizzare`DeletePolicy`.

SDK per Go V2

> **Note**
>
> C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
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
}
```

- Per i dettagli sull'API, consulta la DeletePolicysezione AWS SDK per Go API Reference.

`DeleteRole`

Il seguente esempio di codice mostra come utilizzare`DeleteRole`.

SDK per Go V2

> **Note**
>
> C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
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
}
```

- Per i dettagli sull'API, consulta la DeleteRolesezione AWS SDK per Go API Reference.

`DeleteServiceLinkedRole`

Il seguente esempio di codice mostra come utilizzare`DeleteServiceLinkedRole`.

SDK per Go V2

> **Note**
>
> C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
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
}
```

- Per i dettagli sull'API, consulta la DeleteServiceLinkedRolesezione AWS SDK per Go API Reference.

`DeleteUser`

Il seguente esempio di codice mostra come utilizzare`DeleteUser`.

SDK per Go V2

> **Note**
>
> C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
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
}
```

- Per i dettagli sull'API, consulta la DeleteUsersezione AWS SDK per Go API Reference.

`DeleteUserPolicy`

Il seguente esempio di codice mostra come utilizzare`DeleteUserPolicy`.

SDK per Go V2

> **Note**
>
> C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
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
}
```

- Per i dettagli sull'API, consulta la DeleteUserPolicysezione AWS SDK per Go API Reference.

`DetachRolePolicy`

Il seguente esempio di codice mostra come utilizzare`DetachRolePolicy`.

SDK per Go V2

> **Note**
>
> C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
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
}
```

- Per i dettagli sull'API, consulta la DetachRolePolicysezione AWS SDK per Go API Reference.

`GetAccountPasswordPolicy`

Il seguente esempio di codice mostra come utilizzare`GetAccountPasswordPolicy`.

SDK per Go V2

> **Note**
>
> C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
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
}
```

- Per i dettagli sull'API, consulta la GetAccountPasswordPolicysezione AWS SDK per Go API Reference.

`GetPolicy`

Il seguente esempio di codice mostra come utilizzare`GetPolicy`.

SDK per Go V2

> **Note**
>
> C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
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
}
```

- Per i dettagli sull'API, consulta la GetPolicysezione AWS SDK per Go API Reference.

`GetRole`

Il seguente esempio di codice mostra come utilizzare`GetRole`.

SDK per Go V2

> **Note**
>
> C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
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
}
```

- Per i dettagli sull'API, consulta la GetRolesezione AWS SDK per Go API Reference.

`GetUser`

Il seguente esempio di codice mostra come utilizzare`GetUser`.

SDK per Go V2

> **Note**
>
> C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
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
}
```

- Per i dettagli sull'API, consulta la GetUsersezione AWS SDK per Go API Reference.

`ListAccessKeys`

Il seguente esempio di codice mostra come utilizzare`ListAccessKeys`.

SDK per Go V2

> **Note**
>
> C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
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
}
```

- Per i dettagli sull'API, consulta la ListAccessKeyssezione AWS SDK per Go API Reference.

`ListAttachedRolePolicies`

Il seguente esempio di codice mostra come utilizzare`ListAttachedRolePolicies`.

SDK per Go V2

> **Note**
>
> C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
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
}
```

- Per i dettagli sull'API, consulta la ListAttachedRolePoliciessezione AWS SDK per Go API Reference.

`ListGroups`

Il seguente esempio di codice mostra come utilizzare`ListGroups`.

SDK per Go V2

> **Note**
>
> C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
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
}
```

- Per i dettagli sull'API, consulta la ListGroupssezione AWS SDK per Go API Reference.

`ListPolicies`

Il seguente esempio di codice mostra come utilizzare`ListPolicies`.

SDK per Go V2

> **Note**
>
> C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
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
}
```

- Per i dettagli sull'API, consulta la ListPoliciessezione AWS SDK per Go API Reference.

`ListRolePolicies`

Il seguente esempio di codice mostra come utilizzare`ListRolePolicies`.

SDK per Go V2

> **Note**
>
> C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
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
}
```

- Per i dettagli sull'API, consulta la ListRolePoliciessezione AWS SDK per Go API Reference.

`ListRoles`

Il seguente esempio di codice mostra come utilizzare`ListRoles`.

SDK per Go V2

> **Note**
>
> C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
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
}
```

- Per i dettagli sull'API, consulta la ListRolessezione AWS SDK per Go API Reference.

`ListSAMLProviders`

Il seguente esempio di codice mostra come utilizzare`ListSAMLProviders`.

SDK per Go V2

> **Note**
>
> C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
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
}
```

- Per i dettagli sull'API, consulta List SAMLProviders in AWS SDK per Go API Reference.

`ListUserPolicies`

Il seguente esempio di codice mostra come utilizzare`ListUserPolicies`.

SDK per Go V2

> **Note**
>
> C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
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
```

```
}
```

- Per i dettagli sull'API, consulta la ListUserPoliciessezione AWS SDK per Go API Reference.

`ListUsers`

Il seguente esempio di codice mostra come utilizzare`ListUsers`.

SDK per Go V2

> **Note**
>
> C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
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
}
```

- Per i dettagli sull'API, consulta la ListUserssezione AWS SDK per Go API Reference.

`PutUserPolicy`

Il seguente esempio di codice mostra come utilizzare`PutUserPolicy`.

SDK per Go V2

> **Note**
>
> C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
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
}
```

- Per i dettagli sull'API, consulta la PutUserPolicysezione AWS SDK per Go API Reference.

Esempi per Kinesis con SDK per Go V2

I seguenti esempi di codice mostrano come eseguire azioni e implementare scenari comuni utilizzando la versione AWS SDK per Go V2 con Kinesis.

Ogni esempio include un link al codice sorgente completo, in cui vengono fornite le istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

- Esempi serverless

Esempi serverless

Invocare una funzione Lambda da un trigger Kinesis

L'esempio di codice seguente mostra come implementare una funzione Lambda che riceve un evento attivato dalla ricezione di record da un flusso Kinesis. La funzione recupera il payload Kinesis, lo decodifica da Base64 e registra il contenuto del record.

SDK per Go V2

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Utilizzo di un evento Kinesis con Lambda tramite 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)
}
```

Segnalazione di errori di elementi batch per funzioni Lambda con un trigger Kinesis

L'esempio di codice seguente mostra come implementare una risposta batch parziale per funzioni Lambda che ricevono eventi da un flusso Kinesis. La funzione riporta gli errori degli elementi batch nella risposta, segnalando a Lambda di riprovare tali messaggi in un secondo momento.

SDK per Go V2

> **Note**
>
> C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Segnalazione di errori di elementi batch di Kinesis con Lambda tramite 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)
}
```

Esempi per Lambda con SDK per Go V2

I seguenti esempi di codice mostrano come eseguire azioni e implementare scenari comuni utilizzando AWS SDK per Go V2 con Lambda.

Nozioni di base: esempi di codice che mostrano come eseguire le operazioni essenziali all'interno di un servizio.

Le azioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le azioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati.

Scenari: esempi di codice che mostrano come eseguire un'attività specifica chiamando più funzioni all'interno dello stesso servizio o combinate con altri Servizi AWS.

AWS i contributi della community sono esempi che sono stati creati e gestiti da più team. AWS Per fornire un feedback, utilizza il meccanismo disponibile nei repository collegati.

Ogni esempio include un link al codice sorgente completo, in cui vengono fornite le istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

- Nozioni di base
- Nozioni di base
- Azioni
- Scenari
- Esempi serverless
- AWS contributi della comunità

Nozioni di base

Hello Lambda

Il seguente esempio di codice mostra come iniziare a usare Lambda.

SDK per Go V2

> **Note**
>
> C'è altro su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
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)
  }
 }
}
```

- Per i dettagli sull'API, consulta la ListFunctionssezione AWS SDK per Go API Reference.

Nozioni di base

Informazioni di base

L'esempio di codice seguente mostra come:

- Crea un ruolo IAM e una funzione Lambda, quindi carica il codice del gestore.
- Invocare la funzione con un singolo parametro e ottenere i risultati.
- Aggiorna il codice della funzione e configuralo con una variabile di ambiente.
- Invocare la funzione con nuovi parametri e ottenere i risultati. Visualizza il log di esecuzione restituito.
- Elenca le funzioni dell'account, quindi elimina le risorse.

Per ulteriori informazioni, consulta Creare una funzione Lambda con la console.

SDK per Go V2

> **Note**
>
> C'è di più su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Creare uno scenario interattivo che ti mostri come iniziare a usare le funzioni Lambda.

```
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
}
```

Creare una struttura che racchiude le singole operazioni Lambda.

```
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"`
}
```

Definire un gestore Lambda che incrementa un numero.

```
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
```

Definire un secondo gestore Lambda che esegue operazioni aritmetiche.

```
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
```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella documentazione di riferimento dell'API AWS SDK per Go .
  - CreateFunction
  - DeleteFunction
  - GetFunction
  - Invoke
  - ListFunctions
  - UpdateFunctionCode
  - UpdateFunctionConfiguration

Azioni

`CreateFunction`

Il seguente esempio di codice mostra come usare`CreateFunction`.

SDK per Go V2

> **Note**
>
> C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
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
}
```

- Per i dettagli sull'API, consulta la CreateFunctionsezione AWS SDK per Go API Reference.

`DeleteFunction`

Il seguente esempio di codice mostra come utilizzare`DeleteFunction`.

SDK per Go V2

> **Note**
>
> C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
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)
 }
}
```

- Per i dettagli sull'API, consulta la DeleteFunctionsezione AWS SDK per Go API Reference.

`GetFunction`

Il seguente esempio di codice mostra come utilizzare`GetFunction`.

SDK per Go V2

> **Note**
>
> C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
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
}
```

- Per i dettagli sull'API, consulta la GetFunctionsezione AWS SDK per Go API Reference.

`Invoke`

Il seguente esempio di codice mostra come utilizzare`Invoke`.

SDK per Go V2

> **Note**
>
> C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
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
}
```

- Per informazioni dettagliate sull'API, consulta Invoke nella documentazione di riferimento dell'API AWS SDK per Go .

`ListFunctions`

Il seguente esempio di codice mostra come usare`ListFunctions`.

SDK per Go V2

> **Note**
>
> C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
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
}
```

- Per i dettagli sull'API, consulta la ListFunctionssezione AWS SDK per Go API Reference.

