Developer Guide

AWS SDK for Go (version 1)

AWS SDK for Go (version 1): Developer Guide

Table of Contents

..... viii
What is the AWS SDK for Go 1
More Info 1
Maintenance and support for SDK major versions 2
Maintenance and support for SDK major versions 2
Getting Started 3
Get an Amazon Account 3
Install the AWS SDK for Go 3
Get your AWS access keys 3
To get your access key ID and secret access key 4
Import Packages 4
Configuring the SDK 6
Creating a Session 6
Specifying the AWS Region 7
Specifying Credentials 8
IAM Roles for Tasks 9
IAM Roles for Amazon EC2 Instances 9
Shared Credentials File 9
Environment Variables 11
Hard-Coded Credentials in an Application (Not Recommended) 12
Other Credentials Providers 12
Configuring a Proxy 12
Logging Service Calls 12
Creating a Custom Endpoint 13
Custom HTTP Client 13
Dialer.KeepAlive 13
Dialer.Timeout 14
Transport.ExpectContinueTimeout 14
Transport.IdleConnTimeout 14
Transport.MaxIdleConns 14
Transport.MaxIdleConnsPerHost 15
Transport.ResponseHeaderTimeout 15
Transport.TLSHandshakeTimeout 15
Create Import Statement 16

Creating a Timeout Struct ... 16
Creating a Function to Create a Custom HTTP Client ... 17
Using a Custom HTTP Client ... 17
Using Sessions ... 20
Concurrency ... 20
Sessions with a Shared Configuration File ... 20
Creating Sessions ... 20
Create Sessions with Option Overrides ... 21
Deprecated New ... 22
Shared Configuration Fields ... 22
Environment Variables ... 11
Adding Request Handlers ... 23
Copying a Session ... 23
Using AWS Services ... 24
Constructing a Service ... 24
Tagging Service Resources ... 25
Getting the HTTP Request and Response with Each Service Call ... 27
Service Operation Calls ... 29
Calling Operations ... 29
Calling Operations with the Request Form ... 30
Handling Operation Response Body ... 30
Concurrently Using Service Clients ... 31
Using Pagination Methods ... 32
Using Waiters ... 33
Handling Errors ... 35
Handling Specific Service Error Codes ... 35
Additional Error Information ... 36
Specific Error Interfaces ... 36
Code Examples ... 38
SDK Request Examples ... 39
Using context.Context with SDK Requests ... 39
Using API Field Setters with SDK Requests ... 39
AWS CloudTrail Examples ... 40
Listing the CloudTrail Trails ... 40
Creating a CloudTrail Trail ... 42
Listing CloudTrail Trail Events ... 45

Deleting a CloudTrail Trail ... 47
Amazon CloudWatch Examples ... 48
Describing CloudWatch Alarms ... 49
Using Alarms and Alarm Actions in CloudWatch ... 50
Getting Metrics from CloudWatch ... 54
Sending Events to Amazon CloudWatch Events ... 58
Getting Log Events from CloudWatch ... 63
AWS CodeBuild Examples ... 65
Getting Information about All AWS CodeBuild Projects ... 66
Building an AWS CodeBuild Project ... 67
Listing Your AWS CodeBuild Project Builds ... 68
Amazon DynamoDB Examples ... 69
Listing all Amazon DynamoDB Tables ... 70
Creating an Amazon DynamoDB Table ... 71
Creating an Amazon DynamoDB Table Item ... 73
Creating Amazon DynamoDB Table Items from a JSON File ... 75
Reading an Amazon DynamoDB Table Item ... 77
Getting Amazon DynamoDB Table Items Using Expression Builder ... 79
Updating an Amazon DynamoDB Table Item ... 82
Deleting an Amazon DynamoDB Table Item ... 84
Amazon EC2 Examples ... 85
Creating Amazon EC2 Instances with Tags or without Block Devices ... 86
Managing Amazon EC2 Instances ... 89
Working with Amazon EC2 Key Pairs ... 97
Using Regions and Availability Zones with Amazon EC2 ... 101
Working with Security Groups in Amazon EC2 ... 104
Using Elastic IP Addresses in Amazon EC2 ... 111
Amazon Glacier Examples ... 117
The Scenario ... 117
Prerequisites ... 117
Create a Vault ... 118
Upload an Archive ... 119
IAM Examples ... 120
Managing IAM Users ... 120
Managing IAM Access Keys ... 130
Managing IAM Account Aliases ... 136

Working with IAM Policies ... 140
Working with IAM Server Certificates ... 148
AWS KMS Examples ... 153
Creating a CMK in AWS Key Management Service ... 154
Encrypting Data with AWS Key Management Service ... 155
Decrypting a Data Blob in AWS Key Management Service ... 156
Re-encrypting a Data Blob in AWS Key Management Service ... 157
AWS Lambda Examples ... 158
Displaying Information about All Lambda Functions ... 159
Creating a Lambda Function ... 160
Running a Lambda Function ... 161
Configuring a Lambda Function to Receive Notifications ... 165
Amazon Polly Examples ... 166
Getting a List of Voices ... 166
Getting a List of Lexicons ... 167
Synthesizing Speech ... 168
Amazon S3 Examples ... 170
Performing Basic Amazon S3 Bucket Operations ... 171
Creating Pre-Signed URLs for Amazon S3 Buckets ... 188
Using an Amazon S3 Bucket as a Static Web Host ... 192
Working with Amazon S3 CORS Permissions ... 197
Working with Amazon S3 Bucket Policies ... 200
Working with Amazon S3 Bucket ACLs ... 204
Encrypting Amazon S3 Bucket Items ... 215
Amazon SES Examples ... 221
Listing Valid Amazon SES Email Addresses ... 221
Verifying an Email Address in Amazon SES ... 222
Sending a Message to an Email Address in Amazon SES ... 224
Deleting an Email Address in Amazon SES ... 227
Getting Amazon SES Statistics ... 228
Amazon SNS Examples ... 229
Listing Your Amazon SNS Topics ... 230
Creating an Amazon SNS Topic ... 230
List Your Amazon SNS Subscriptions ... 231
Subscribe to an Amazon SNS Topic ... 233
Sending a Message to All Amazon SNS Topic Subscribers ... 234

Amazon SQS Examples ... 235
Using Amazon SQS Queues ... 236
Sending and Receiving Messages in Amazon SQS ... 241
Managing Visibility Timeout in Amazon SQS Queues ... 246
Enabling Long Polling in Amazon SQS Queues ... 248
Using Dead Letter Queues in Amazon SQS ... 254
Amazon WorkDocs Examples ... 256
Listing Users ... 257
Listing User Docs ... 259
SDK Utilities ... 262
Amazon CloudFront URL Signer ... 262
Amazon DynamoDB Attributes Converter ... 262
Amazon Elastic Compute Cloud Metadata ... 263
Retrieving an Instance's Region ... 264
Amazon S3 Transfer Managers ... 264
Upload Manager ... 264
Download Manager ... 270
Security ... 273
Data Protection ... 273
Identity and Access Management ... 274
Audience ... 275
Authenticating with identities ... 275
Managing access using policies ... 279
How AWS services work with IAM ... 281
Troubleshooting AWS identity and access ... 281
Compliance Validation ... 283
Resilience ... 284
Infrastructure Security ... 285
Enforcing a minimum TLS version ... 285
How do I set my TLS version? ... 286
S3 Encryption Client Migration ... 287
Migration Overview ... 287
Update Existing Clients to Read New Formats ... 288
Migrate Encryption and Decryption Clients to V2 ... 289
Document History ... 296

We announced the upcoming end-of-support for AWS SDK for Go V1. We recommend that you migrate to AWS SDK for Go V2. For dates, additional details, and information on how to migrate, please refer to the linked announcement.

What is the AWS SDK for Go

The AWS SDK for Go provides APIs and utilities that developers can use to build Go applications that use AWS services, such as Amazon Elastic Compute Cloud (Amazon EC2) and Amazon Simple Storage Service (Amazon S3).

> **Note**
>
> This document is for version 1 of the AWS SDK for Go. If you're looking for version 2 of the SDK, see the version 2 developer guide and the version 2 SDK API reference.

The SDK removes the complexity of coding directly against a web service interface. It hides a lot of the lower-level plumbing, such as authentication, request retries, and error handling.

The SDK also includes helpful utilities. For example, the Amazon S3 download and upload manager can automatically break up large objects into multiple parts and transfer them in parallel.

Use the AWS SDK for Go Developer Guide to help you install, configure, and use the SDK. The guide provides configuration information, sample code, and an introduction to the SDK utilities.

More Info

- To learn about everything you need before you can start using the AWS SDK for Go, see Getting Started with the AWS SDK for Go.
- For code examples, see AWS SDK for Go Code Examples.
- You can browse the AWS SDK for Go examples in the aws-doc-sdk-examples repo on GitHub.
- To learn about the SDK utilities, see Using the AWS SDK for Go Utilities.
- For learn about the types and functionality that the library provides, see the AWS SDK for Go API Reference.
- To view a video introduction of the SDK and a sample application demonstration, see AWS SDK for Go: Gophers Get Going with AWS from AWS re:Invent 2015.

Maintenance and support for SDK major versions

Maintenance and support for SDK major versions

For information about maintenance and support for SDK major versions and their underlying dependencies, see the following in the AWS SDKs and Tools Reference Guide:

- AWS SDKs and tools maintenance policy
- AWS SDKs and tools version support matrix

Getting Started with the AWS SDK for Go

The AWS SDK for Go requires Go 1.5 or later. You can view your current version of Go by running the `go version` command. For information about installing or upgrading your version of Go, see https://golang.org/doc/install.

Get an Amazon Account

Before you can use the AWS SDK for Go, you must have an Amazon account. See How do I create and activate a new Amazon Web Services account? for details.

Install the AWS SDK for Go

To install the SDK and its dependencies, run the following Go command.

```
go get -u github.com/aws/aws-sdk-go/...
```

If you set the Go vendor experiment environment variable to 1, you can use the following command to get the SDK. The SDK's runtime dependencies are vendored in the `vendor/` folder.

```
go get -u github.com/aws/aws-sdk-go
```

Get your AWS access keys

Access keys consist of an *access key ID* and *secret access key*, which are used to sign programmatic requests that you make to AWS. If you don't have access keys, you can create them by using the AWS Management Console. We recommend that you use IAM access keys instead of AWS root account access keys. IAM lets you securely control access to AWS services and resources in your AWS account.

> **Note**
>
> To create access keys, you must have permissions to perform the required IAM actions. For more information, see Granting IAM User Permission to Manage Password Policy and Credentials in the IAM User Guide.

To get your access key ID and secret access key

1. Open the IAM console.
2. On the navigation menu, choose **Users**.
3. Choose your IAM user name (not the check box).
4. Open the **Security credentials** tab, and then choose **Create access key**.
5. To see the new access key, choose **Show**. Your credentials resemble the following:
   - Access key ID: `AKIAIOSFODNN7EXAMPLE`
   - Secret access key: `wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY`
6. To download the key pair, choose **Download .csv file**. Store the keys

in a secure location.