`UpdateFunctionCode`

Il seguente esempio di codice mostra come utilizzare`UpdateFunctionCode`.

SDK per Go V2

> **Note**
>
> C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
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
}
```

- Per i dettagli sull'API, consulta la UpdateFunctionCodesezione AWS SDK per Go API Reference.

`UpdateFunctionConfiguration`

Il seguente esempio di codice mostra come utilizzare`UpdateFunctionConfiguration`.

SDK per Go V2

> **Note**
>
> C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
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)
```

```
 }
}
```

- Per i dettagli sull'API, consulta la UpdateFunctionConfigurationsezione AWS SDK per Go API Reference.

Scenari

Confermare automaticamente gli utenti noti con una funzione Lambda

L'esempio di codice seguente mostra come confermare automaticamente gli utenti noti di Amazon Cognito con una funzione Lambda.

- Configura un pool di utenti per chiamare una funzione Lambda per il trigger `PreSignUp`.
- Registra un utente con Amazon Cognito.
- La funzione Lambda esegue la scansione di una tabella DynamoDB e conferma automaticamente gli utenti noti.
- Accedi come nuovo utente, quindi elimina le risorse.

SDK per Go V2

> **Note**
>
> C'è di più su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Esegui uno scenario interattivo al prompt dei comandi.

```
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))
}
```

Gestisci il trigger `PreSignUp` con una funzione Lambda.

```
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)
}
```

Crea una struttura che esegue operazioni comuni.

```
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))
}
```

Crea una struttura che racchiude le azioni di Amazon Cognito.

```
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
}
```

Crea una struttura che racchiude le azioni di DynamoDB.

```
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
}
```

Crea una struttura che racchiuda le azioni di CloudWatch Logs.

```
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
```

```
}
```

Crea una struttura che racchiuda le azioni. 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
}
```

Eliminare le risorse.

```
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!")
 }
}
```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella documentazione di riferimento dell'API AWS SDK per Go .
  - DeleteUser
  - InitiateAuth
  - SignUp
  - UpdateUserPool

Migrare automaticamente gli utenti noti con una funzione Lambda

L'esempio di codice seguente mostra come migrare automaticamente gli utenti noti di Amazon Cognito con una funzione Lambda.

- Configura un pool di utenti per chiamare una funzione Lambda per il trigger `MigrateUser`.
- Accedi ad Amazon Cognito con un nome utente e un indirizzo e-mail non incluso nel pool di utenti.
- La funzione Lambda esegue la scansione di una tabella DynamoDB e migra automaticamente gli utenti noti al pool di utenti.
- Esegui il flusso della password dimenticata per reimpostare la password per l'utente migrato.
- Accedi come nuovo utente, quindi elimina le risorse.

SDK per Go V2

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Esegui uno scenario interattivo al prompt dei comandi.

```
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))
}
```

Gestisci il trigger `MigrateUser` con una funzione Lambda.

```
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)
}
```

Crea una struttura che esegue operazioni comuni.

```
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))
}
```

Crea una struttura che racchiude le azioni di Amazon Cognito.

```
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
}
```

Crea una struttura che racchiude le azioni di DynamoDB.

```
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
}
```

Crea una struttura che racchiuda le azioni di CloudWatch Logs.

```
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
}
```

Crea una struttura che racchiuda le azioni. 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
}
```

Eliminare le risorse.

```
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!")
 }
}
```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella documentazione di riferimento dell'API AWS SDK per Go .
  - ConfirmForgotPassword
  - DeleteUser
  - ForgotPassword
  - InitiateAuth
  - SignUp
  - UpdateUserPool

Scrivere i dati di attività personalizzate con una funzione Lambda dopo l'autenticazione utente di Amazon Cognito tramite un SDK

L'esempio di codice seguente mostra come scrivere dati di attività personalizzate con una funzione Lambda dopo l'autenticazione utente di Amazon Cognito.

- Usa le funzioni di amministratore per aggiungere un utente a un pool di utenti.
- Configura un pool di utenti per chiamare una funzione Lambda per il trigger `PostAuthentication`.
- Accedere con il nuovo utente ad Amazon Cognito.

- La funzione Lambda scrive informazioni personalizzate nei CloudWatch log e in una tabella DynamoDB.
- Acquisisci e visualizza dati personalizzati dalla tabella DynamoDB, quindi elimina le risorse.

SDK per Go V2

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Esegui uno scenario interattivo al prompt dei comandi.

```
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))
}
```

Gestisci il trigger `PostAuthentication` con una funzione Lambda.

```
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)
}
```

Crea una struttura che esegue operazioni comuni.

```
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))
}
```

Crea una struttura che racchiude le azioni di Amazon Cognito.

```
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
}
```

Crea una struttura che racchiude le azioni di DynamoDB.

```
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
}
```

Crea una struttura che racchiuda le azioni di CloudWatch Logs.

```
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
}
```

Crea una struttura che racchiuda le azioni. 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
}
```

Eliminare le risorse.

```
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!")
 }
}
```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella documentazione di riferimento dell'API AWS SDK per Go .
  - AdminCreateUser
  - AdminSetUserPassword
  - DeleteUser
  - InitiateAuth

- UpdateUserPool

Esempi serverless

Connessione a un database Amazon RDS in una funzione Lambda

L'esempio di codice seguente mostra come implementare una funzione Lambda che si connette a un database RDS. La funzione effettua una semplice richiesta al database e restituisce il risultato.

SDK per Go V2

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Connessione a un database Amazon RDS in una funzione Lambda tramite 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)
}
```

Invocare una funzione Lambda da un trigger Kinesis

L'esempio di codice seguente mostra come implementare una funzione Lambda che riceve un evento attivato dalla ricezione di record da un flusso Kinesis. La funzione recupera il payload Kinesis, lo decodifica da Base64 e registra il contenuto del record.

SDK per Go V2

> **Note**
>
> C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Utilizzo di un evento Kinesis con Lambda tramite 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)
}
```

Invocare una funzione Lambda da un trigger DynamoDB

L'esempio di codice seguente mostra come implementare una funzione Lambda che riceve un evento attivato dalla ricezione di record da un flusso DynamoDB. La funzione recupera il payload DynamoDB e registra il contenuto del record.

SDK per Go V2

> **Note**
>
> C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Utilizzo di un evento DynamoDB con Lambda tramite 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)
}
```

Invocare una funzione Lambda da un trigger Amazon DocumentDB

L'esempio di codice seguente mostra come implementare una funzione Lambda che riceve un evento attivato dalla ricezione di record da un flusso di modifiche di DocumentDB. La funzione recupera il payload DocumentDB e registra il contenuto del record.

SDK per Go V2

> **Note**
>
> C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Utilizzo di un evento Amazon DocumentDB con Lambda tramite 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))
}
```

Invocare una funzione Lambda da un trigger Amazon MSK

L'esempio di codice seguente mostra come implementare una funzione Lambda che riceve un evento attivato dalla ricezione di record da un cluster Amazon MSK. La funzione recupera il payload MSK e registra il contenuto del record.

SDK per Go V2

> **Note**
>
> C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Utilizzo di un evento Amazon MSK con Lambda tramite 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)
}
```

Invocazione di una funzione Lambda da un trigger Amazon S3

L'esempio di codice seguente mostra come implementare una funzione Lambda che riceve un evento attivato dal caricamento di un oggetto in un bucket S3. La funzione recupera il nome del bucket S3 e la chiave dell'oggetto dal parametro evento e chiama l'API Amazon S3 per recuperare e registrare il tipo di contenuto dell'oggetto.

SDK per Go V2

> **Note**
>
> C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Utilizzo di un evento S3 con Lambda tramite 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)
}
```

Invocazione di una funzione Lambda da un trigger Amazon SNS

L'esempio di codice seguente mostra come implementare una funzione Lambda che riceve un evento attivato dal ricevimento di messaggi da un argomento SNS. La funzione recupera i messaggi dal parametro dell'evento e registra il contenuto di ogni messaggio.

SDK per Go V2

> **Note**
>
> C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Utilizzo di un evento SNS con Lambda tramite 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)
}
```

Invocazione di una funzione Lambda da un trigger Amazon SQS

L'esempio di codice seguente mostra come implementare una funzione Lambda che riceve un evento attivato dalla ricezione di messaggi da una coda SQS. La funzione recupera i messaggi dal parametro dell'evento e registra il contenuto di ogni messaggio.

SDK per Go V2

> **Note**
>
> C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Utilizzo di un evento SQS con Lambda tramite 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)
}
```

Segnalazione di errori di elementi batch per funzioni Lambda con un trigger Kinesis

L'esempio di codice seguente mostra come implementare una risposta batch parziale per funzioni Lambda che ricevono eventi da un flusso Kinesis. La funzione riporta gli errori degli elementi batch nella risposta, segnalando a Lambda di riprovare tali messaggi in un secondo momento.

SDK per Go V2

> **Note**
>
> C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Segnalazione di errori di elementi batch di Kinesis con Lambda tramite 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)
}
```

Segnalazione di errori di elementi batch per funzioni Lambda con un trigger DynamoDB

L'esempio di codice seguente mostra come implementare una risposta batch parziale per funzioni Lambda che ricevono eventi da un flusso DynamoDB. La funzione riporta gli errori degli elementi batch nella risposta, segnalando a Lambda di riprovare tali messaggi in un secondo momento.

SDK per Go V2

> **Note**
>
> C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Segnalazione di errori di elementi batch di DynamoDB con Lambda tramite 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)
```

```
}
```

Segnalazione di errori di elementi batch per funzioni Lambda con un trigger Amazon SQS

L'esempio di codice seguente mostra come implementare una risposta batch parziale per funzioni Lambda che ricevono eventi da una coda SQS. La funzione riporta gli errori degli elementi batch nella risposta, segnalando a Lambda di riprovare tali messaggi in un secondo momento.

SDK per Go V2

> **Note**
>
> C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Segnalazione di errori di elementi batch di SQS con Lambda tramite 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, sqsEvent events.SQSEvent) (map[string]interface{},
 error) {
 batchItemFailures := []map[string]interface{}{}

 for _, message := range sqsEvent.Records {
  if len(message.Body) > 0 {
   // Your message processing condition here
   fmt.Printf("Successfully processed message: %s\n", message.Body)
  } else {
   // Message processing failed
   fmt.Printf("Failed to process message %s\n", message.MessageId)
```

```
   batchItemFailures = append(batchItemFailures, map[string]interface{}
{"itemIdentifier": message.MessageId})
  }
 }

 sqsBatchResponse := map[string]interface{}{
  "batchItemFailures": batchItemFailures,
 }
 return sqsBatchResponse, nil
}

func main() {
 lambda.Start(handler)
}
```

AWS contributi della comunità

Creare e testare un'applicazione serverless

L'esempio di codice seguente mostra come creare e testare un'applicazione serverless utilizzando Gateway API con Lambda e DynamoDB

SDK per Go V2

Mostra come creare e testare un'applicazione serverless composta da Gateway API con Lambda e DynamoDB utilizzando l'SDK per Go.

Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, vedi l'esempio completo su GitHub.

Servizi utilizzati in questo esempio

- Gateway API
- DynamoDB
- Lambda

Esempi per Amazon MSK con SDK per Go V2

I seguenti esempi di codice mostrano come eseguire azioni e implementare scenari comuni utilizzando la AWS SDK per Go versione 2 con Amazon MSK.

Ogni esempio include un link al codice sorgente completo, in cui vengono fornite le istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

- Esempi serverless

Esempi serverless

Invocare una funzione Lambda da un trigger Amazon MSK

L'esempio di codice seguente mostra come implementare una funzione Lambda che riceve un evento attivato dalla ricezione di record da un cluster Amazon MSK. La funzione recupera il payload MSK e registra il contenuto del record.

SDK per Go V2

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Utilizzo di un evento Amazon MSK con Lambda tramite 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)
}
```

Esempi per Centrale Partner con SDK per Go V2

I seguenti esempi di codice mostrano come eseguire azioni e implementare scenari comuni utilizzando AWS SDK per Go V2 con Partner Central.

Le azioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le azioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati.

Ogni esempio include un link al codice sorgente completo, in cui vengono fornite le istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

- Azioni

Azioni

`GetOpportunity`

Il seguente esempio di codice mostra come utilizzare`GetOpportunity`.

SDK per Go V2

Ottiene un'opportunità.

```
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))
}
```

- Per i dettagli sull'API, consulta la GetOpportunitysezione AWS SDK per Go API Reference.

`ListOpportunities`

Il seguente esempio di codice mostra come utilizzare`ListOpportunities`.

SDK per Go V2

Elenca le opportunità.

```
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))
}
```

- Per i dettagli sull'API, consulta la ListOpportunitiessezione AWS SDK per Go API Reference.

Esempi per Amazon RDS con SDK per Go V2

I seguenti esempi di codice mostrano come eseguire azioni e implementare scenari comuni utilizzando la AWS SDK per Go versione 2 con Amazon RDS.

Nozioni di base: esempi di codice che mostrano come eseguire le operazioni essenziali all'interno di un servizio.

Le azioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le azioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati.

Ogni esempio include un link al codice sorgente completo, in cui vengono fornite le istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

- Nozioni di base
- Nozioni di base
- Azioni
- Esempi serverless

Nozioni di base

Hello Amazon RDS

Il seguente esempio di codice mostra come iniziare a usare Amazon RDS.

SDK per Go V2

> **Note**
>
> C'è altro su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
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)
  }
 }
}
```

- Per i dettagli sull'API, consulta Descrivi DBInstances in AWS SDK per Go API Reference.

Nozioni di base

Informazioni di base

L'esempio di codice seguente mostra come:

- Creare un gruppo di parametri database personalizzati e imposta i relativi valori.
- Creare un'istanza database configurata per utilizzare il gruppo di parametri. L'istanza DB contiene anche un database.
- Acquisire uno snapshot dell'istanza.
- Eliminare l'istanza e il gruppo di parametri.

SDK per Go V2

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Esegui uno scenario interattivo al prompt dei comandi.

```
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()
}
```

Definisci le funzioni richiamate dallo scenario per gestire le operazioni di Amazon RDS.

```
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
}
```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella documentazione di riferimento dell'API AWS SDK per Go .
  - CreaDBInstance
  - Crea DBParameter gruppo
  - CreaDBSnapshot

- EliminaDBInstance
- Elimina DBParameter gruppo
- Descrivi DBEngine versioni
- Descriva DBInstances
- DBParameterDescrivi gruppi
- Descriva DBParameters
- Descriva DBSnapshots
- DescribeOrderableDBInstanceOpzioni
- Modifica DBParameter gruppo

Azioni

`CreateDBInstance`

Il seguente esempio di codice mostra come utilizzare`CreateDBInstance`.

SDK per Go V2

> **Note**
>
> C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
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
 }
}
```

- Per i dettagli sull'API, consulta Create DBInstance in AWS SDK per Go API Reference.

`CreateDBParameterGroup`

Il seguente esempio di codice mostra come utilizzare`CreateDBParameterGroup`.

SDK per Go V2

> **Note**
>
> C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
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
 }
}
```

- Per i dettagli sull'API, consulta Create DBParameter Group in AWS SDK per Go API Reference.

`CreateDBSnapshot`

Il seguente esempio di codice mostra come utilizzare`CreateDBSnapshot`.

SDK per Go V2

> **Note**
>
> C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
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
 }
}
```

- Per i dettagli sull'API, consulta Create DBSnapshot in AWS SDK per Go API Reference.

`DeleteDBInstance`

Il seguente esempio di codice mostra come utilizzare`DeleteDBInstance`.

SDK per Go V2

> **Note**
>
> C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
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
 }
}
```

- Per i dettagli sull'API, consulta Delete DBInstance in AWS SDK per Go API Reference.

`DeleteDBParameterGroup`

Il seguente esempio di codice mostra come utilizzare`DeleteDBParameterGroup`.

SDK per Go V2

> **Note**
>
> C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
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
 }
}
```

- Per i dettagli sull'API, consulta Delete DBParameter Group in AWS SDK per Go API Reference.

`DescribeDBEngineVersions`

Il seguente esempio di codice mostra come utilizzare`DescribeDBEngineVersions`.

SDK per Go V2

> **Note**
>
> C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
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
 }
}
```

- Per i dettagli sull'API, consulta Descrivi DBEngine le versioni in AWS SDK per Go API Reference.

`DescribeDBInstances`

Il seguente esempio di codice mostra come utilizzare`DescribeDBInstances`.

SDK per Go V2

> **Note**
>
> C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
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
 }
}
```

- Per i dettagli sull'API, consulta Descrivi DBInstances in AWS SDK per Go API Reference.

`DescribeDBParameterGroups`

Il seguente esempio di codice mostra come utilizzare`DescribeDBParameterGroups`.

SDK per Go V2

> **Note**
>
> C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
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
 }
}
```

- Per i dettagli sull'API, consulta Descrivere DBParameter i gruppi in AWS SDK per Go API Reference.

`DescribeDBParameters`

Il seguente esempio di codice mostra come utilizzare`DescribeDBParameters`.

SDK per Go V2

> **Note**
>
> C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
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
}
```

- Per i dettagli sull'API, consulta Descrivi DBParameters in AWS SDK per Go API Reference.

`DescribeDBSnapshots`

Il seguente esempio di codice mostra come utilizzare`DescribeDBSnapshots`.

SDK per Go V2

> **Note**
>
> C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
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
 }
}
```

- Per i dettagli sull'API, consulta Descrivi DBSnapshots in AWS SDK per Go API Reference.

`DescribeOrderableDBInstanceOptions`

Il seguente esempio di codice mostra come utilizzare`DescribeOrderableDBInstanceOptions`.

SDK per Go V2

> **Note**
>
> C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
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
}
```

- Per i dettagli sull'API, consulta DescribeOrderableDBInstanceOpzioni in AWS SDK per Go API Reference.

`ModifyDBParameterGroup`

Il seguente esempio di codice mostra come utilizzare`ModifyDBParameterGroup`.

SDK per Go V2

> **Note**
>
> C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
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
 }
}
```

- Per i dettagli sull'API, consulta Modify DBParameter Group in AWS SDK per Go API Reference.

Esempi serverless

Connessione a un database Amazon RDS in una funzione Lambda

L'esempio di codice seguente mostra come implementare una funzione Lambda che si connette a un database RDS. La funzione effettua una semplice richiesta al database e restituisce il risultato.

SDK per Go V2

> **Note**
>
> C'è di più su GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Connessione a un database Amazon RDS in una funzione Lambda tramite 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)
```

```
}
```

Esempi per Amazon Redshift con SDK per Go V2

I seguenti esempi di codice mostrano come eseguire azioni e implementare scenari comuni utilizzando la AWS SDK per Go versione 2 con Amazon Redshift.

Nozioni di base: esempi di codice che mostrano come eseguire le operazioni essenziali all'interno di un servizio.

Le azioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le azioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati.

Ogni esempio include un link al codice sorgente completo, in cui vengono fornite le istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

- Nozioni di base
- Nozioni di base
- Azioni

Nozioni di base

Hello Amazon Redshift

Il seguente esempio di codice mostra come iniziare a usare Amazon Redshift.

SDK per Go V2

> **Note**
>
> C'è altro su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
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)
 }
}
```

- Per i dettagli sull'API, consulta la DescribeClusterssezione AWS SDK per Go API Reference.

Nozioni di base

Informazioni di base

L'esempio di codice seguente mostra come:

- Creare un cluster Redshift.
- Elencare i database nel cluster.
- Creare una tabella denominata Movies.
- Popolare la tabella Movies.
- Eseguire query nella tabella Movies per anno.
- Modificare il cluster Redshift.
- Eliminare il cluster Amazon Redshift.

SDK per Go V2

> **Note**
>
> C'è di più su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
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)
   }
  }
 }
}
```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella documentazione di riferimento dell'API AWS SDK per Go .
  - CreateCluster
  - DescribeClusters
  - DescribeStatement
  - ExecuteStatement

- GetStatementResult
- ListDatabasesPaginator
- ModifyCluster

Azioni

`CreateCluster`

Il seguente esempio di codice mostra come usare`CreateCluster`.

SDK per Go V2

> **Note**
>
> C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
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
}
```

- Per i dettagli sull'API, consulta la CreateClustersezione AWS SDK per Go API Reference.

`DeleteCluster`

Il seguente esempio di codice mostra come utilizzare`DeleteCluster`.

SDK per Go V2

> **Note**
>
> C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
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
}
```

- Per i dettagli sull'API, consulta la DeleteClustersezione AWS SDK per Go API Reference.

`DescribeClusters`

Il seguente esempio di codice mostra come utilizzare`DescribeClusters`.

SDK per Go V2

> **Note**
>
> C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
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
}
```

- Per i dettagli sull'API, consulta la DescribeClusterssezione AWS SDK per Go API Reference.