> **Important**
>
> Keep the keys confidential to protect your AWS account, and never email them. Do not share them outside your organization, even if an inquiry appears to come from AWS or Amazon.com. *No one who legitimately represents Amazon will ever ask you for your secret key.*

Related topics

- What Is IAM? in IAM User Guide.
- AWS Security Credentials in Amazon Web Services General Reference.

Import Packages

After you have installed the SDK, you import AWS packages into your Go applications to use the SDK, as shown in the following example, which imports the AWS, Session, and Amazon S3 libraries:

```
import (
    "github.com/aws/aws-sdk-go/aws"
    "github.com/aws/aws-sdk-go/aws/session"
    "github.com/aws/aws-sdk-go/service/s3"
```

```
)
```

Configuring the AWS SDK for Go

In the AWS SDK for Go, you can configure settings for service clients, such as the log level and maximum number of retries. Most settings are optional. However, for each service client, you must specify an AWS Region and your credentials. The SDK uses these values to send requests to the correct Region and sign requests with the correct credentials. You can specify these values as part of a session or as environment variables.

Creating a Session

Before you can create a service client you must create a session, which is part of the `github.com/aws/aws-sdk-go/aws/session` package.

There are a number of ways of configuring a session but the following are the most common.

Create a session using the default Region and credentials:

```
import (
    "github.com/aws/aws-sdk-go/aws"
    "github.com/aws/aws-sdk-go/aws/session"
)

// ...

sess := session.Must(session.NewSessionWithOptions(session.Options{
    SharedConfigState: session.SharedConfigEnable,
}))
```

Create a session in **us-west-2**:

```
import (
    "github.com/aws/aws-sdk-go/aws"
    "github.com/aws/aws-sdk-go/aws/session"
)

// ...

sess, err := session.NewSession(&aws.Config{
    Region: aws.String("us-west-2")},
```

```
)
```

See session for additional information.

Specifying the AWS Region

When you specify the Region, you specify where to send requests, such as `us-west-2` or `us-east-2`. For a list of Regions for each service, see Service endpoints and quotas in the Amazon Web Services General Reference.

The SDK does not have a default Region. To specify a Region:

- Set the `AWS_REGION` environment variable to the default Region
- Set the `AWS_SDK_LOAD_CONFIG` environment variable to **true** to get the Region value from the *config* file in the `.aws/` folder in your home directory
- Set the **NewSessionWithOptions** method argument **SharedConfigState** to **SharedConfigEnable** when you create a session to get the Region value from the *config* file in the `.aws/` folder in your home directory
- Set the Region explicitly when you create a session

If you set a Region using all of these techniques, the SDK uses the Region you explicitly specified in the session.

The following examples show you how to configure the environment variable.

Linux, OS X, or Unix

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

Windows

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

The following snippet specifies the Region in a session:

```
sess, err := session.NewSession(&aws.Config{Region: aws.String("us-west-2")})
```

Specifying Credentials

The AWS SDK for Go requires credentials (an access key and secret access key) to sign requests to AWS. You can specify your credentials in several different locations, depending on your particular use case. For information about obtaining credentials, see Setting Up.

When you initialize a new service client without providing any credential arguments, the SDK uses the default credential provider chain to find AWS credentials. The SDK uses the first provider in the chain that returns credentials without an error. The default provider chain looks for credentials in the following order:

1. Environment variables.
2. Shared credentials file.
3. If your application uses an ECS task definition or RunTask API operation, IAM role for tasks.
4. If your application is running on an Amazon EC2 instance, IAM role for Amazon EC2.

The SDK detects and uses the built-in providers automatically, without requiring manual configurations. For example, if you use IAM roles for Amazon EC2 instances, your applications automatically use the instance's credentials. You don't need to manually configure credentials in your application.

As a best practice, AWS recommends that you specify credentials in the following order:

1. Use IAM roles for tasks if your application uses an ECS task definition or RunTask API operation.
2. Use IAM roles for Amazon EC2 (if your application is running on an Amazon EC2 instance).

   IAM roles provide applications on the instance temporary security credentials to make AWS calls. IAM roles provide an easy way to distribute and manage credentials on multiple Amazon EC2 instances.
3. Use a shared credentials file.

   This credentials file is the same one used by other SDKs and the AWS CLI. If you're already using a shared credentials file, you can also use it for this purpose.
4. Use environment variables.

   Setting environment variables is useful if you're doing development work on a machine other than an Amazon EC2 instance.

IAM Roles for Tasks

If your application uses an Amazon ECS task definition or `RunTask` operation, use IAM Roles for Tasks to specify an IAM role that can be used by the containers in a task.

IAM Roles for Amazon EC2 Instances

If you are running your application on an Amazon EC2 instance, use the instance's IAM role to get temporary security credentials to make calls to AWS.

If you have configured your instance to use IAM roles, the SDK uses these credentials for your application automatically. You don't need to manually specify these credentials.

Shared Credentials File

A credential file is a plaintext file that contains your access keys. The file must be on the same machine on which you're running your application. The file must be named `credentials` and located in the `.aws/` folder in your home directory. The home directory can vary by operating system. In Windows, you can refer to your home directory by using the environment variable `%UserProfile%`. In Unix-like systems, you can use the environment variable `$HOME` or ~ (tilde).

If you already use this file for other SDKs and tools (like the AWS CLI), you don't need to change anything to use the files in this SDK. If you use different credentials for different tools or applications, you can use *profiles* to configure multiple access keys in the same configuration file.

Creating the Credentials File

If you don't have a shared credentials file (`.aws/credentials`), you can use any text editor to create one in your home directory. Add the following content to your credentials file, replacing `<YOUR_ACCESS_KEY_ID>` and `<YOUR_SECRET_ACCESS_KEY>` with your credentials.

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

The `[default]` heading defines credentials for the default profile, which the SDK will use unless you configure it to use another profile.

You can also use temporary security credentials by adding the session tokens to your profile, as shown in the following example:

```
[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>
```

Specifying Profiles

You can include multiple access keys in the same configuration file by associating each set of access keys with a profile. For example, in your credentials file, you can declare multiple profiles, as follows.

```
[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>
```

By default, the SDK checks the `AWS_PROFILE` environment variable to determine which profile to use. If no `AWS_PROFILE` variable is set, the SDK uses the default profile.

If you have an application named `myapp` that uses the SDK, you can run it with the test credentials by setting the variable to `test-account myapp`, as shown in the following command.

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

You can also use the SDK to select a profile by specifying `os.Setenv("AWS_PROFILE", test-account)` before constructing any service clients or by manually setting the credential provider, as shown in the following example.

```
sess, err := session.NewSession(&aws.Config{
    Region:      aws.String("us-west-2"),
    Credentials: credentials.NewSharedCredentials("", "test-account"),
})
```

In addition, checking if your credentials have been found is fairly easy.

```
_, err := sess.Config.Credentials.Get()
```

If `ChainProvider` is being used, set `CredentialsChainVerboseErrors` to `true` in the session config.

> **Note**
>
> If you specify credentials in environment variables, the SDK will always use those credentials, no matter which profile you specify.

Environment Variables

By default, the SDK detects AWS credentials set in your environment and uses them to sign requests to AWS. That way you don't need to manage credentials in your applications.

The SDK looks for credentials in the following environment variables:

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

The following examples show how you configure the environment variables.

Linux, OS X, or 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
```

Hard-Coded Credentials in an Application (Not Recommended)

> **Warning**
>
> Do not embed credentials inside an application. Use this method only for testing purposes.

You can hard-code credentials in your application by passing the access keys to a configuration instance, as shown in the following snippet.

```
sess, err := session.NewSession(&aws.Config{
    Region:      aws.String("us-west-2"),
    Credentials: credentials.NewStaticCredentials("AKID", "SECRET_KEY", "TOKEN"),
})
```

Other Credentials Providers

The SDK provides other methods for retrieving credentials in the `aws/credentials` package. For example, you can retrieve temporary security credentials from AWS Security Token Service or credentials from encrypted storage. For more information, see Credentials.

Configuring a Proxy

If you cannot directly connect to the internet, you can use Go-supported environment variables (HTTP_PROXY) or create a custom HTTP client to configure your proxy. Use the Config.HTTPClient struct to specify a custom HTTP client. For more information about how to create an HTTP client to use a proxy, see the Transport struct in the Go `http` package.

Logging Service Calls

You can enable logging in a client by setting the `LogLevel` in a configuration instance, as shown in the following snippet, which sets the log level to `LogDebugWithHTTPBody` for a new DynamoDB client.

```
svc := dynamodb.New(sess, aws.NewConfig().WithLogLevel(aws.LogDebugWithHTTPBody))
```

See LogLevelType for the different log level values.

Creating a Custom Endpoint

In most cases you use the endpoint that is pre-configured for a service. However, you can specify a custom endpoint, such as for pre-release versions of the service, as shown in the following snippet, which sets the `Endpoint` to `https://test.us-west-2.amazonaws.com` for a new DynamoDB client.

```
svc := dynamodb.New(sess, &aws.Config{Endpoint: aws.String("https://test.us-
west-2.amazonaws.com")})
```

See aws.Config for details.

Creating a Custom HTTP Client

The AWS SDK for Go uses a default HTTP client with default configuration values. Although you can change some of these configuration values, the default HTTP client and transport are not sufficiently configurable for customers using the AWS SDK for Go in an environment with high throughput and low latency requirements. This section describes how to create a custom HTTP client, and use that client to create AWS SDK for Go calls.

To assist you in creating a custom HTTP client, this section describes how to create a structure to encapsulate the custom settings, create a function to create a custom HTTP client based on those settings, and use that custom HTTP client to call an AWS SDK for Go service client.

Let's define what we want to customize.

Dialer.KeepAlive

This setting represents the keep-alive period for an active network connection.

Set to a negative value to disable keep-alives.

Set to **0** to enable keep-alives if supported by the protocol and operating system.

Network protocols or operating systems that do not support keep-alives ignore this field. By default, TCP enables keep alive.

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

We'll call this `ConnKeepAlive` as **time.Duration**.

Dialer.Timeout

This setting represents the maximum amount of time a dial to wait for a connection to be created.

Default is 30 seconds.

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

We'll call this `Connect` as **time.Duration**.

Transport.ExpectContinueTimeout

This setting represents the maximum amount of time to wait for a server's first response headers after fully writing the request headers, if the request has an "Expect: 100-continue" header. This time does not include the time to send the request header. The HTTP client sends its payload after this timeout is exhausted.

Default 1 second.

Set to **0** for no timeout and send request payload without waiting. One use case is when you run into issues with proxies or third party services that take a session similar to the use of Amazon S3 in the function shown later.

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

We'll call this `ExpectContinue` as **time.Duration**.

Transport.IdleConnTimeout

This setting represents the maximum amount of time to keep an idle network connection alive between HTTP requests.

Set to **0** for no limit.

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

We'll call this `IdleConn` as **time.Duration**.

Transport.MaxIdleConns

This setting represents the maximum number of idle (keep-alive) connections across all hosts. One use case for increasing this value is when you are seeing many connections in a short period from the same clients

0 means no limit.

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

We'll call this `MaxAllIdleConns` as **int**.

Transport.MaxIdleConnsPerHost

This setting represents the maximum number of idle (keep-alive) connections to keep per-host. One use case for increasing this value is when you are seeing many connections in a short period from the same clients

Default is two idle connections per host.

Set to **0** to use DefaultMaxIdleConnsPerHost (2).

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

We'll call this `MaxHostIdleConns` as **int**.

Transport.ResponseHeaderTimeout

This setting represents the maximum amount of time to wait for a client to read the response header.

If the client isn't able to read the response's header within this duration, the request fails with a timeout error.

Be careful setting this value when using long-running Lambda functions, as the operation does not return any response headers until the Lambda function has finished or timed out. However, you can still use this option with the **InvokeAsync** API operation.

Default is no timeout; wait forever.

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

We'll call this `ResponseHeader` as **time.Duration**.

Transport.TLSHandshakeTimeout

This setting represents the maximum amount of time waiting for a TLS handshake to be completed.

Default is 10 seconds.

Zero means no timeout.

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

We'll call this TLSHandshake as **time.Duration**.

Create Import Statement

The complete example imports the following Go packages.