`ModifyCluster`

Il seguente esempio di codice mostra come utilizzare`ModifyCluster`.

SDK per Go V2

> **Note**
>
> C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
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
}
```

- Per i dettagli sull'API, consulta la ModifyClustersezione AWS SDK per Go API Reference.

Esempi per Amazon S3 con SDK per Go V2

I seguenti esempi di codice mostrano come eseguire azioni e implementare scenari comuni utilizzando la versione AWS SDK per Go V2 con Amazon S3.

Nozioni di base: esempi di codice che mostrano come eseguire le operazioni essenziali all'interno di un servizio.

Le azioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le azioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati.

Scenari: esempi di codice che mostrano come eseguire un'attività specifica chiamando più funzioni all'interno dello stesso servizio o combinate con altri Servizi AWS.

Ogni esempio include un link al codice sorgente completo, in cui vengono fornite le istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

- Nozioni di base
- Nozioni di base
- Azioni
- Scenari
- Esempi serverless

Nozioni di base

Hello Amazon S3

Il seguente esempio di codice mostra come iniziare a usare Amazon S3.

SDK per Go V2

> **Note**
>
> C'è altro su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
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)
   }
  }
}
```

- Per i dettagli sull'API, consulta la ListBucketssezione AWS SDK per Go API Reference.

Nozioni di base

Informazioni di base

L'esempio di codice seguente mostra come:

- Crea un bucket e carica un file in tale bucket.
- Scaricare un oggetto da un bucket.
- Copiare un oggetto in una sottocartella in un bucket.
- Elencare gli oggetti in un bucket.
- Elimina il bucket e tutti gli oggetti in esso contenuti.

SDK per Go V2

> **Note**
>
> C'è di più su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Definisci una struttura che racchiude le azioni del bucket e dell'oggetto utilizzate dallo scenario.

```
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
}
```

Esegue uno scenario interattivo che mostra come utilizzare i bucket e gli oggetti S3.

```
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))
}
```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella documentazione di riferimento dell'API AWS SDK per Go .
  - CopyObject

- CreateBucket
- DeleteBucket
- DeleteObjects
- GetObject
- ListObjectsV2
- PutObject

Azioni

`CopyObject`

Il seguente esempio di codice mostra come usare`CopyObject`.

SDK per Go V2

> **Note**
>
> C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
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
}
```

- Per i dettagli sull'API, consulta la CopyObjectsezione AWS SDK per Go API Reference.

`CreateBucket`

Il seguente esempio di codice mostra come utilizzare`CreateBucket`.

SDK per Go V2

> **Note**
>
> C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Crea un bucket con la configurazione predefinita.

```
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
}
```

Crea un bucket con Object Lock abilitato e attende la verifica della sua esistenza.

```
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
}
```

- Per i dettagli sull'API, consulta la CreateBucketsezione AWS SDK per Go API Reference.

`DeleteBucket`

Il seguente esempio di codice mostra come utilizzare`DeleteBucket`.

SDK per Go V2

> **Note**
>
> C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
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
}
```

- Per i dettagli sull'API, consulta la DeleteBucketsezione AWS SDK per Go API Reference.

`DeleteObject`

Il seguente esempio di codice mostra come utilizzare`DeleteObject`.

SDK per Go V2

> **Note**
>
> C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
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
}
```

- Per i dettagli sull'API, consulta la DeleteObjectsezione AWS SDK per Go API Reference.

`DeleteObjects`

Il seguente esempio di codice mostra come utilizzare`DeleteObjects`.

SDK per Go V2

> **Note**
>
> C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
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
}
```

- Per i dettagli sull'API, consulta la DeleteObjectssezione AWS SDK per Go API Reference.

`GetObject`

Il seguente esempio di codice mostra come utilizzare`GetObject`.

SDK per Go V2

> **Note**
>
> C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
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
}
```

- Per i dettagli sull'API, consulta la GetObjectsezione AWS SDK per Go API Reference.

`GetObjectLegalHold`

Il seguente esempio di codice mostra come utilizzare`GetObjectLegalHold`.

SDK per Go V2

> **Note**
>
> C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
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
}
```

- Per i dettagli sull'API, consulta la GetObjectLegalHoldsezione AWS SDK per Go API Reference.

`GetObjectLockConfiguration`

Il seguente esempio di codice mostra come utilizzare`GetObjectLockConfiguration`.

SDK per Go V2

> **Note**
>
> C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
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
}
```

- Per i dettagli sull'API, consulta la GetObjectLockConfigurationsezione AWS SDK per Go API Reference.

`GetObjectRetention`

Il seguente esempio di codice mostra come utilizzare`GetObjectRetention`.

SDK per Go V2

> **Note**
>
> C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
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
}
```

- Per i dettagli sull'API, consulta la GetObjectRetentionsezione AWS SDK per Go API Reference.

`HeadBucket`

Il seguente esempio di codice mostra come utilizzare`HeadBucket`.

SDK per Go V2

> **Note**
>
> C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
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
}
```

- Per i dettagli sull'API, consulta la HeadBucketsezione AWS SDK per Go API Reference.

`ListBuckets`

Il seguente esempio di codice mostra come utilizzare`ListBuckets`.

SDK per Go V2

> **Note**
>
> C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
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
}
```

- Per i dettagli sull'API, consulta la ListBucketssezione AWS SDK per Go API Reference.

`ListObjectVersions`

Il seguente esempio di codice mostra come utilizzare`ListObjectVersions`.

SDK per Go V2

> **Note**
>
> C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
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
}
```

- Per i dettagli sull'API, consulta la ListObjectVersionssezione AWS SDK per Go API Reference.

`ListObjectsV2`

Il seguente esempio di codice mostra come utilizzare`ListObjectsV2`.

SDK per Go V2

> **Note**
>
> C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
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
}
```

- Per i dettagli sull'API, consulta la ListObjectsversione V2 in AWS SDK per Go API Reference.

`PutObject`

Il seguente esempio di codice mostra come utilizzare`PutObject`.

SDK per Go V2

> **Note**
>
> C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Inserisce un oggetto in un bucket utilizzando l'API di basso livello.

```
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
}
```

Carica un oggetto in un bucket utilizzando un gestore di trasferimento.

```
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
}
```

- Per i dettagli sull'API, consulta la PutObjectsezione AWS SDK per Go API Reference.

`PutObjectLegalHold`

Il seguente esempio di codice mostra come utilizzare`PutObjectLegalHold`.

SDK per Go V2

> **Note**
>
> C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
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
}
```

- Per i dettagli sull'API, consulta la PutObjectLegalHoldsezione AWS SDK per Go API Reference.

`PutObjectLockConfiguration`

Il seguente esempio di codice mostra come utilizzare`PutObjectLockConfiguration`.

SDK per Go V2

> **Note**
>
> C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Imposta la configurazione Object Lock di un bucket.

```
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
}
```

Imposta il periodo di conservazione predefinito di un bucket.

```
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
}
```

- Per i dettagli sull'API, consulta la PutObjectLockConfigurationsezione AWS SDK per Go API Reference.

`PutObjectRetention`

Il seguente esempio di codice mostra come utilizzare`PutObjectRetention`.

SDK per Go V2

> **Note**
>
> C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
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
}
```

- Per i dettagli sull'API, consulta la PutObjectRetentionsezione AWS SDK per Go API Reference.

Scenari

Creazione di un URL prefirmato

L'esempio di codice seguente mostra come creare un URL prefirmato per Amazon S3 e caricare un oggetto.

SDK per Go V2

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Crea funzioni che eseguono il wrap delle operazioni S3 di prefirma.

```
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
}
```

Esegui un esempio interattivo che generi e utilizzi presigned URLs per caricare, scaricare ed eliminare un oggetto S3.

```
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))
}
```

Definisci un wrapper di richieste HTTP utilizzato dall'esempio per effettuare richieste HTTP.

```
// 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)
}
```

Bloccare gli oggetti di Amazon S3

L’esempio di codice seguente mostra come utilizzare le funzionalità Object Lock di S3.

SDK per Go V2

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Esegue uno scenario interattivo che dimostra le funzionalità Object Lock di Amazon S3.

```
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))
}
```

Definisce una struttura per il wrapping delle azioni S3 utilizzate in questo esempio.

```
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
}
```

Eliminare le risorse.

```
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)
  }
 }
}
```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella documentazione di riferimento dell'API AWS SDK per Go .
  - GetObjectLegalHold
  - GetObjectLockConfiguration
  - GetObjectRetention
  - PutObjectLegalHold
  - PutObjectLockConfiguration
  - PutObjectRetention

Caricamento o download di file di grandi dimensioni

L'esempio di codice seguente mostra come caricare o scaricare file di grandi dimensioni in e da Amazon S3.

Per ulteriori informazioni, consulta Caricamento di un oggetto utilizzando il caricamento in più parti.

SDK per Go V2

> **Note**
>
> C'è dell'altro GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Crea funzioni che utilizzano i gestori di caricamento e scaricamento per suddividere i dati in parti e trasferirli contemporaneamente.

```
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
}
```

Esegue uno scenario interattivo che mostra come utilizzare i gestori di caricamento e download nel contesto.

```
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))
}
```

Esempi serverless

Invocazione di una funzione Lambda da un trigger Amazon S3

L'esempio di codice seguente mostra come implementare una funzione Lambda che riceve un evento attivato dal caricamento di un oggetto in un bucket S3. La funzione recupera il nome del bucket S3 e la chiave dell'oggetto dal parametro evento e chiama l'API Amazon S3 per recuperare e registrare il tipo di contenuto dell'oggetto.

SDK per Go V2

> **Note**
>
> C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Utilizzo di un evento S3 con Lambda tramite 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)
}
```

Esempi per Amazon SNS con SDK per Go V2

I seguenti esempi di codice mostrano come eseguire azioni e implementare scenari comuni utilizzando la AWS SDK per Go versione 2 con Amazon SNS.

Le azioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le azioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati.

Scenari: esempi di codice che mostrano come eseguire un'attività specifica chiamando più funzioni all'interno dello stesso servizio o combinate con altri Servizi AWS.

Ogni esempio include un link al codice sorgente completo, in cui vengono fornite le istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

- Nozioni di base
- Azioni
- Scenari
- Esempi serverless

Nozioni di base

Hello Amazon SNS

Il seguente esempio di codice mostra come iniziare a usare Amazon SNS.

SDK per Go V2

> **Note**
>
> C'è altro su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
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)
 }
}
}
```

- Per i dettagli sull'API, consulta la ListTopicssezione AWS SDK per Go API Reference.

Azioni

`CreateTopic`

Il seguente esempio di codice mostra come utilizzare`CreateTopic`.

SDK per Go V2

> **Note**
>
> C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
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
}
```

- Per i dettagli sull'API, consulta la CreateTopicsezione AWS SDK per Go API Reference.

`DeleteTopic`

Il seguente esempio di codice mostra come utilizzare`DeleteTopic`.

SDK per Go V2

> **Note**
>
> C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
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
}
```

- Per i dettagli sull'API, consulta la DeleteTopicsezione AWS SDK per Go API Reference.

`ListTopics`

Il seguente esempio di codice mostra come utilizzare`ListTopics`.

SDK per Go V2

> **Note**
>
> C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
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)
  }
 }
}
```

- Per i dettagli sull'API, consulta la ListTopicssezione AWS SDK per Go API Reference.

`Publish`

Il seguente esempio di codice mostra come utilizzare`Publish`.

SDK per Go V2

> **Note**
>
> C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
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
}
```

- Per informazioni dettagliate sull'API, consulta Pubblicazione nella documentazione di riferimento dell'API AWS SDK per Go .

`Subscribe`

Il seguente esempio di codice mostra come usare`Subscribe`.

SDK per Go V2

> **Note**
>
> C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Iscrivi una coda a un argomento con filtri opzionali.

```
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
}
```

- Per informazioni dettagliate sull'API, consulta Sottoscrizione nella documentazione di riferimento dell'API AWS SDK per Go .

Scenari

Pubblicazione di messaggi nelle code

L'esempio di codice seguente mostra come:

- Creazione di un argomento (FIFO o non FIFO).
- Sottoscrizione di diverse code all'argomento con la possibilità di applicare un filtro.
- Pubblicazione di un messaggio nell'argomento.
- Esame delle code per i messaggi ricevuti.

SDK per Go V2

> **Note**
>
> C'è dell'altro GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Esegui uno scenario interattivo al prompt dei comandi.

```
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))
}
```

Definisce una struttura per il wrapping delle azioni di Amazon SNS utilizzate in questo esempio.

```
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
}
```

Definisce una struttura per il wrapping delle azioni di Amazon SQS utilizzate in questo esempio.

```
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
}
```

Eliminare le risorse.

```
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)
  }
 }
}
```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella documentazione di riferimento dell'API AWS SDK per Go .
  - CreateQueue
  - CreateTopic
  - DeleteMessageBatch
  - DeleteQueue
  - DeleteTopic
  - GetQueueAttributes
  - Pubblicare
  - ReceiveMessage
  - SetQueueAttributes
  - Subscribe
  - Unsubscribe

Esempi serverless

Invocazione di una funzione Lambda da un trigger Amazon SNS

L'esempio di codice seguente mostra come implementare una funzione Lambda che riceve un evento attivato dal ricevimento di messaggi da un argomento SNS. La funzione recupera i messaggi dal parametro dell'evento e registra il contenuto di ogni messaggio.

SDK per Go V2

> **Note**
>
> C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Utilizzo di un evento SNS con Lambda tramite 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)
}
```

Esempi per Amazon SQS con SDK per Go V2

I seguenti esempi di codice mostrano come eseguire azioni e implementare scenari comuni utilizzando la versione AWS SDK per Go V2 con Amazon SQS.

Le azioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le azioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati.

Scenari: esempi di codice che mostrano come eseguire un'attività specifica chiamando più funzioni all'interno dello stesso servizio o combinate con altri Servizi AWS.

Ogni esempio include un link al codice sorgente completo, in cui vengono fornite le istruzioni su come configurare ed eseguire il codice nel contesto.

Argomenti

- Nozioni di base
- Azioni
- Scenari
- Esempi serverless

Nozioni di base

Hello Amazon SQS

Il seguente esempio di codice mostra come iniziare a usare Amazon SQS.

SDK per Go V2

> **Note**
>
> C'è altro su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
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)
  }
 }
}
```

- Per i dettagli sull'API, consulta la ListQueuessezione AWS SDK per Go API Reference.

Azioni

`CreateQueue`

Il seguente esempio di codice mostra come utilizzare`CreateQueue`.

SDK per Go V2

> **Note**
>
> C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
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
}
```

- Per i dettagli sull'API, consulta la CreateQueuesezione AWS SDK per Go API Reference.

`DeleteMessageBatch`

Il seguente esempio di codice mostra come utilizzare`DeleteMessageBatch`.

SDK per Go V2

> **Note**
>
> C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
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
}
```

- Per i dettagli sull'API, consulta la DeleteMessageBatchsezione AWS SDK per Go API Reference.

`DeleteQueue`

Il seguente esempio di codice mostra come utilizzare`DeleteQueue`.

SDK per Go V2

> **Note**
>
> C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
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
}
```

- Per i dettagli sull'API, consulta la DeleteQueuesezione AWS SDK per Go API Reference.

`GetQueueAttributes`

Il seguente esempio di codice mostra come utilizzare`GetQueueAttributes`.

SDK per Go V2

> **Note**
>
> C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
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
}
```

- Per i dettagli sull'API, consulta la GetQueueAttributessezione AWS SDK per Go API Reference.

`ListQueues`

Il seguente esempio di codice mostra come utilizzare`ListQueues`.

SDK per Go V2

> **Note**
>
> C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
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)
 }
}
}
```

- Per i dettagli sull'API, consulta la ListQueuessezione AWS SDK per Go API Reference.

`ReceiveMessage`

Il seguente esempio di codice mostra come utilizzare`ReceiveMessage`.

SDK per Go V2

> **Note**
>
> C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
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
}
```

- Per i dettagli sull'API, consulta la ReceiveMessagesezione AWS SDK per Go API Reference.

`SetQueueAttributes`

Il seguente esempio di codice mostra come utilizzare`SetQueueAttributes`.

SDK per Go V2

> **Note**
>
> C'è altro da fare GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
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
```

- Per i dettagli sull'API, consulta la SetQueueAttributessezione AWS SDK per Go API Reference.

Scenari

Pubblicazione di messaggi nelle code

L'esempio di codice seguente mostra come:

- Creazione di un argomento (FIFO o non FIFO).
- Sottoscrizione di diverse code all'argomento con la possibilità di applicare un filtro.
- Pubblicazione di un messaggio nell'argomento.
- Esame delle code per i messaggi ricevuti.

SDK per Go V2

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Esegui uno scenario interattivo al prompt dei comandi.

```
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))
}
```

Definisce una struttura per il wrapping delle azioni di Amazon SNS utilizzate in questo esempio.

```
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
}
```

Definisce una struttura per il wrapping delle azioni di Amazon SQS utilizzate in questo esempio.

```
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
}
```

Eliminare le risorse.

```
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)
  }
 }
}
```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella documentazione di riferimento dell'API AWS SDK per Go .
  - CreateQueue
  - CreateTopic

- DeleteMessageBatch
- DeleteQueue
- DeleteTopic
- GetQueueAttributes
- Pubblicare
- ReceiveMessage
- SetQueueAttributes
- Subscribe
- Unsubscribe

Esempi serverless

Invocazione di una funzione Lambda da un trigger Amazon SQS

L'esempio di codice seguente mostra come implementare una funzione Lambda che riceve un evento attivato dalla ricezione di messaggi da una coda SQS. La funzione recupera i messaggi dal parametro dell'evento e registra il contenuto di ogni messaggio.

SDK per Go V2

> **Note**
>
> C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Utilizzo di un evento SQS con Lambda tramite 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)
}
```

Segnalazione di errori di elementi batch per funzioni Lambda con un trigger Amazon SQS

L'esempio di codice seguente mostra come implementare una risposta batch parziale per funzioni Lambda che ricevono eventi da una coda SQS. La funzione riporta gli errori degli elementi batch nella risposta, segnalando a Lambda di riprovare tali messaggi in un secondo momento.

SDK per Go V2

> **Note**
>
> C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Segnalazione di errori di elementi batch di SQS con Lambda tramite 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, sqsEvent events.SQSEvent) (map[string]interface{},
 error) {
 batchItemFailures := []map[string]interface{}{}

 for _, message := range sqsEvent.Records {
  if len(message.Body) > 0 {
   // Your message processing condition here
   fmt.Printf("Successfully processed message: %s\n", message.Body)
  } else {
   // Message processing failed
   fmt.Printf("Failed to process message %s\n", message.MessageId)
   batchItemFailures = append(batchItemFailures, map[string]interface{}
{"itemIdentifier": message.MessageId})
  }
 }

 sqsBatchResponse := map[string]interface{}{
  "batchItemFailures": batchItemFailures,
 }
 return sqsBatchResponse, nil
}

func main() {
 lambda.Start(handler)
}
```

Migrazione alla v2 AWS SDK per Go

Versione minima Go

AWS SDK per Go Richiede una versione minima di Go 1.23. L'ultima versione di Go può essere scaricata dalla pagina Download. Consulta la Cronologia delle versioni per ulteriori informazioni su ogni versione di Go e sulle informazioni pertinenti necessarie per l'aggiornamento.

Modularizzazione

AWS SDK per Go È stato aggiornato per sfruttare i moduli Go, che sono diventati la modalità di sviluppo predefinita in Go 1.13. Alcuni pacchetti forniti dall'SDK sono stati modularizzati e sono rispettivamente distribuiti e rilasciati in modo indipendente. Questa modifica consente una migliore modellazione delle dipendenze delle applicazioni e consente all'SDK di fornire nuove caratteristiche e funzionalità che seguono la strategia di controllo delle versioni del modulo Go.

Di seguito sono elencati alcuni moduli Go forniti dall'SDK:

Modulo	Description
`github.com/aws/aws-sdk-go-v2`	Il core dell'SDK
`github.com/aws/aws-sdk-go-v2/config`	Caricamento della configurazione condivisa
`github.com/aws/aws-sdk-go-v2/credentials`	AWS Fornitori di credenziali
`github.com/aws/aws-sdk-go-v2/feature/ec2/imds`	Client del servizio di metadati di Amazon EC2 Instance

I client di servizio e i moduli di utilità di livello superiore dell'SDK sono annidati nei seguenti percorsi di importazione:

Importa root	Description
`github.com/aws/aws-sdk-go-v2/service/`	Moduli Service Client
`github.com/aws/aws-sdk-go-v2/feature/`	Utilità di alto livello per servizi come Amazon S3 Transfer Manager

Caricamento della configurazione

Il pacchetto di sessione e le funzionalità associate vengono sostituiti con un sistema di configurazione semplificato fornito dal pacchetto config. Il `config` pacchetto è un modulo Go separato e può essere incluso nelle dipendenze dell'applicazione con. `go get`

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

La sessione. NEW, session.NewSessionNewSessionWithOptions, e Session.must deve essere migrato a config. LoadDefaultConfig.

Il `config` pacchetto fornisce diverse funzioni di supporto che aiutano a sovrascrivere il caricamento della configurazione condivisa a livello di codice. I nomi di queste funzioni sono preceduti da un'opzione `With` seguita da un'opzione che sostituiscono. Diamo un'occhiata ad alcuni esempi di come migrare l'utilizzo del pacchetto. `session`

Per ulteriori informazioni sul caricamento della configurazione condivisa, consultaConfigurare l'SDK.

Esempi

Migrazione da a NewSession LoadDefaultConfig

L'esempio seguente mostra come viene migrato l'utilizzo di parametri `session.NewSession` senza argomenti aggiuntivi. `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
}
```