```
import (
    "bytes"
    "context"
    "flag"
    "fmt"
    "io"
    "net"
    "net/http"
    "time"

    "github.com/aws/aws-sdk-go/aws"
    "github.com/aws/aws-sdk-go/aws/session"
    "github.com/aws/aws-sdk-go/service/s3"

    "golang.org/x/net/http2"
)
```

Creating a Timeout Struct

Let's create a struct to hold the timeout values we want to be able to set on our HTTP client.

```
type HTTPClientSettings struct {
    Connect          time.Duration
    ConnKeepAlive    time.Duration
    ExpectContinue   time.Duration
    IdleConn         time.Duration
    MaxAllIdleConns  int
    MaxHostIdleConns int
    ResponseHeader   time.Duration
    TLSHandshake     time.Duration
```

```
}
```

Creating a Function to Create a Custom HTTP Client

Next let's create a function that takes a **ClientTimeout** struct and creates a custom HTTP client based on those timeout values.

```
func NewHTTPClientWithSettings(httpSettings HTTPClientSettings) (*http.Client, error) {
    var client http.Client
    tr := &http.Transport{
        ResponseHeaderTimeout: httpSettings.ResponseHeader,
        Proxy:                 http.ProxyFromEnvironment,
        DialContext: (&net.Dialer{
            KeepAlive: httpSettings.ConnKeepAlive,
            DualStack: true,
            Timeout:   httpSettings.Connect,
        }).DialContext,
        MaxIdleConns:          httpSettings.MaxAllIdleConns,
        IdleConnTimeout:       httpSettings.IdleConn,
        TLSHandshakeTimeout:   httpSettings.TLSHandshake,
        MaxIdleConnsPerHost:   httpSettings.MaxHostIdleConns,
        ExpectContinueTimeout: httpSettings.ExpectContinue,
    }

    // So client makes HTTP/2 requests
    err := http2.ConfigureTransport(tr)
    if err != nil {
        return &client, err
    }

    return &http.Client{
        Transport: tr,
    }, nil
}
```

Using a Custom HTTP Client

Let's create a custom HTTP client and use it to create an Amazon S3 client.

The following example creates an **http.Client** that is configured to have:

- a five second TCP connection timeout
- a five second TLS handshake timeout
- a five second wait for the HTTP response headers

```
httpClient, err := NewHTTPClientWithSettings(HTTPClientSettings{
    Connect:          5 * time.Second,
    ExpectContinue:   1 * time.Second,
    IdleConn:         90 * time.Second,
    ConnKeepAlive:    30 * time.Second,
    MaxAllIdleConns:  100,
    MaxHostIdleConns: 10,
    ResponseHeader:   5 * time.Second,
    TLSHandshake:     5 * time.Second,
})
if err != nil {
    fmt.Println("Got an error creating custom HTTP client:")
    fmt.Println(err)
    return
}

sess := session.Must(session.NewSession(&aws.Config{
    HTTPClient: httpClient,
}))

svc := s3.New(sess)
```

All of these settings give the client approximately 15 seconds create a connection, do a TLS handshake, and receive the response headers from the service. The time that the client takes to read the response body is not covered by these timeouts. To specify a total timeout for the request to include reading the response body, use the AWS SDK for Go client's **WithContext** API operation methods, such as the Amazon S3 operation PutObjectWithContext with a **context.Withtimeout**.

The following example uses a timeout context to limit the total time an API request can be active to a maximum of 20 seconds. The SDK must be able to read the full HTTP response body (Object body) within the timeout or the SDK returns a timeout error. For API operations that return an **io.ReadCloser** in their response type, the Context's timeout includes reading the content from the **io.ReadCloser**.

```
ctx, cancelFn := context.WithTimeout(context.TODO(), 20*time.Second)
defer cancelFn()
```

```
resp, err := svc.GetObjectWithContext(ctx, &s3.GetObjectInput{
    Bucket: bucket,
    Key:    object,
})
if err != nil {
    return body, err
}

return resp.Body, nil
```

See the complete example on GitHub.

Using Sessions to Configure Service Clients in the AWS SDK for Go

In the AWS SDK for Go, a session is an object that contains configuration information for service clients. which you use to interact with AWS services. For example, sessions can include information about the region where requests will be sent, which credentials to use, or additional request handlers. Whenever you create a service client, you must specify a session. For more information about sessions, see the session package in the AWS SDK for Go API Reference.

Sessions can be shared across all service clients that share the same base configuration. The session is built from the SDK's default configuration and request handlers.

You should cache sessions when possible. This is because creating a new session loads all configuration values from the environment and configuration files each time the session is created. Sharing the session value across all of your service clients ensures the configuration is loaded the fewest number of times.

Concurrency

Sessions are safe to use concurrently as long as the session isn't being modified. The SDK doesn't modify the session once the session is created. Creating service clients concurrently from a shared session is safe.

Sessions with a Shared Configuration File

Using the previous method, you can create sessions that load the additional configuration file only if the `AWS_SDK_LOAD_CONFIG` environment variable is set. Alternatively you can explicitly create a session with a shared configuration enabled. To do this, you can use `NewSessionWithOptions` to configure how the session is created. Using the `NewSessionWithOptions` with `SharedConfigState` set to `SharedConfigEnable` will create the session as if the `AWS_SDK_LOAD_CONFIG` environment variable was set.

Creating Sessions

When you create a `session`, you can pass in optional `aws.Config` values that override the default or that override the current configuration values. This allows you to provide additional or case-based configuration as needed.

By default `NewSession` only loads credentials from the shared credentials file (`~/.aws/credentials`). If the `AWS_SDK_LOAD_CONFIG` environment variable is set to a truthy value, the session is created from the configuration values from the shared configuration (`~/.aws/config`) and shared credentials (`~/.aws/credentials`) files. See Sessions with a Shared Configuration File for more information.

Create a session with the default configuration and request handlers. The following example creates a session with credentials, region, and profile values from either the environment variables or the shared credentials file. It requires that the `AWS_PROFILE` is set, or `default` is used.

```
sess, err := session.NewSession()
```

The SDK provides a default configuration that all sessions use, unless you override a field. For example, you can specify an AWS Region when you create a session by using the `aws.Config` struct. For more information about the fields you can specify, see the aws.Config in the AWS SDK for Go API Reference.

```
sess, err := session.NewSession(&aws.Config{
    Region: aws.String("us-east-2")},
)
```

Create an Amazon S3 client instance from a session:

```
sess, err := session.NewSession()
if err != nil {
    // Handle Session creation error
}
svc := s3.New(sess)
```

Create Sessions with Option Overrides

In addition to `NewSession`, you can create sessions using `NewSessionWithOptions`. This function allows you to control and override how the session will be created through code, instead of being driven by environment variables only.

Use NewSessionWithOptions when you want to provide the config profile, or override the shared credentials state (`AWS_SDK_LOAD_CONFIG`).

```
// Equivalent to session.New
```

```
sess, err := session.NewSessionWithOptions(session.Options{})

// Specify profile to load for the session's config
sess, err := session.NewSessionWithOptions(session.Options{
    Profile: "profile_name",
})

// Specify profile for config and region for requests
sess, err := session.NewSessionWithOptions(session.Options{
    Config: aws.Config{Region: aws.String("us-east-2")},
    Profile: "profile_name",
})

// Force enable Shared Config support
sess, err := session.NewSessionWithOptions(session.Options{
    SharedConfigState: SharedConfigEnable,
})

// Assume an IAM role with MFA prompting for token code on stdin
sess := session.Must(session.NewSessionWithOptions(session.Options{
    AssumeRoleTokenProvider: stscreds.StdinTokenProvider,
    SharedConfigState: SharedConfigEnable,
}))
```

Deprecated New

The `New` function has been deprecated because it doesn't provide a good way to return errors that occur when loading the configuration files and values. Because of this, `NewSession` was created so errors can be retrieved when creating a session fails.

Shared Configuration Fields

By default, the SDK loads credentials from the shared credentials file `~/.aws/credentials`. Any other configuration values are provided by the environment variables, SDK defaults, and user-provided `aws.config` values.

If the `AWS_SDK_LOAD_CONFIG` environment variable is set, or the **SharedConfigEnable** option is used to create the session (as shown in the following example), additional configuration information is also loaded from the shared configuration file (`~/.aws/config`), if it exists. If any configuration setting value differs between the two files, the value from the shared credentials file (`~/.aws/credentials`) takes precedence.

```
sess := session.Must(session.NewSessionWithOptions(session.Options{
    SharedConfigState: session.SharedConfigEnable,
}))
```

See the session package's documentation for more information on shared credentials setup.

Environment Variables

When a session is created, you can set several environment variables to adjust how the SDK functions, and what configuration data it loads when creating sessions. Environment values are optional. For credentials, you must set both an access key and a secret access key. Otherwise, Go ignores the one you've set. All environment variable values are strings unless otherwise noted.

See the session package's documentation for more information on environment variable setup.

Adding Request Handlers

You can add handlers to a session for processing HTTP requests. All service clients that use the session inherit the handlers. For example, the following handler logs every request and its payload made by a service client.

```
// Create a session, and add additional handlers for all service
// clients created with the Session to inherit. Adds logging handler.
sess, err := session.NewSession()
sess.Handlers.Send.PushFront(func(r *request.Request) {
    // Log every request made and its payload
    logger.Println("Request: %s/%s, Payload: %s",
        r.ClientInfo.ServiceName, r.Operation, r.Params)
})
```

Copying a Session

You can use the Copy method to create copies of sessions. Copying sessions is useful when you want to create multiple sessions that have similar settings. Each time you copy a session, you can specify different values for any field. For example, the following snippet copies the `sess` session while overriding the `Region` field to `us-east-2`:

```
usEast2Sess := sess.Copy(&aws.Config{Region: aws.String("us-east-2")})
```

Using the AWS SDK for Go with AWS Services

To make calls to an AWS service, you must first construct a service client instance with a session. A service client provides low-level access to every API action for that service. For example, you create an Amazon S3 service client to make calls to Amazon S3.

When you call service operations, you pass in input parameters as a struct. A successful call usually results in an output struct that you can use. For example, after you successfully call an Amazon S3 create bucket action, the action returns an output struct with the bucket's location.

For the list of service clients, including their methods and parameters, see the AWS SDK for Go API Reference.

Constructing a Service

To construct a service client instance, use the NewSession() function. The following example creates an Amazon S3 service client.

```
sess, err := session.NewSession()
if err != nil {
    fmt.Println("Error creating session ", err)
    return
}
svc := s3.New(sess)
```

After you have a service client instance, you can use it to call service operations. For more information about configurations, see Configuring the AWS SDK for Go.

When you create a service client, you can pass in custom configurations so that you don't need to create a session for each configuration. The SDK merges the two configurations, overriding session values with your custom configuration. For example, in the following snippet, the Amazon S3 client uses the `mySession` session but overrides the `Region` field with a custom value (`us-west-2`):

```
svc := s3.New(mySession, aws.NewConfig().WithRegion("us-west-2"))
```

Tagging Service Resources

You can tag service resources, such as Amazon S3 buckets, so that you can determine the costs of your service resources at whatever level of granularity you require.

The following example shows how to tag the Amazon S3 bucket `amzn-s3-demo-bucket` with `Cost Center` tag with the value `123456` and `Stack` tag with the value `MyTestStack`.

```
package main

import (
    "github.com/aws/aws-sdk-go/aws"
    "github.com/aws/aws-sdk-go/aws/session"
    "github.com/aws/aws-sdk-go/service/s3"

    "fmt"
)

// Tag S3 bucket amzn-s3-demo-bucket with cost center tag "123456" and stack tag
 "MyTestStack".
//
// See:
//    http://docs.aws.amazon.com/awsaccountbilling/latest/aboutv2/cost-alloc-tags.html
func main() {
    // Pre-defined values
    bucket := "amzn-s3-demo-bucket"
    tagName1 := "Cost Center"
    tagValue1 := "123456"
    tagName2 := "Stack"
    tagValue2 := "MyTestStack"

    // Initialize a session in us-west-2 that the SDK will use to load credentials
    // from the shared credentials file. (~/.aws/credentials).
    sess, err := session.NewSession(&aws.Config{
        Region: aws.String("us-west-2")},
    )
    if err != nil {
        fmt.Println(err.Error())
        return
    }

    // Create S3 service client
    svc := s3.New(sess)
```

```
    // Create input for PutBucket method
    putInput := &s3.PutBucketTaggingInput{
        Bucket: aws.String(bucket),
        Tagging: &s3.Tagging{
            TagSet: []*s3.Tag{
                {
                    Key:   aws.String(tagName1),
                    Value: aws.String(tagValue1),
                },
                {
                    Key:   aws.String(tagName2),
                    Value: aws.String(tagValue2),
                },
            },
        },
    }

    _, err = svc.PutBucketTagging(putInput)
    if err != nil {
        fmt.Println(err.Error())
        return
    }

    // Now show the tags
    // Create input for GetBucket method
    getInput := &s3.GetBucketTaggingInput{
        Bucket: aws.String(bucket),
    }

    result, err := svc.GetBucketTagging(getInput)
    if err != nil {
        fmt.Println(err.Error())
        return
    }

    numTags := len(result.TagSet)

    if numTags > 0 {
        fmt.Println("Found", numTags, "Tag(s):")
        fmt.Println("")

        for _, t := range result.TagSet {
            fmt.Println("  Key:  ", *t.Key)
```

```
            fmt.Println("  Value:", *t.Value)
            fmt.Println("")
        }
    } else {
        fmt.Println("Did not find any tags")
    }
}
```

Note that if a tag of the same name already exists, its value is overwritten by the new value.

Getting the HTTP Request and Response with Each Service Call

You can direct the AWS SDK for Go to display the HTTP request and response it sends and receives for each call by including a configuration option when constructing the service client.

The following example uses the DynamoDB **ListTables** operation to illustrate how to add a custom header to a service call.

```
package main

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/dynamodb"

    "fmt"
    "os"
)

func main() {
    // Initialize a session in us-west-2 that the SDK will use to load credentials
    // from the shared config file. (~/.aws/credentials).
    sess, err := session.NewSession(&aws.Config{
        Region: aws.String("us-west-2")},
    )
    if err != nil {
        fmt.Println("Error getting session:")
        fmt.Println(err)
        os.Exit(1)
    }
```

```
    // Create DynamoDB client
    // and expose HTTP requests/responses
    svc := dynamodb.New(sess, aws.NewConfig().WithLogLevel(aws.LogDebugWithHTTPBody))

    // Add "CustomHeader" header with value of 10
    svc.Handlers.Send.PushFront(func(r *request.Request) {
        r.HTTPRequest.Header.Set("CustomHeader", fmt.Sprintf("%d", 10))
    })

    // Call ListTables just to see HTTP request/response
    // The request should have the CustomHeader set to 10
    _, _ = svc.ListTables(&dynamodb.ListTablesInput{})
}
```

If you run this program, the output should be similar to the following, where **ACCESS-KEY** is the access key of the user and **TABLE-1**, through **TABLE-N** are the names of the tables.

```
2017/10/25 11:10:57 DEBUG: Request dynamodb/ListTables Details:
---[ REQUEST POST-SIGN ]-----------------------------
POST / HTTP/1.1
Host: dynamodb.us-west-2.amazonaws.com
User-Agent: aws-sdk-go/1.10.34 (go1.8; windows; amd64)
Content-Length: 2
Accept-Encoding: identity
Authorization: AWS4-HMAC-SHA256 Credential=ACCESS-KEY/20171025/us-west-2/dynamodb/
aws4_request, SignedHeaders=accept-encoding;content-length;content-type;host;x-amz-
date;x-amz-target,
 Signature=9c92efe5d6c597cf29e4f7cc74de6dc2e39f8010a0d4957a397c59ef9cde21f2
Content-Type: application/x-amz-json-1.0
CustomHeader: 10
X-Amz-Date: 20171025T181057Z
X-Amz-Target: DynamoDB_20120810.ListTables

{}
-----------------------------------------------------
2017/10/25 11:10:58 DEBUG: Response dynamodb/ListTables Details:
---[ RESPONSE ]--------------------------------------
HTTP/1.1 200 OK
Content-Length: 177
Connection: keep-alive
Content-Type: application/x-amz-json-1.0
Date: Wed, 25 Oct 2017 18:10:58 GMT
Server: Server
```

```
X-Amz-Crc32: 3023160996
X-Amzn-Requestid: M5B4BM4UU569MVBSDG5O2O9ITJVV4KQNSO5AEMVJF66Q9ASUAAJG


-----------------------------------------------------
2017/10/25 11:10:58 {"TableNames":["TABLE-1","...","TABLE-N"]}
```

Service Operation Calls

You can call a service operation directly or with its request form. When you call a service operation, the SDK synchronously validates the input, builds the request, signs it with your credentials, sends it to AWS, and then gets a response or an error. In most cases, you can call service operations directly.

Calling Operations

Calling the operation will sync as the request is built, signed, sent, and the response is received. If an error occurs during the operation, it will be returned. The output or resulting structure won't be valid.

For example, to call the Amazon S3 GET Object API, use the Amazon S3 service client instance and call its GetObject method:

```
result, err := s3Svc.GetObject(&s3.GetObjectInput{...})
// result is a *s3.GetObjectOutput struct pointer
// err is a error which can be cast to awserr.Error.
```

Passing Parameters to a Service Operation

When calling an operation on a service, you pass in input parameters as option values, similar to passing in a configuration. For example, to retrieve an object, you must specify a bucket and the object's key by passing in the following parameters to the `GetObject` method:

```
svc := s3.New(session.New())
svc.GetObject(&s3.GetObjectInput{
    Bucket: aws.String("amzn-s3-demo-bucket"),
    Key:    aws.String("keyName"),
})
```

Each service operation has an associated input struct and, usually, an output struct. The structs follow the naming pattern *OperationName* `Input` and *OperationName* `Output`.

For more information about the parameters of each method, see the service client documentation in the AWS SDK for Go API Reference.

Calling Operations with the Request Form

Calling the request form of a service operation, which follows the naming pattern *OperationName* `Request`, provides a simple way to control when a request is built, signed, and sent. Calling the request form immediately returns a request object. The request object output is a struct pointer that is not valid until the request is sent and returned successfully.

Calling the request form can be useful when you want to construct a number of pre-signed requests, such as pre-signed Amazon S3 URLs. You can also use the request form to modify how the SDK sends a request.

The following example calls the request form of the `GetObject` method. The Send method signs the request before sending it.

```
req, result := s3Svc.GetObjectRequest(&s3.GetObjectInput{...})
// result is a *s3.GetObjectOutput struct pointer, not populated until req.Send()
 returns
// req is a *aws.Request struct pointer. Used to Send request.
if err := req.Send(); err != nil {
    // process error
    return
}
// Process result
```

Handling Operation Response Body

Some API operations return a response struct that contain a Body field that is an `io.ReadCloser`. If you're making requests with these operations, always be sure to call `Close` on the field.

```
resp, err := s3svc.GetObject(&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
}
```

Concurrently Using Service Clients

You can create goroutines that concurrently use the same service client to send multiple requests. You can use a service client with as many goroutines as you want. However, you cannot concurrently modify the service client's configuration and request handlers. If you do, the service client operations might encounter race conditions. Define service client settings before you concurrently use it.

In the following example, an Amazon S3 service client is used in multiple goroutines. The example concurrently outputs all objects in `amzn-s3-demo-bucket1`, `amzn-s3-demo-bucket2`, and `amzn-s3-demo-bucket3`, which are all in the same region. To make sure all objects from the same bucket are printed together, the example uses a channel.

```
sess, err := session.NewSession()
if err != nil {
    fmt.Println("Error creating session ", err)
}
var wg sync.WaitGroup
keysCh := make(chan string, 10)

svc := s3.New(sess)
buckets := []string{"amzn-s3-demo-bucket1", "amzn-s3-demo-bucket2", "amzn-s3-demo-
bucket3"}
for _, bucket := range buckets {
    params := &s3.ListObjectsInput{
        Bucket:  aws.String(bucket),
        MaxKeys: aws.Int64(100),
    }
    wg.Add(1)
    go func(param *s3.ListObjectsInput) {
        defer wg.Done()

        err = svc.ListObjectsPages(params,
```

```
            func(page *s3.ListObjectsOutput, last bool) bool {
                // Add the objects to the channel for each page
                for _, object := range page.Contents {
                    keysCh <- fmt.Sprintf("%s:%s", *params.Bucket, *object.Key)
                }
                return true
            },
        )
        if err != nil {
            fmt.Println("Error listing", *params.Bucket, "objects:", err)
        }
    }(params)
}
go func() {
    wg.Wait()
    close(keysCh)
}()
for key := range keysCh {
    // Print out each object key as its discovered
    fmt.Println(key)
}
```

Using Pagination Methods

Typically, when you retrieve a list of items, you might need to check the output for a token or marker to confirm whether AWS returned all results from your request. If present, you use the token or marker to request the next set of results. Instead of managing these tokens or markers, you can use pagination methods provided by the SDK.

Pagination methods iterate over a list operation until the method retrieves the last page of results or until the callback function returns `false`. The names of these methods use the following pattern: *OperationName* `Pages`. For example, the pagination method for the Amazon S3 list objects operation (`ListObjects`) is `ListObjectPages`.

The following example uses the `ListObjectPages` pagination method to list up to three pages of object keys from the `ListObject` operation. Each page consists of up to 10 keys, which is defined by the `MaxKeys` field.