Migrazione da NewSession con le opzioni AWS.config

L'esempio mostra come migrare l'override dei valori durante il caricamento della configurazione. `aws.Config` È possibile fornire una o più funzioni di `config.With*` supporto per `config.LoadDefaultConfig` sovrascrivere i valori di configurazione caricati. In questo esempio la AWS regione viene sostituita dall'utilizzo della configurazione. `us-west-2` WithRegionfunzione di supporto.

```
// 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
}
```

Migrazione da NewSessionWithOptions

Questo esempio mostra come migrare i valori di override durante il caricamento della configurazione. È possibile fornire zero o più funzioni di `config.With*` supporto per `config.LoadDefaultConfig` sovrascrivere i valori di configurazione caricati. In questo esempio mostriamo come sovrascrivere il profilo di destinazione utilizzato durante il caricamento della configurazione condivisa dell' AWS SDK.

```
// 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
}
```

Mocking e `*iface`

I `*iface` pacchetti e le interfacce in esso contenuti (ad esempio S3iFace.S3api) sono stati rimossi. Queste definizioni di interfaccia non sono stabili poiché vengono interrotte ogni volta che un servizio aggiunge una nuova operazione.

L'utilizzo di `*iface` deve essere sostituito da interfacce definite dal chiamante con ambito per le operazioni di servizio utilizzate:

```
// 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)
}
```

Per ulteriori informazioni, consulta Test unitario con la AWS SDK per Go v2.

Credenziali e fornitori di credenziali

Il pacchetto aws/credentials e i provider di credenziali associati sono stati trasferiti nella posizione del pacchetto delle credenziali. Il `credentials` pacchetto è un modulo Go che si recupera utilizzando. `go get`

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

La versione AWS SDK per Go v2 aggiorna i AWS Credential Provider per fornire un'interfaccia coerente per il recupero delle credenziali. AWS Ogni provider implementa aws. CredentialsProviderinterfaccia, che definisce un `Retrieve` metodo che restituisce un`(aws.Credentials, error)`. aws.Credentials che è analogo al tipo v1 Credentials.value. AWS SDK per Go

È necessario `aws.CredentialsProvider` avvolgere gli oggetti con aws. CredentialsCacheper consentire la memorizzazione nella cache delle credenziali. Si usa NewCredentialsCacheper costruire un oggetto. `aws.CredentialsCache` Per impostazione predefinita, le credenziali configurate da `config.LoadDefaultConfig` sono racchiuse con. `aws.CredentialsCache`

La tabella seguente elenca le modifiche alla posizione dei provider di AWS credenziali dalla AWS SDK per Go v1 alla v2.

Name	Importazione V1	Importazione V2
Credenziali del ruolo Amazon EC2 IAM	`github.com/aws/aws-sdk-go/aws/credentials/ec2rolecreds`	`github.com/aws/aws-sdk-go-v2/credentials/ec2rolecreds`
Credenziali degli endpoint	`github.com/aws/aws-sdk-go/aws/credentials/endpointcreds`	`github.com/aws/aws-sdk-go-v2/credentials/endpointcreds`
Credenziali di processo	`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`

Credenziali statiche

Applicazioni che utilizzano credenziali. NewStaticCredentialsper costruire credenziali statiche a livello di codice devono utilizzare credenziali. NewStaticCredentialsProvider.

Esempio

```
// 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
}
```

Credenziali del ruolo Amazon EC2 IAM

È necessario migrare l'utilizzo NewCredentialse l'utilizzo di NewCredentialsWithClientNew.

Il `ec2rolecreds` pacchetto utilizza le opzioni funzionali `ec2rolecreds.New` di EC2RoleCreds.options come input, consentendoti di sovrascrivere lo specifico client Amazon EC2 Instance Metadata Service da utilizzare o di sovrascrivere la finestra di scadenza delle credenziali.

Esempio

```
// 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
}
```

Credenziali degli endpoint

È necessario migrare l'utilizzo NewCredentialsCliente l'utilizzo NewProviderClientdi New.

La `New` funzione del `endpointcreds` pacchetto accetta un argomento di tipo stringa contenente l'URL di un endpoint HTTP o HTTPS da cui recuperare le credenziali e le opzioni funzionali di EndpointCreds.options per modificare il provider delle credenziali e sovrascrivere impostazioni di configurazione specifiche.

Credenziali di processo

È necessario migrare l'utilizzo di NewCredentialsNewCredentialsCommand, e NewCredentialsTimeoututilizzare NewProvidero. NewProviderCommand

La `NewProvider` funzione del `processcreds` pacchetto accetta un argomento di stringa che è il comando da eseguire nella shell dell'ambiente host e le opzioni funzionali di Options per modificare il provider delle credenziali e sovrascrivere impostazioni di configurazione specifiche.

`NewProviderCommand`richiede un'implementazione dell'NewCommandBuilderinterfaccia che definisce comandi di processo più complessi che potrebbero accettare uno o più argomenti della riga di comando o avere determinati requisiti dell'ambiente di esecuzione. DefaultNewCommandBuilderimplementa questa interfaccia e definisce un generatore di comandi per un processo che richiede più argomenti della riga di comando.

Esempio

```
// 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 Credenziali

AssumeRole

È necessario migrare l'utilizzo di NewCredentialse da utilizzare. NewCredentialsWithClientNewAssumeRoleProvider

La `NewAssumeRoleProvider` funzione del `stscreds` pacchetto deve essere chiamata con un STS.Client e il AWS Identity and Access Management Role ARN deve essere assunto dalle credenziali configurate fornite`sts.Client`. È inoltre possibile fornire una serie di opzioni funzionali o AssumeRoleOptionsmodificare altre impostazioni opzionali del provider.

Esempio

```
// 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

È necessario migrare l'utilizzo di NewWebIdentityCredentialsNewWebIdentityRoleProvider, e NewWebIdentityRoleProviderWithTokenda utilizzare NewWebIdentityRoleProvider.

La `NewWebIdentityRoleProvider` funzione del `stscreds` pacchetto deve essere chiamata con un STS.Client e l' AWS Identity and Access Management ARN del ruolo deve essere assunto utilizzando le credenziali configurate fornite `sts.Client` e un'implementazione di a IdentityTokenRetrieverper fornire il token 2.0 o OAuth OpenID Connect ID. IdentityTokenFileè un file `IdentityTokenRetriever` che può essere utilizzato per fornire il token di identità Web da un file che si trova nel file system host dell'applicazione. È inoltre possibile fornire una serie di opzioni funzionali WebIdentityRoleOptionsper modificare altre impostazioni opzionali per il provider.

Esempio

```
// 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
}
```

Client di servizio

AWS SDK per Go fornisce moduli client di servizio annidati nel percorso di `github.com/aws/aws-sdk-go-v2/service` importazione. Ogni client di servizio è contenuto in un pacchetto Go che utilizza l'identificatore univoco di ogni servizio. La tabella seguente fornisce alcuni esempi di percorsi di importazione dei servizi in. AWS SDK per Go

Nome del servizio	Percorso di importazione V1	Percorso di importazione V2
Simple Storage Service (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 Registri Amazon	`github.com/aws/aws-sdk-go/service/cloudwatchlogs`	`github.com/aws/aws-sdk-go-v2/service/cloudwatchlogs`

Ogni pacchetto client di servizio è un modulo Go con versione indipendente. Per aggiungere il client di servizio come dipendenza dell'applicazione, utilizzate il `go get` comando con il percorso di importazione del servizio. Ad esempio, per aggiungere il client Amazon S3 alle tue dipendenze, usa

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

Costruzione del cliente

È possibile creare client AWS SDK per Go utilizzando le funzioni `New` o il `NewFromConfig` costruttore nel pacchetto del client. Durante la migrazione dalla AWS SDK per Go v1, si consiglia di utilizzare la `NewFromConfig` variante, che restituirà un nuovo client di servizio utilizzando i valori di un. `aws.Config` Il `aws.Config` valore sarà stato creato durante il caricamento della configurazione condivisa dell'SDK utilizzando. `config.LoadDefaultConfig` Per i dettagli sulla creazione di client di servizio, consultaUsa la AWS SDK per Go v2 con i servizi AWS.