```
svc, err := s3.NewSession(sess)
if err != nil {
    fmt.Println("Error creating session ", err)
```

```
}
inputparams := &s3.ListObjectsInput{
    Bucket:  aws.String("amzn-s3-demo-bucket"),
    MaxKeys: aws.Int64(10),
}
pageNum := 0
svc.ListObjectsPages(inputparams, func(page *s3.ListObjectsOutput, lastPage bool) bool
 {
    pageNum++
    for _, value := range page.Contents {
        fmt.Println(*value.Key)
    }
    return pageNum < 3
})
```

Using Waiters

The SDK provides waiters that continuously check for completion of a job. For example, when you send a request to create an Amazon S3 bucket, you can use a waiter to check when the bucket has been successfully created. That way, subsequent operations on the bucket are done only after the bucket has been created.

The following example uses a waiter that waits until specific instances have stopped.

```
sess, err := session.NewSession(aws.NewConfig().WithRegion("us-west-2"))
if err != nil {
    fmt.Println("Error creating session ", err)
}
// Create an EC2 client
ec2client := ec2.New(sess)
// Specify two instances to stop
instanceIDsToStop := aws.StringSlice([]string{"i-12345678", "i-23456789"})
// Send request to stop instances
_, err = ec2client.StopInstances(&ec2.StopInstancesInput{
  InstanceIds: instanceIDsToStop,
})
if err != nil {
  panic(err)
}
// Use a waiter function to wait until the instances are stopped
describeInstancesInput := &ec2.DescribeInstancesInput{
  InstanceIds: instanceIDsToStop,
```

```
}
if err := ec2client.WaitUntilInstanceStopped(describeInstancesInput); err != nil {
  panic(err)
}
fmt.Println("Instances are stopped.")
```

Handling Errors in the AWS SDK for Go

The AWS SDK for Go returns errors that satisfy the Go `error` interface type and the Error interface in the `aws/awserr` package. You can use the `Error()` method to get a formatted string of the SDK error message without any special handling.

```
if err != nil {
    if awsErr, ok := err.(awserr.Error); ok {
        // process SDK error
    }
}
```

Errors returned by the SDK are backed by a concrete type that will satisfy the `awserr.Error` interface. The interface has the following methods, which provide classification and information about the error.

- `Code` returns the classification code by which related errors are grouped.
- `Message` returns a description of the error.
- `OrigErr` returns the original error of type `error` that is wrapped by the `awserr.Error` interface, such as a standard library error or a service error.

Handling Specific Service Error Codes

The following example demonstrates how to handle error codes that you encounter while using the AWS SDK for Go. The example assumes you have already set up and configured the SDK (that is, all required packages are imported and your credentials and region are set). For more information, see Getting Started with the AWS SDK for Go and Configuring the AWS SDK for Go.

This example highlights how you can use the `awserr.Error` type to perform logic based on specific error codes returned by service API operations.

In this example the *S3 GetObject* API operation is used to request the contents of an object in S3. The example handles the *NoSuchBucket* and *NoSuchKey* error codes, printing custom messages to stderr. If any other error is received, a generic message is printed.

```
svc := s3.New(sess)
resp, err := svc.GetObject(&s3.GetObjectInput{
```

```
    Bucket: aws.String(os.Args[1]),
    Key:    aws.String(os.Args[2]),
})

if err != nil {
    // Casting to the awserr.Error type will allow you to inspect the error
    // code returned by the service in code. The error code can be used
    // to switch on context specific functionality. In this case a context
    // specific error message is printed to the user based on the bucket
    // and key existing.
    //
    // For information on other S3 API error codes see:
    // http://docs.aws.amazon.com/AmazonS3/latest/API/ErrorResponses.html
    if aerr, ok := err.(awserr.Error); ok {
        switch aerr.Code() {
        case s3.ErrCodeNoSuchBucket:
            exitErrorf("bucket %s does not exist", os.Args[1])
        case s3.ErrCodeNoSuchKey:
            exitErrorf("object with key %s does not exist in bucket %s", os.Args[2],
 os.Args[1])
        }
    }
```

See the complete example on GitHub.

Additional Error Information

In addition to the `awserr.Error` interface, you might be able to use other interfaces to get more information about an error.

Specific Error Interfaces

Other packages might provide their own error interfaces. For example, the service/s3/s3manager package provides a MultiUploadFailure interface to retrieve the upload ID. This is helpful when you must manually clean up a failed multi-part upload.

```
output, err := s3manager.Upload(svc, input, opts)
if err != nil {
    if multierr, ok := err.(MultiUploadFailure); ok {
        // Process error and its associated uploadID
        fmt.Println("Error:", multierr.Code(), multierr.Message(), multierr.UploadID())
    } else {
```

```
        // Process error generically
        fmt.Println("Error:", err.Error())
    }
}
```

For more information, see the s3Manager.MultiUploadFailure interface in the AWS SDK for Go API Reference.

AWS SDK for Go Code Examples

The AWS SDK for Go examples can help you write your own Go applications that use Amazon Web Services. The examples assume you have already set up and configured the SDK (that is, you have imported all required packages and set your credentials and region). For more information, see Getting Started with the AWS SDK for Go and Configuring the AWS SDK for Go.

Find the source code for these examples and others in the AWS documentation code examples repository on GitHub. To propose a new code example for the AWS documentation team to consider producing, create a new request. The team is looking to produce code examples that cover broader scenarios and use cases, versus simple code snippets that cover only individual API calls. For instructions, see the *Proposing new code examples* section in the Readme on GitHub.

Topics

- AWS SDK for Go Request Examples
- AWS CloudTrail Examples Using the AWS SDK for Go
- Amazon CloudWatch Examples Using the AWS SDK for Go
- AWS CodeBuild Examples Using the AWS SDK for Go
- Amazon DynamoDB Examples Using the AWS SDK for Go
- Amazon EC2 Examples Using the AWS SDK for Go
- Amazon S3 Glacier Examples Using the AWS SDK for Go
- IAM Examples Using the AWS SDK for Go
- AWS Key Management Service Examples Using the AWS SDK for Go
- AWS Lambda Examples Using the AWS SDK for Go
- Amazon Polly Examples Using the AWS SDK for Go
- Amazon S3 Examples Using the AWS SDK for Go
- Amazon SES Examples Using the AWS SDK for Go
- Amazon SNS Examples Using the AWS SDK for Go
- Amazon SQS Examples Using the AWS SDK for Go
- Amazon WorkDocs Examples

AWS SDK for Go Request Examples

The AWS SDK for Go examples can help you write your own applications. The examples assume you have already set up and configured the SDK (that is, you have imported all required packages and set your credentials and region). For more information, see Getting Started with the AWS SDK for Go and Configuring the AWS SDK for Go.

Using context.Context with SDK Requests

In Go 1.7, the `context.Context` type was added to `http.Request`. This type provides an easy way to implement deadlines and cancellations on requests.

To use this pattern with the SDK, call `WithContext` on the `HTTPRequest` field of the SDK's `request.Request` type, and provide your `Context value`. The following example highlights this process with a timeout on Amazon SQS`ReceiveMessage`.

```
req, resp := svc.ReceiveMessageRequest(params)
req.HTTPRequest = req.HTTPRequest.WithContext(ctx)

err := req.Send()
if err != nil {
    fmt.Println("Got error receiving message:")
    fmt.Println(err.Error())
} else {
    fmt.Println(resp)
}
```

Using API Field Setters with SDK Requests

In addition to setting API parameters by using struct fields, you can also use chainable setters on the API operation parameter fields. This enables you to use a chain of setters to set the fields of the API struct.

```
svc := s3.New(sess)

_, err := svc.PutObject((&s3.PutObjectInput{}).
    SetBucket(*bucket).
    SetKey(*key).
    SetBody(strings.NewReader("object body")), //.
```

```
//      SetWebsiteRedirectLocation("https://example.com/something"),
)
```

You can also use this pattern with nested fields in API operation requests.

```
resp, err := svc.UpdateService((&ecs.UpdateServiceInput{}).
    SetService("myService").
    SetDeploymentConfiguration((&ecs.DeploymentConfiguration{}).
        SetMinimumHealthyPercent(80),
    ),
)
```

AWS CloudTrail Examples Using the AWS SDK for Go

CloudTrail is an AWS service that helps you enable governance, compliance, and operational and risk auditing of your AWS account. Actions taken by a user, role, or an AWS service are recorded as events in CloudTrail. Events include actions taken in the AWS Management Console, AWS Command Line Interface, and AWS SDKs and APIs.

The examples assume you have already set up and configured the SDK (that is, you've imported all required packages and set your credentials and region). For more information, see Getting Started with the AWS SDK for Go and Configuring the AWS SDK for Go.

You can download complete versions of these example files from the aws-doc-sdk-examples repository on GitHub.

Topics

- Listing the CloudTrail Trails
- Creating a CloudTrail Trail
- Listing CloudTrail Trail Events
- Deleting a CloudTrail Trail

Listing the CloudTrail Trails

This example uses the DescribeTrails operation to list the names of the CloudTrail trails and the bucket in which CloudTrail stores information in the `us-west-2` region.

Choose Copy to save the code locally.

Create the file *describe_trails.go*. Add the following statements to import the Go and AWS SDK for Go packages used in the example.

```
import (
    "fmt"

    "github.com/aws/aws-sdk-go/aws/session"
    "github.com/aws/aws-sdk-go/service/cloudtrail"
)
```

Initialize the session that the SDK uses to load credentials from the shared credentials file `.aws/credentials` in your home folder, and create a new service client.

```
sess := session.Must(session.NewSessionWithOptions(session.Options{
    SharedConfigState: session.SharedConfigEnable,
}))

// Create CloudTrail client
svc := cloudtrail.New(sess)
```

Call **DescribeTrails**. If an error occurs, print the error and exit. If no error occurs, loop through the trails, printing the name of each trail and the bucket.

```
resp, err := svc.DescribeTrails(&cloudtrail.DescribeTrailsInput{TrailNameList: nil})
if err != nil {
    fmt.Println("Got error calling CreateTrail:")
    fmt.Println(err.Error())
    return
}

fmt.Println("Found", len(resp.TrailList), "trail(s)")
fmt.Println("")

for _, trail := range resp.TrailList {
    fmt.Println("Trail name:  " + *trail.Name)
    fmt.Println("Bucket name: " + *trail.S3BucketName)
    fmt.Println("")
}
```

See the complete example on GitHub.

Creating a CloudTrail Trail

This example uses the CreateTrail operation to create a CloudTrail trail in the `us-west-2` region. It requires two inputs, the name of the trail and the name of the bucket in which CloudTrail stores information. If the bucket does not have the proper policy, include the **-p** flag to attach the correct policy to the bucket.

Choose Copy to save the code locally.

Create the file *create_trail.go*. Add the following statements to import the Go and AWS SDK for Go packages used in the example.

```
import (
    "github.com/aws/aws-sdk-go/aws"
    "github.com/aws/aws-sdk-go/aws/session"
    "github.com/aws/aws-sdk-go/service/cloudtrail"
    "github.com/aws/aws-sdk-go/service/s3"
    "github.com/aws/aws-sdk-go/service/sts"

    "encoding/json"
    "flag"
    "fmt"
)
```

Get the names of the trail and bucket, and whether to attach the policy to the bucket. If either the trail name or bucket name is missing, display an error message and exit.

```
trailNamePtr := flag.String("n", "", "The name of the trail")
bucketNamePtr := flag.String("b", "", "the name of the bucket to which the trails are
 uploaded")
addPolicyPtr := flag.Bool("p", false, "Whether to add the CloudTrail policy to the
 bucket")

flag.Parse()

if *trailNamePtr == "" || *bucketNamePtr == "" {
    fmt.Println("You must supply a trail name and bucket name.")
    return
}
```

Initialize the session that the SDK uses to load credentials from the shared credentials file `.aws/credentials` in your home folder.

```
sess := session.Must(session.NewSessionWithOptions(session.Options{
    SharedConfigState: session.SharedConfigEnable,
}))
```

If the **-p** flag was specfied, add a policy to the bucket.

```
if *addPolicyPtr {
    svc := sts.New(sess)
    input := &sts.GetCallerIdentityInput{}

    result, err := svc.GetCallerIdentity(input)
    if err != nil {
        fmt.Println("Got error snarfing caller identity:")
        fmt.Println(err.Error())
        return
    }

    accountID := aws.StringValue(result.Account)

    s3Policy := map[string]interface{}{
        "Version": "2012-10-17",
        "Statement": []map[string]interface{}{
            {
                "Sid":    "AWSCloudTrailAclCheck20150319",
                "Effect": "Allow",
                "Principal": map[string]interface{}{
                    "Service": "cloudtrail.amazonaws.com",
                },
                "Action":   "s3:GetBucketAcl",
                "Resource": "arn:aws:s3:::" + *bucketNamePtr,
            },
            {
                "Sid":    "AWSCloudTrailWrite20150319",
                "Effect": "Allow",
                "Principal": map[string]interface{}{
                    "Service": "cloudtrail.amazonaws.com",
                },
                "Action":   "s3:PutObject",
                "Resource": "arn:aws:s3:::" + *bucketNamePtr + "/AWSLogs/" + accountID
 + "/*",
```

```
                "Condition": map[string]interface{}{
                    "StringEquals": map[string]interface{}{
                        "s3:x-amz-acl": "bucket-owner-full-control",
                    },
                },
            },
        },
    }

    policy, err := json.Marshal(s3Policy)
    if err != nil {
        fmt.Println("Error marshalling request")
        return
    }

    // Create S3 service
    s3Svc := s3.New(sess)

    // Now set the policy on the bucket
    _, err = s3Svc.PutBucketPolicy(&s3.PutBucketPolicyInput{
        Bucket: aws.String(*bucketNamePtr),
        Policy: aws.String(string(policy)),
    })
    if err != nil {
        fmt.Print("Got error adding bucket policy:")
        fmt.Print(err.Error())
        return
    }

    fmt.Printf("Successfully set bucket %q's policy\n", *bucketNamePtr)
}
```

Create the CloudTrail client, the input for **CreateTrail**, and call **CreateTrail**. If an error occurs, print the error and exit. If no error occurs, print a success message.

```
svc := cloudtrail.New(sess)

input := &cloudtrail.CreateTrailInput{
    Name:         aws.String(*trailNamePtr),
    S3BucketName: aws.String(*bucketNamePtr),
}

_, err := svc.CreateTrail(input)
```

```
if err != nil {
    fmt.Println("Got error calling CreateTrail:")
    fmt.Println(err.Error())
    return
}

fmt.Println("Created the trail", *trailNamePtr, "for bucket", *bucketNamePtr)
```

See the complete example on GitHub.

Listing CloudTrail Trail Events

This example uses the LookupEvents operation to list the CloudTrail trail events in the `us-west-2` region.

Choose Copy to save the code locally.

Create the file *lookup_events.go*. Add the following statements to import the Go and AWS SDK for Go packages used in the example.

```
import (
    "github.com/aws/aws-sdk-go/aws"
    "github.com/aws/aws-sdk-go/aws/session"
    "github.com/aws/aws-sdk-go/service/cloudtrail"

    "flag"
    "fmt"
    "time"
)
```

Get the name of the trail. If the trail name is missing, display an error message and exit.

```
trailNamePtr := flag.String("n", "", "The name of the trail")

flag.Parse()

if *trailNamePtr == "" {
    fmt.Println("You must supply a trail name")
    return
}
```

Initialize the session that the SDK uses to load credentials from the shared credentials file `.aws/credentials` in your home folder, and create a new service client.

```
sess := session.Must(session.NewSessionWithOptions(session.Options{
    SharedConfigState: session.SharedConfigEnable,
}))
```

Create the CloudTrail client, and the input for and call **LookupEvents**. If an error occurs, print the error and exit. If no error occurs, loop through the events, printing information about each event.

```
svc := cloudtrail.New(sess)

input := &cloudtrail.LookupEventsInput{EndTime: aws.Time(time.Now())}

resp, err := svc.LookupEvents(input)
if err != nil {
    fmt.Println("Got error calling CreateTrail:")
    fmt.Println(err.Error())
    return
}

fmt.Println("Found", len(resp.Events), "events before now")
fmt.Println("")

for _, event := range resp.Events {
    fmt.Println("Event:")
    fmt.Println(aws.StringValue(event.CloudTrailEvent))
    fmt.Println("")
    fmt.Println("Name    ", aws.StringValue(event.EventName))
    fmt.Println("ID:     ", aws.StringValue(event.EventId))
    fmt.Println("Time:   ", aws.TimeValue(event.EventTime))
    fmt.Println("User:   ", aws.StringValue(event.Username))

    fmt.Println("Resources:")

    for _, resource := range event.Resources {
        fmt.Println("  Name:", aws.StringValue(resource.ResourceName))
        fmt.Println("  Type:", aws.StringValue(resource.ResourceType))
    }

    fmt.Println("")
```

See the complete example on GitHub.

Deleting a CloudTrail Trail

This example uses the DeleteTrail operation to delete a CloudTrail trail in the `us-west-2` region. It requires one input, the name of the trail.

Choose Copy to save the code locally.

Create the file *delete_trail.go*. Add the following statements to import the Go and AWS SDK for Go packages used in the example.

```
import (
    "github.com/aws/aws-sdk-go/aws"
    "github.com/aws/aws-sdk-go/aws/session"
    "github.com/aws/aws-sdk-go/service/cloudtrail"

    "flag"
    "fmt"
)
```

Get the name of the trail. If the trail name is missing, display an error message and exit.

```
trailNamePtr := flag.String("n", "", "The name of the trail to delete")

flag.Parse()

if *trailNamePtr == "" {
    fmt.Println("You must supply a trail name")
    return
}
```

Initialize the session that the SDK uses to load credentials from the shared credentials file `.aws/credentials` in your home folder and create the client.

```
sess := session.Must(session.NewSessionWithOptions(session.Options{
    SharedConfigState: session.SharedConfigEnable,
}))
```

Create a CloudTrail client and call **DeleteTrail** with the trail name. If an error occurs, print the error and exit. If no error occurs, print a success message.

```
svc := cloudtrail.New(sess)

_, err := svc.DeleteTrail(&cloudtrail.DeleteTrailInput{Name:
 aws.String(*trailNamePtr)})
if err != nil {
    fmt.Println("Got error calling CreateTrail:")
    fmt.Println(err.Error())
    return
}

fmt.Println("Successfully deleted trail", *trailNamePtr)
```

See the complete example on GitHub.

Amazon CloudWatch Examples Using the AWS SDK for Go

Amazon CloudWatch is a web service that monitors your AWS resources and the applications you run on AWS in real time. You can use CloudWatch to collect and track metrics, which are variables you can measure for your resources and applications. CloudWatch alarms send notifications or automatically make changes to the resources you are monitoring based on rules that you define.

The AWS SDK for Go examples show you how to integrate CloudWatch into your Go applications. The examples assume you have already set up and configured the SDK (that is, you have imported all required packages and set your credentials and region). For more information, see Getting Started with the AWS SDK for Go and Configuring the AWS SDK for Go.

You can download complete versions of these example files from the aws-doc-sdk-examples repository on GitHub.

Topics

- Describing CloudWatch Alarms
- Using Alarms and Alarm Actions in CloudWatch
- Getting Metrics from CloudWatch
- Sending Events to Amazon CloudWatch Events
- Getting Log Events from CloudWatch

Describing CloudWatch Alarms

This example shows you how to retrieve basic information that describes your CloudWatch alarms.

You can download complete versions of these example files from the aws-doc-sdk-examples repository on GitHub.

Scenario

An alarm watches a single metric over a time period you specify. The alarm performs one or more actions based on the value of the metric relative to a given threshold over a number of time periods.

In this example, Go code is used to describe alarms in CloudWatch. The code uses the AWS SDK for Go to describe alarms by using this method of the `AWS.CloudWatch` client class:

- DescribeAlarms

Prerequisites

- You have set up and configured the AWS SDK for Go.
- You are familiar with CloudWatch alarms. To learn more, see Creating Amazon CloudWatch Alarms in the Amazon CloudWatch User Guide.

Describe Alarms

Choose **Copy** to save the code locally.

Create the file `describe_alarms.go`. Import the packages used in the example.

```
import (
    "github.com/aws/aws-sdk-go/aws/session"
    "github.com/aws/aws-sdk-go/service/cloudwatch"

    "fmt"
    "os"
)
```

Initialize a session that the SDK will use to load credentials from the shared credentials file `~/.aws/credentials`, load your configuration from the shared configuration file `~/.aws/config`, and create a CloudWatch client.

```
sess := session.Must(session.NewSessionWithOptions(session.Options{
    SharedConfigState: session.SharedConfigEnable,
}))

svc := cloudwatch.New(sess)
```

Call `DescribeAlarms`, and print the results.

```
resp, err := svc.DescribeAlarms(nil)
for _, alarm := range resp.MetricAlarms {
    fmt.Println(*alarm.AlarmName)
}
```

See the complete example on GitHub.

Using Alarms and Alarm Actions in CloudWatch

These Go examples show you how to change the state of your Amazon EC2 instances automatically based on a CloudWatch alarm, as follows:

- Creating and enabling actions on an alarm
- Disabling actions on an alarm

You can download complete versions of these example files from the aws-doc-sdk-examples repository on GitHub.

Scenario

You can use alarm actions to create alarms that automatically stop, terminate, reboot, or recover your Amazon EC2 instances. You can use the stop or terminate actions when you no longer need an instance to be running. You can use the reboot and recover actions to automatically reboot the instance.

In this example, Go code is used to define an alarm action in CloudWatch that triggers the reboot of an Amazon EC2 instance. The code uses the AWS SDK for Go to manage instances by using these methods of PutMetricAlarm type:

- PutMetricAlarm
- EnableAlarmActions
- DisableAlarmActions

Prerequisites

- You have set up and configured the AWS SDK for Go.
- You are familiar with CloudWatch alarm actions. To learn more, see Create Alarms to Stop, Terminate, Reboot, or Recover an Instance in the Amazon CloudWatch User Guide.

Create and Enable Actions on an Alarm

Choose **Copy** to save the code locally.

Create the file `create_enable_alarms.go`.

Import packages used in the example.