Esempio 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)
```

Esempio 2: sovrascrivere le impostazioni del client

```
// 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"
})
```

Endpoints

Il pacchetto endpoints non esiste più in. AWS SDK per Go Ogni client di servizio ora incorpora i metadati AWS degli endpoint richiesti nel pacchetto client. Ciò riduce la dimensione binaria complessiva delle applicazioni compilate non includendo più i metadati degli endpoint per i servizi non utilizzati dall'applicazione.

Inoltre, ogni servizio ora espone la propria interfaccia per la risoluzione degli endpoint in. `EndpointResolverV2` Ogni API utilizza un set unico di parametri per un servizio`EndpointParameters`, i cui valori provengono dall'SDK da varie posizioni quando viene richiamata un'operazione.

Per impostazione predefinita, i client del servizio utilizzano la AWS regione configurata per risolvere l'endpoint del servizio per la regione di destinazione. Se l'applicazione richiede un endpoint personalizzato, è possibile specificare un comportamento personalizzato sul `EndpointResolverV2` campo della `aws.Config` struttura. Se la tua applicazione implementa un EndPoints.resolver personalizzato, devi migrarlo per renderlo conforme a questa nuova interfaccia per servizio.

Per ulteriori informazioni sugli endpoint e sull'implementazione di un resolver personalizzato, consulta. Configurazione degli endpoint client

Autenticazione

AWS SDK per Go Supporta un comportamento di autenticazione più avanzato, che consente l'uso di nuove funzionalità di AWS servizio come codecatalyst e S3 Express One Zone. Inoltre, questo comportamento può essere personalizzato in base al cliente.

Richiamo delle operazioni API

Il numero di metodi operativi dei client di servizio è stato ridotto in modo significativo. I `<OperationName>` metodi `<OperationName>Request<OperationName>WithContext`, e sono stati tutti consolidati in un unico metodo operativo,`<OperationName>`.

Esempio

L'esempio seguente mostra come le chiamate all' PutObject operazione Amazon S3 verrebbero migrate dalla AWS SDK per Go v1 alla v2.

```
// 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
})
```

Tipi di dati di servizio

I tipi di input e output di primo livello di un'operazione si trovano nel pacchetto client del servizio. I tipi di input e output per una determinata operazione seguono lo schema di `<OperationName>Input` e`<OperationName>Output`, where `OperationName` è il nome dell'operazione che si sta richiamando. Ad esempio, la forma di input e output per l' PutObject operazione Amazon S3 sono PutObjectInpute PutObjectOutputrispettivamente.

Tutti gli altri tipi di dati di servizio, diversi dai tipi di input e output, sono stati migrati al `types` pacchetto situato nella gerarchia dei percorsi di importazione dei pacchetti del client di servizio. Ad esempio, s3. AccessControlPolicytype si trova ora in types. AccessControlPolicy.

Valori di enumerazione

L'SDK ora offre un'esperienza digitata per tutti i campi di enumerazione delle API. Invece di utilizzare un valore letterale di stringa copiato dalla documentazione di riferimento dell'API di servizio, ora puoi utilizzare uno dei tipi concreti presenti nel pacchetto del client del servizio. `types` Ad esempio, puoi fornire all' PutObjectInput operazione Amazon S3 un ACL da applicare a un oggetto. Nella AWS SDK per Go v1, questo parametro era un tipo. `*string` Nel AWS SDK per Go, questo parametro è ora un tipo. ObjectCannedACL. Il `types` pacchetto fornisce le costanti generate per i valori di enumerazione validi che possono essere assegnati a questo campo. Ad esempio tipi. ObjectCannedACLPrivateè la costante per il valore ACL predefinito «privato». Questo valore può essere utilizzato al posto della gestione delle costanti di stringa all'interno dell'applicazione.

Parametri del puntatore

La AWS SDK per Go v1 richiedeva che i riferimenti ai puntatori fossero passati per tutti i parametri di input alle operazioni di servizio. La AWS SDK per Go v2 ha semplificato l'esperienza con la maggior parte dei servizi eliminando la necessità di passare i valori di input come puntatori, ove possibile. Questa modifica significa che le operazioni di molti client di servizio non richiedono più che l'applicazione trasmetta riferimenti ai puntatori per i seguenti tipi:`uint8`,,,`uint16`,`uint32`,, `int8int16`,`int32`. `float32` `float64` `bool` Allo stesso modo, i tipi di elementi slice e map sono

stati aggiornati di conseguenza per indicare se i relativi elementi devono essere passati come riferimenti puntatori.

Il pacchetto aws contiene funzioni di supporto per la creazione di puntatori per i tipi integrati di Go, questi helper dovrebbero essere usati per gestire più facilmente la creazione di tipi di puntatori per questi tipi di Go. Allo stesso modo, vengono forniti metodi di supporto per dereferenziare in modo sicuro i valori dei puntatori per questi tipi. Ad esempio, la funzione AWS.String converte da ⇒. `string` `*string` Al contrario, l'aws. ToStringconverte da `*string` ⇒. `string` Quando si aggiorna l'applicazione dalla AWS SDK per Go v1 alla v2, è necessario migrare l'utilizzo degli helper per la conversione dai tipi di puntatore alle varianti senza puntatore. Ad esempio, aws. StringValuedeve essere aggiornato `aaws.ToString`.

Tipi di errori

Sfrutta appieno la AWS SDK per Go funzionalità di risoluzione degli errori introdotta in Go 1.13. I servizi che modellano le risposte agli errori hanno generato tipi disponibili nel `types` pacchetto del client che possono essere utilizzati per verificare se un errore operativo del client è stato causato da uno di questi tipi. Ad esempio, l'`GetObject`operazione Amazon S3 può restituire un `NoSuchKey` errore se si tenta di recuperare una chiave oggetto che non esiste. Puoi utilizzare Errors.as per verificare se l'errore di operazione restituito è di tipo A. NoSuchKeyerrore. Se un servizio non modella un tipo specifico di errore, puoi utilizzare la fucina. APIErrortipo di interfaccia per controllare il codice di errore e il messaggio restituiti dal servizio. Questa funzionalità sostituisce awSerr.Error e l'altra funzionalità awserr della v1. AWS SDK per Go Per ulteriori informazioni sulla gestione degli errori, vedere. Gestione degli errori nella AWS SDK per Go V2

Esempio

```
// 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
}
```

Impaginatori

Gli impaginatori per le operazioni di servizio non vengono più richiamati come metodi sul client del servizio. Per utilizzare un paginatore per un'operazione, è necessario creare un impaginatore per un'operazione utilizzando uno dei metodi del costruttore del paginatore. Ad esempio, per utilizzare paginate sull'operazione Amazon `ListObjectsV2` S3, è necessario creare il relativo impaginatore utilizzando s3. NewListObjectsPaginatore V2. Questo costruttore restituisce un ListObjectsV2Paginator che fornisce i metodi `HasMorePages` e `NextPage` per determinare se ci sono più pagine da recuperare e richiama l'operazione per recuperare rispettivamente la pagina successiva. Maggiori dettagli sull'utilizzo degli impaginatori SDK sono disponibili all'indirizzo. Utilizzo di Operation Paginators

Diamo un'occhiata a un esempio di come migrare da un paginatore AWS SDK per Go v1 all'equivalente v2. AWS SDK per Go

Esempio

```
// 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

I camerieri addetti alle operazioni di servizio non vengono più richiamati come metodi sul client del servizio. Per utilizzare un cameriere devi prima creare il tipo di cameriere desiderato, quindi invocare il metodo di attesa. Ad esempio, per attendere che esista un Amazon S3 Bucket, devi creare un cameriere. `BucketExists` Usa s3. NewBucketExistsWaitercostruttore per creare un s3. BucketExistsWaiter. `s3.BucketExistsWaiter`Fornisce un `Wait` metodo che può essere utilizzato per attendere che un bucket diventi disponibile.

Richieste predefinite

L'SDK V1 supportava tecnicamente la preassegnazione di qualsiasi operazione AWS SDK, tuttavia, ciò non rappresenta con precisione ciò che è effettivamente supportato a livello di servizio (e in realtà la maggior parte delle AWS operazioni di servizio non supporta la prefirma).

AWS SDK per Go risolve questo problema esponendo `PresignClient` implementazioni specifiche nei pacchetti di servizi con operazioni presignabili specifiche per il supporto. APIs

Nota: se a un servizio manca il supporto di prefirma per un'operazione che stavi utilizzando con successo in SDK v1, faccelo sapere segnalando un problema su. GitHub

Utilizza Presign e PresignRequestdeve essere convertito per utilizzare client di prefirma specifici del servizio.

L'esempio seguente mostra come migrare la prefirma di una richiesta S3: 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)
}
```

Richiedi la personalizzazione

L'API monolitica Request.Request è stata ricompartimentalizzata.

Ingresso/uscita dell'operazione

I `Request` campi opachi `Params` e`Data`, che contengono rispettivamente le strutture di input e output dell'operazione, sono ora accessibili all'interno di specifiche fasi del middleware come input/output:

Gestori di richieste che fanno riferimento e devono essere migrati al middleware. `Request.Params` `Request.Data`

migrando **`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))
    // ...
}
```

migrando **`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)
    // ...
}
```

richiesta/risposta HTTP

I `HTTPResponse` campi `HTTPRequest` e di `Request` sono ora esposti in fasi middleware specifiche. Poiché il middleware è indipendente dal trasporto, è necessario eseguire un'asserzione di tipo sull'input o sull'output del middleware per rivelare la richiesta o la risposta HTTP sottostante.

Gestori di richieste che fanno riferimento e che devono essere migrati al middleware. `Request.HTTPRequest` `Request.HTTPResponse`

migrando **`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", "..."))
    // ...
}
```

fasi del gestore

Le fasi del middleware SDK v2 sono le successive alle fasi del gestore v1.

La tabella seguente fornisce una mappatura approssimativa delle fasi del gestore v1 alla loro posizione equivalente all'interno dello stack di middleware V2:

nome del gestore v1	fase middleware v2
Convalida	Inizializzazione
Creazione	Serializza
Sign	Finalizzare
Invia	n/a (1)
ValidateResponse	Deserializza
Demolisci sceriffo	Deserializza
UnmarshalMetadata	Deserializza

nome del gestore v1	fase middleware v2
UnmarshalError	Deserializza
Riprova	Finalizza, dopo il `"Retry"` middleware (2)
AfterRetry	Finalizza, prima del `"Retry"` middleware, post- (2,3) `next.HandleFinalize()`
CompleteAttempt	Finalizzazione, fine del passaggio
Completa	Inizializzazione, inizio del passaggio, post- `next.HandleInitialize()` (3)

(1) La `Send` fase in v1 è effettivamente il round-trip del client HTTP incluso nella v2. Questo comportamento è controllato dal campo sulle opzioni del `HTTPClient` client.

(2) Qualsiasi middleware dopo il `"Retry"` middleware nella fase Finalize farà parte del ciclo di riprova.

(3) Lo «stack» del middleware al momento dell'operazione è integrato in una funzione di gestione decorata ripetutamente. Ogni gestore è responsabile della chiamata a quello successivo nella catena. Ciò significa implicitamente che una fase del middleware può agire anche DOPO che è stata chiamata la fase successiva.

Ad esempio, per la fase di inizializzazione, che si trova in cima allo stack, ciò significa che i middleware di inizializzazione che agiscono dopo aver chiamato il gestore successivo operano effettivamente alla fine della richiesta:

```
// 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
}
```

Funzionalità

Servizio di metadati Amazon EC2 Instance

AWS SDK per Go Fornisce un client Amazon EC2 Instance Metadata Service (IMDS) che puoi utilizzare per interrogare l'IMDS locale durante l'esecuzione della tua applicazione su un'istanza Amazon. EC2 Il client IMDS è un modulo Go separato che può essere aggiunto all'applicazione utilizzando

```
go get github.com/aws/aws-sdk-go-v2/feature/ec2/imds
```

Il costruttore del client e le operazioni del metodo sono state aggiornate per corrispondere al design degli altri client di servizi SDK.

Esempio

```
// 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

Il gestore di trasferimenti Amazon S3 è disponibile per la gestione simultanea di caricamenti e download di oggetti. Questo pacchetto si trova in un modulo Go esterno al percorso di importazione del client di servizio. Questo modulo può essere recuperato utilizzando`go get github.com/aws/aws-sdk-go-v2/feature/s3/manager`.

s3. NewUploadere s3. NewUploaderWithClientsono stati sostituiti con il gestore dei metodi del costruttore. NewUploaderper creare un client Upload manager.

s3. NewDownloadere s3. NewDownloaderWithClientsono stati sostituiti con un gestore di metodi a singolo costruttore. NewDownloaderper creare un client di gestione dei download.

Utilità di CloudFront firma Amazon

AWS SDK per Go Fornisce utilità di CloudFront firma Amazon in un modulo Go esterno al percorso di importazione del client di servizio. Questo modulo può essere recuperato utilizzando. `go get`

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

Client di crittografia Amazon S3

A partire da AWS SDK per Go, il client di crittografia Amazon S3 è un modulo separato in AWS Crypto Tools. L'ultima versione del client di crittografia S3 per Go, 3.x, è ora disponibile all'indirizzo. GitHub Questo modulo può essere recuperato utilizzando: `go get`

```
go get github.com/aws/amazon-s3-encryption-client-go/v3
```

I due client separati `EncryptionClient` (v1, v2) e `DecryptionClient` (v1, v2) sono APIs stati sostituiti con un singolo client, S3 EncryptionClient V3, che espone sia la funzionalità di crittografia che quella di decrittografia.

Come altri client di servizio, l'operazione è stata condensata: AWS SDK per Go APIs

- La `GetObjectGetObjectRequest`, e la `GetObjectWithContext` decrittografia APIs sono sostituite da. GetObject
- La `PutObjectWithContext` crittografia `PutObjectPutObjectRequest`, e viene sostituita APIs da. PutObject

Per informazioni su come migrare alla versione principale 3.x del client di crittografia, consulta questa guida.

Modifiche alle personalizzazioni del servizio

Simple Storage Service (Amazon S3)

Durante la migrazione dalla AWS SDK per Go v1 alla v2, una modifica importante di cui tenere conto riguarda la gestione delle chiavi `SSECustomerKey` utilizzate per la crittografia lato server con chiavi fornite dal cliente (SSE-C). Nella AWS SDK per Go v1, la codifica di Base64 veniva gestita internamente dall'SDK. `SSECustomerKey` In SDK v2, questa codifica automatica è stata rimossa e ora è necessario codificarla manualmente in Base64 prima di passarla all'SDK. `SSECustomerKey`

Esempio di regolazione:

```
// 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
```

Sicurezza in AWS SDK per Go

La sicurezza del cloud AWS è la massima priorità. In qualità di AWS cliente, puoi beneficiare di data center e architetture di rete progettati per soddisfare i requisiti delle organizzazioni più sensibili alla sicurezza.

La sicurezza è una responsabilità condivisa tra te e te. AWS Il modello di responsabilità condivisa descrive questo aspetto come sicurezza del cloud e sicurezza nel cloud:

- Sicurezza del cloud: AWS è responsabile della protezione dell'infrastruttura che gestisce AWS i servizi in Cloud AWS. AWS fornisce inoltre servizi che è possibile utilizzare in modo sicuro. I revisori esterni testano e verificano regolarmente l'efficacia della nostra sicurezza nell'ambito dei AWS Programmi di AWS conformità dei Programmi di conformità dei di . Per ulteriori informazioni sui programmi di conformità applicabili AWS SDK per Go, consulta AWS Servizi nell'ambito del programma di conformitàAWS .
- Sicurezza nel cloud: la tua responsabilità è determinata dal AWS servizio che utilizzi. Sei anche responsabile di altri fattori, tra cui la riservatezza dei dati, i requisiti della tua azienda e le leggi e normative vigenti.

Questa documentazione ti aiuta a capire come applicare il modello di responsabilità condivisa durante l'utilizzo AWS SDK per Go. I seguenti argomenti mostrano come eseguire la configurazione AWS SDK per Go per soddisfare gli obiettivi di sicurezza e conformità. Imparerai anche a utilizzare altri AWS servizi che ti aiutano a monitorare e proteggere AWS SDK per Go le tue risorse.

Argomenti

- Protezione dei dati in AWS SDK per Go
- Convalida della conformità per AWS SDK per Go
- Resilienza in AWS SDK per Go

Protezione dei dati in AWS SDK per Go

Il modello di responsabilità AWS condivisa modello di di si applica alla protezione dei dati in AWS SDK per Go. Come descritto in questo modello, AWS è responsabile della protezione dell'infrastruttura globale che gestisce tutti i Cloud AWS. L'utente è responsabile del controllo dei contenuti ospitati su questa infrastruttura. L'utente è inoltre responsabile della configurazione della

protezione e delle attività di gestione per i Servizi AWS utilizzati. Per ulteriori informazioni sulla privacy dei dati, vedi le Domande frequenti sulla privacy dei dati. Per informazioni sulla protezione dei dati in Europa, consulta il post del blog relativo al Modello di responsabilità condivisa AWS e GDPR nel Blog sulla sicurezza AWS .

Ai fini della protezione dei dati, consigliamo di proteggere Account AWS le credenziali e configurare i singoli utenti con AWS IAM Identity Center or AWS Identity and Access Management (IAM). In tal modo, a ogni utente verranno assegnate solo le autorizzazioni necessarie per svolgere i suoi compiti. Ti suggeriamo, inoltre, di proteggere i dati nei seguenti modi:

- Utilizza l'autenticazione a più fattori (MFA) con ogni account.
- Usa SSL/TLS per comunicare con le risorse. AWS È richiesto TLS 1.2 ed è consigliato TLS 1.3.
- Configura l'API e la registrazione delle attività degli utenti con. AWS CloudTrail Per informazioni sull'utilizzo dei CloudTrail percorsi per acquisire AWS le attività, consulta Lavorare con i CloudTrail percorsi nella Guida per l'AWS CloudTrail utente.
- Utilizza soluzioni di AWS crittografia, insieme a tutti i controlli di sicurezza predefiniti all'interno Servizi AWS.
- Utilizza i servizi di sicurezza gestiti avanzati, come Amazon Macie, che aiutano a individuare e proteggere i dati sensibili archiviati in Amazon S3.
- Se hai bisogno di moduli crittografici convalidati FIPS 140-3 per accedere AWS tramite un'interfaccia a riga di comando o un'API, usa un endpoint FIPS. Per ulteriori informazioni sugli endpoint FIPS disponibili, consulta il Federal Information Processing Standard (FIPS) 140-3.

Ti consigliamo di non inserire mai informazioni riservate o sensibili, ad esempio gli indirizzi e-mail dei clienti, nei tag o nei campi di testo in formato libero, ad esempio nel campo Nome. Ciò include quando lavori AWS SDK per Go o Servizi AWS utilizzi la console, l'API o. AWS CLI AWS SDKs I dati inseriti nei tag o nei campi di testo in formato libero utilizzati per i nomi possono essere utilizzati per i la fatturazione o i log di diagnostica. Quando fornisci un URL a un server esterno, ti suggeriamo vivamente di non includere informazioni sulle credenziali nell'URL per convalidare la tua richiesta al server.

Convalida della conformità per AWS SDK per Go

Per sapere se un Servizio AWS programma rientra nell'ambito di specifici programmi di conformità, consulta Servizi AWS la sezione Scope by Compliance Program Servizi AWS e scegli il programma

di conformità che ti interessa. Per informazioni generali, consulta Programmi di AWS conformità Programmi di di .

È possibile scaricare report di audit di terze parti utilizzando AWS Artifact. Per ulteriori informazioni, consulta Scaricamento dei report in AWS Artifact .

La vostra responsabilità di conformità durante l'utilizzo Servizi AWS è determinata dalla sensibilità dei dati, dagli obiettivi di conformità dell'azienda e dalle leggi e dai regolamenti applicabili. Per ulteriori informazioni sulla responsabilità di conformità durante l'utilizzo Servizi AWS, consulta la Documentazione AWS sulla sicurezza.

Resilienza in AWS SDK per Go

L'infrastruttura AWS globale è costruita attorno a Regioni AWS zone di disponibilità. Regioni AWS forniscono più zone di disponibilità fisicamente separate e isolate, collegate con reti a bassa latenza, ad alto throughput e altamente ridondanti. Con le zone di disponibilità, puoi progettare e gestire applicazioni e database che eseguono automaticamente il failover tra zone di disponibilità senza interruzioni. Le zone di disponibilità sono più disponibili, tolleranti ai guasti e scalabili rispetto alle infrastrutture a data center singolo o multiplo tradizionali.

Per ulteriori informazioni sulle zone di disponibilità, vedere Global Regioni AWS Infrastructure.AWS

Cronologia dei documenti per la Guida per gli sviluppatori AWS SDK per Go v2

La tabella seguente descrive le versioni della documentazione per la AWS SDK per Go v2.

Modifica	Descrizione	Data
Intercettori HTTP	Nuova sezione sugli intercett ori HTTP, i loro vantaggi rispetto al middleware, gli hook disponibili e i metodi di registrazione.	4 settembre 2025
Protezione dell'integrità dei dati con checksum	Contenuto aggiornato con dettagli sul calcolo automatico dei checksum.	16 gennaio 2025
Versione iniziale	Versione iniziale della Guida per gli AWS SDK per Go sviluppatori v2	31 ottobre 2024

Le traduzioni sono generate tramite traduzione automatica. In caso di conflitto tra il contenuto di una traduzione e la versione originale in Inglese, quest'ultima prevarrà.