```
import (
    "github.com/aws/aws-sdk-go/aws"
    "github.com/aws/aws-sdk-go/aws/session"
    "github.com/aws/aws-sdk-go/service/cloudwatch"

    "fmt"
    "os"
)
```

Get an instance name, value, and alarm name.

```
if len(os.Args) != 4 {
    fmt.Println("You must supply an instance name, value, and alarm name")
    os.Exit(1)
}

instance := os.Args[1]
value := os.Args[2]
name := os.Args[3]
```

Initialize a session that the SDK will use to load credentials from the shared credentials file `~/.aws/credentials`, load your configuration from the shared configuration file `~/.aws/config`, and create a CloudWatch client.

```
sess := session.Must(session.NewSessionWithOptions(session.Options{
    SharedConfigState: session.SharedConfigEnable,
}))

// Create new CloudWatch client.
svc := cloudwatch.New(sess)
```

Create a metric alarm that reboots an instance if its CPU utilization is greater than 70 percent.

```
_, err := svc.PutMetricAlarm(&cloudwatch.PutMetricAlarmInput{
    AlarmName:          aws.String(name),
    ComparisonOperator: aws.String(cloudwatch.ComparisonOperatorGreaterThanThreshold),
    EvaluationPeriods:  aws.Int64(1),
    MetricName:         aws.String("CPUUtilization"),
    Namespace:          aws.String("AWS/EC2"),
    Period:             aws.Int64(60),
    Statistic:          aws.String(cloudwatch.StatisticAverage),
    Threshold:          aws.Float64(70.0),
    ActionsEnabled:     aws.Bool(true),
    AlarmDescription:   aws.String("Alarm when server CPU exceeds 70%"),
    Unit:               aws.String(cloudwatch.StandardUnitSeconds),

    // This is apart of the default workflow actions. This one will reboot the
 instance, if the
    // alarm is triggered.
    AlarmActions: []*string{
        aws.String(fmt.Sprintf("arn:aws:swf:us-east-1:%s:action/actions/
AWS_EC2.InstanceId.Reboot/1.0", instance)),
    },
    Dimensions: []*cloudwatch.Dimension{
        {
            Name:  aws.String("InstanceId"),
            Value: aws.String(value),
        },
    },
})
```

Call `EnableAlarmActions` with the new alarm for the instance, and display a message.

```
result, err := svc.EnableAlarmActions(&cloudwatch.EnableAlarmActionsInput{
    AlarmNames: []*string{
        aws.String(name),
    },
})
fmt.Println("Alarm action enabled", result)
```

See the complete example on GitHub.

Disable Actions on an Alarm

Choose **Copy** to save the code locally.

Create the file `disable_alarm.go`.

Import the packages used in the example.

```
import (
    "github.com/aws/aws-sdk-go/aws"
    "github.com/aws/aws-sdk-go/aws/session"
    "github.com/aws/aws-sdk-go/service/cloudwatch"

    "fmt"
    "os"
)
```

Get the name of the alarm from the command line.

```
if len(os.Args) != 2 {
    fmt.Println("You must supply an alarm name")
    os.Exit(1)
}
```

Initialize a session that the SDK will use to load credentials from the shared credentials file `~/.aws/credentials`, load your configuration from the shared configuration file `~/.aws/config`, and create a CloudWatch client.

```
sess := session.Must(session.NewSessionWithOptions(session.Options{
    SharedConfigState: session.SharedConfigEnable,
}))

// Create new CloudWatch client.
```

```
svc := cloudwatch.New(sess)
```

Call `DisableAlarmActions` to disable the actions for this alarm and display a message.

```
// Disable the alarm.
_, err := svc.DisableAlarmActions(&cloudwatch.DisableAlarmActionsInput{
    AlarmNames: []*string{
        aws.String(name),
    },
})
fmt.Println("Success")
```

See the complete example on GitHub.

Getting Metrics from CloudWatch

These Go examples show you how to retrieve a list of published CloudWatch metrics and publish data points to CloudWatch metrics with the AWS SDK for Go, as follows:

- Listing metrics
- Submitting custom metrics

You can download complete versions of these example files from the aws-doc-sdk-examples repository on GitHub.

Scenario

Metrics are data about the performance of your systems. You can enable detailed monitoring of some resources, such as your Amazon EC2 instances, or your own application metrics.

In this example, Go code is used to get metrics from CloudWatch and to send events to CloudWatch Events. The code uses the AWS SDK for Go to get metrics from CloudWatch by using these methods of the CloudWatch type:

- ListMetrics
- PutMetricData

Prerequisites

- You have set up and configured the AWS SDK for Go.

- You are familiar with CloudWatch metrics. To learn more, see Using Amazon CloudWatch Metrics in the Amazon CloudWatch User Guide.

List Metrics

Choose **Copy** to save the code locally.

Create the file `listing_metrics.go`.

Import the packages used in the example.

```
import (
    "github.com/aws/aws-sdk-go/aws"
    "github.com/aws/aws-sdk-go/aws/session"
    "github.com/aws/aws-sdk-go/service/cloudwatch"

    "fmt"
    "os"
)
```

Get the metric name, namespace, and dimension name from the command line.

```
if len(os.Args) != 4 {
    fmt.Println("You must supply a metric name, namespace, and dimension name")
    os.Exit(1)
}

metric := os.Args[1]
namespace := os.Args[2]
dimension := os.Args[3]
```

Initialize a session that the SDK will use to load credentials from the shared credentials file `~/.aws/credentials`, load your configuration from the shared configuration file `~/.aws/config`, and create a CloudWatch client.

```
sess := session.Must(session.NewSessionWithOptions(session.Options{
    SharedConfigState: session.SharedConfigEnable,
}))

// Create CloudWatch client
```

```
svc := cloudwatch.New(sess)
```

Call `ListMetrics`, supplying the metric name, namespace, and dimension name. Print the metrics returned in the result.

```
result, err := svc.ListMetrics(&cloudwatch.ListMetricsInput{
    MetricName: aws.String(metric),
    Namespace:  aws.String(namespace),
    Dimensions: []*cloudwatch.DimensionFilter{
        &cloudwatch.DimensionFilter{
            Name: aws.String(dimension),
        },
    },
})
fmt.Println("Metrics", result.Metrics)
```

See the complete example on GitHub.

Submit Custom Metrics

Choose **Copy** to save the code locally.

Create the file `custom_metrics.go`.

Import the packages used in the example.

```
import (
    "github.com/aws/aws-sdk-go/aws"
    "github.com/aws/aws-sdk-go/aws/session"
    "github.com/aws/aws-sdk-go/service/cloudwatch"

    "fmt"
)
```

Initialize a session that the SDK will use to load credentials from the shared credentials file `~/.aws/credentials`, load your configuration from the shared configuration file `~/.aws/config`, and create a CloudWatch client.

```
sess := session.Must(session.NewSessionWithOptions(session.Options{
    SharedConfigState: session.SharedConfigEnable,
}))
```

```
// Create new cloudwatch client.
svc := cloudwatch.New(sess)
```

Call `PutMetricData` with the the custom namespace “Site/Traffic”. The namespace has two custom dimensions: “SiteName” and “PageURL”. “SiteName” has the value “example.com”, the “UniqueVisitors” value 5885 and the “UniqueVisits” value 8628. “PageURL” has the value “my-page.html”, and a “PageViews” value 18057.

```
_, err := svc.PutMetricData(&cloudwatch.PutMetricDataInput{
    Namespace: aws.String("Site/Traffic"),
    MetricData: []*cloudwatch.MetricDatum{
        &cloudwatch.MetricDatum{
            MetricName: aws.String("UniqueVisitors"),
            Unit:       aws.String("Count"),
            Value:      aws.Float64(5885.0),
            Dimensions: []*cloudwatch.Dimension{
                &cloudwatch.Dimension{
                    Name:  aws.String("SiteName"),
                    Value: aws.String("example.com"),
                },
            },
        },
        &cloudwatch.MetricDatum{
            MetricName: aws.String("UniqueVisits"),
            Unit:       aws.String("Count"),
            Value:      aws.Float64(8628.0),
            Dimensions: []*cloudwatch.Dimension{
                &cloudwatch.Dimension{
                    Name:  aws.String("SiteName"),
                    Value: aws.String("example.com"),
                },
            },
        },
        &cloudwatch.MetricDatum{
            MetricName: aws.String("PageViews"),
            Unit:       aws.String("Count"),
            Value:      aws.Float64(18057.0),
            Dimensions: []*cloudwatch.Dimension{
                &cloudwatch.Dimension{
                    Name:  aws.String("PageURL"),
                    Value: aws.String("my-page.html"),
                },
            },
```

```
        },
    },
})
```

If there are any errors, print them out, otherwise list some information about the custom metrics.

```
if err != nil {
    fmt.Println("Error adding metrics:", err.Error())
    return
}

// Get information about metrics
result, err := svc.ListMetrics(&cloudwatch.ListMetricsInput{
    Namespace: aws.String("Site/Traffic"),
})
if err != nil {
    fmt.Println("Error getting metrics:", err.Error())
    return
}

for _, metric := range result.Metrics {
    fmt.Println(*metric.MetricName)

    for _, dim := range metric.Dimensions {
        fmt.Println(*dim.Name + ":", *dim.Value)
        fmt.Println()
    }
}
```

See the complete example on GitHub.

Sending Events to Amazon CloudWatch Events

These Go examples show you how to use the AWS SDK for Go to:

- Create and update a rule used to trigger an event
- Define one or more targets to respond to an event
- Send events that are matched to targets for handling

You can download complete versions of these example files from the aws-doc-sdk-examples repository on GitHub.

Scenario

CloudWatch Events delivers a near real-time stream of system events that describe changes in AWS resources to any of various targets. Using simple rules, you can match events and route them to one or more target functions or streams.

In these examples, Go code is used to send events to CloudWatch Events. The code uses the AWS SDK for Go to manage instances by using these methods of the CloudWatchEvents type:

- PutRule
- PutTargets
- PutEvents

Prerequisites

- You have set up and configured the AWS SDK for Go.
- You are familiar with CloudWatch Events. To learn more, see Adding Events with PutEvents in the Amazon CloudWatch Events User Guide.

Tasks Before You Start

To set up and run this example, you must first complete these tasks:

1. Create a Lambda function using the hello-world blueprint to serve as the target for events. To learn how, see Step 1: Create an AWS Lambda function in the CloudWatch Events User Guide.
2. Create an IAM role whose policy grants permission to CloudWatch Events and that includes `events.amazonaws.com` as a trusted entity. For more information about creating an IAM role, see Creating a Role to Delegate Permissions to an AWS service in the IAM User Guide.

   Use the following role policy when creating the IAM role.

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Sid": "CloudWatchEventsFullAccess",
         "Effect": "Allow",
```

```
         "Action": "events:*",
         "Resource": "*"
      },
      {
         "Sid": "IAMPassRoleForCloudWatchEvents",
         "Effect": "Allow",
         "Action": "iam:PassRole",
         "Resource": "arn:aws:iam::*:role/AWS_Events_Invoke_Targets"
      }
   ]
}
```

Use the following trust relationship when creating the IAM role.

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Effect": "Allow",
         "Principal": {
            "Service": "events.amazonaws.com"
         },
         "Action": "sts:AssumeRole"
      }
   ]
}
```

Create a Scheduled Rule

Choose **Copy** to save the code locally.

Create the file `events_schedule_rule.go`. Import the packages used in the example.

```
import (
    "github.com/aws/aws-sdk-go/aws"
    "github.com/aws/aws-sdk-go/aws/session"
    "github.com/aws/aws-sdk-go/service/cloudwatchevents"

    "fmt"
)
```

Initialize a session that the SDK will use to load credentials from the shared credentials file `~/.aws/credentials`, load your configuration from the shared configuration file `~/.aws/config`, and create a CloudWatch Events client.

```
sess := session.Must(session.NewSessionWithOptions(session.Options{
    SharedConfigState: session.SharedConfigEnable,
}))

// Create the cloudwatch events client
svc := cloudwatchevents.New(sess)
```

Call `PutRule`, supplying a name, `DEMO_EVENT`, ARN of the IAM role you created, `IAM_ROLE_ARN`, and an expression defining the schedule. Finally, display the ARN of the rule.

```
result, err := svc.PutRule(&cloudwatchevents.PutRuleInput{
    Name:               aws.String("DEMO_EVENT"),
    RoleArn:            aws.String("IAM_ROLE_ARN"),
    ScheduleExpression: aws.String("rate(5 minutes)"),
})
fmt.Println("Rule ARN:", result.RuleArn)
```

See the complete example on GitHub.

Add a Lambda Function Target

Choose **Copy** to save the code locally.

Create the file `events_put_targets.go`. Import the packages used in the example.

```
import (
    "github.com/aws/aws-sdk-go/aws"
    "github.com/aws/aws-sdk-go/aws/session"
    "github.com/aws/aws-sdk-go/service/cloudwatchevents"

    "fmt"
)
```

Initialize a session that the SDK will use to load credentials from the shared credentials file `~/.aws/credentials`, load your configuration from the shared configuration file `~/.aws/config`, and create a new CloudWatch Events client.

```
sess := session.Must(session.NewSessionWithOptions(session.Options{
    SharedConfigState: session.SharedConfigEnable,
}))

// Create the cloudwatch events client
svc := cloudwatchevents.New(sess)
```

Call `PutTargets`, supplying a name for the rule, `DEMO_EVENT`. For the target, specify the ARN of the Lambda function you created, `LAMBDA_FUNCTION_ARN`, and the ID of the rule, `myCloudWatchEventsTarget`. Print any errors, or a success message with any targets that failed.

```
result, err := svc.PutTargets(&cloudwatchevents.PutTargetsInput{
    Rule: aws.String("DEMO_EVENT"),
    Targets: []*cloudwatchevents.Target{
        &cloudwatchevents.Target{
            Arn: aws.String("LAMBDA_FUNCTION_ARN"),
            Id:  aws.String("myCloudWatchEventsTarget"),
        },
    },
})
fmt.Println("Success", result)
```

See the complete example on GitHub.

Send Events

Choose **Copy** to save the code locally.

Create the file `events_put_events.go`. Import the packages used in the example.

```
import (
    "github.com/aws/aws-sdk-go/aws"
    "github.com/aws/aws-sdk-go/aws/session"
    "github.com/aws/aws-sdk-go/service/cloudwatchevents"

    "fmt"
)
```

Initialize a session that the SDK will use to load credentials from the shared credentials file `~/.aws/credentials`, load your configuration from the shared configuration file `~/.aws/config`, and create a CloudWatch Events client.

```
sess := session.Must(session.NewSessionWithOptions(session.Options{
    SharedConfigState: session.SharedConfigEnable,
}))

// Create the cloudwatch events client
svc := cloudwatchevents.New(sess)
```

Call `PutEvents`, supplying key-name value pairs in the `Details` field, and specifying the ARN of the Lambda function you created, `RESOURCE_ARN`. See PutEventsRequestEntry for a description of the fields. Finally, display the ingested events.

```
result, err := svc.PutEvents(&cloudwatchevents.PutEventsInput{
    Entries: []*cloudwatchevents.PutEventsRequestEntry{
        &cloudwatchevents.PutEventsRequestEntry{
            Detail:     aws.String("{ \"key1\": \"value1\", \"key2\": \"value2\" }"),
            DetailType: aws.String("appRequestSubmitted"),
            Resources: []*string{
                aws.String("RESOURCE_ARN"),
            },
            Source: aws.String("com.company.myapp"),
        },
    },
})
fmt.Println("Ingested events:", result.Entries)
```

See the complete example on GitHub.

Getting Log Events from CloudWatch

The following example lists up to 100 of the latest events for a log group's log stream. Replace LOG-GROUP-NAME with the name of the CloudWatch log group and LOG-STREAM-NAME with the name of the log stream for the log group.

```
package main

import (
    "flag"
    "fmt"

    "github.com/aws/aws-sdk-go/aws/session"
    "github.com/aws/aws-sdk-go/service/cloudwatchlogs"
```

```
)

// GetLogEvents retrieves CloudWatchLog events.
// Inputs:
//     sess is the current session, which provides configuration for the SDK's service
 clients
//     limit is the maximum number of log events to retrieve
//     logGroupName is the name of the log group
//     logStreamName is the name of the log stream
// Output:
//     If success, a GetLogEventsOutput object containing the events and nil
//     Otherwise, nil and an error from the call to GetLogEvents
func GetLogEvents(sess *session.Session, limit *int64, logGroupName *string,
 logStreamName *string) (*cloudwatchlogs.GetLogEventsOutput, error) {
    svc := cloudwatchlogs.New(sess)

    resp, err := svc.GetLogEvents(&cloudwatchlogs.GetLogEventsInput{
        Limit:         limit,
        LogGroupName:  logGroupName,
        LogStreamName: logStreamName,
    })
    if err != nil {
        return nil, err
    }

    return resp, nil
}

func main() {
    limit := flag.Int64("l", 100, "The maximum number of events to retrieve")
    logGroupName := flag.String("g", "", "The name of the log group")
    logStreamName := flag.String("s", "", "The name of the log stream")
    flag.Parse()

    if *logGroupName == "" || *logStreamName == "" {
        fmt.Println("You must supply a log group name (-g LOG-GROUP) and log stream
 name (-s LOG-STREAM)")
        return
    }

    sess := session.Must(session.NewSessionWithOptions(session.Options{
        SharedConfigState: session.SharedConfigEnable,
    }))
```

```
    resp, err := GetLogEvents(sess, limit, logGroupName, logStreamName)
    if err != nil {
        fmt.Println("Got error getting log events:")
        fmt.Println(err)
        return
    }

    fmt.Println("Event messages for stream " + *logStreamName + " in log group  " +
 *logGroupName)

    gotToken := ""
    nextToken := ""

    for _, event := range resp.Events {
        gotToken = nextToken
        nextToken = *resp.NextForwardToken

        if gotToken == nextToken {
            break
        }

        fmt.Println("  ", *event.Message)
    }
}
```

AWS CodeBuild Examples Using the AWS SDK for Go

CodeBuild is a fully managed build service that compiles source code, runs tests, and produces software packages that are ready to deploy. The AWS SDK for Go examples can integrate AWS CodeBuild into your applications. The examples assume you have already set up and configured the SDK (that is, you've imported all required packages and set your credentials and region). For more information, see Getting Started with the AWS SDK for Go and Configuring the AWS SDK for Go.

You can download complete versions of these example files from the aws-doc-sdk-examples repository on GitHub.

Topics

- Getting Information about All AWS CodeBuild Projects
- Building an AWS CodeBuild Project
- Listing Your AWS CodeBuild Project Builds

Getting Information about All AWS CodeBuild Projects

The following example lists the names of up to 100 of your AWS CodeBuild projects.

```
package main

import (
    "github.com/aws/aws-sdk-go/aws"
    "github.com/aws/aws-sdk-go/aws/session"
    "github.com/aws/aws-sdk-go/service/codebuild"
    "fmt"
    "os"
)

// Lists a CodeBuild projects in the region configured in the shared config
func main() {
    // Initialize a session in us-west-2 that the SDK will use to load
    // credentials from the shared credentials file ~/.aws/credentials.
    sess, err := session.NewSession(&aws.Config{
        Region: aws.String("us-west-2")},
    )

    // Create CodeBuild service client
    svc := codebuild.New(sess)

    // Get the list of projects
    result, err := svc.ListProjects(
        &codebuild.ListProjectsInput{
            SortBy:    aws.String("NAME"),
            SortOrder: aws.String("ASCENDING", )})

    if err != nil {
        fmt.Println("Got error listing projects: ", err)
        os.Exit(1)
    }

    for _, p := range result.Projects {
        fmt.Println(*p)
    }
}
```

Choose Copy to save the code locally. See the complete example on GitHub.

Building an AWS CodeBuild Project

The following example builds the AWS CodeBuild project specified on the command line. If no command-line argument is supplied, it emits an error and quits.

```
package main

import (
    "github.com/aws/aws-sdk-go/aws"
    "github.com/aws/aws-sdk-go/aws/session"
    "github.com/aws/aws-sdk-go/service/codebuild"
    "fmt"
    "os"
)

// Builds a CodeBuild project in the region configured in the shared config
func main() {
    // Requires one argument, the name of the project.
    if len(os.Args) != 2 {
        fmt.Println("Project name required!")
        os.Exit(1)
    }

    project := os.Args[1]

    // Initialize a session in us-west-2 that the SDK will use to load
    // credentials from the shared credentials file ~/.aws/credentials.
    sess, err := session.NewSession(&aws.Config{
        Region: aws.String("us-west-2")},
    )

    // Create CodeBuild service client
    svc := codebuild.New(sess)

    // Build the project
    _, err = svc.StartBuild(&codebuild.StartBuildInput{ProjectName:
 aws.String(project)})
    if err != nil {
        fmt.Println("Got error building project: ", err)
        os.Exit(1)
    }

    fmt.Printf("Started build for project %q\n", project)
```

```
}
```

Choose Copy to save the code locally. See the complete example on GitHub.

Listing Your AWS CodeBuild Project Builds

The following example displays information about your AWS CodeBuild project builds, including the name of the project, when the build started, and how long each phase of the build took, in seconds.

```
package main

import (
    "github.com/aws/aws-sdk-go/aws"
    "github.com/aws/aws-sdk-go/aws/session"
    "github.com/aws/aws-sdk-go/service/codebuild"
    "fmt"
    "os"
)

// Lists the CodeBuild builds for all projects in the region configured in the shared
 config
func main() {
    // Initialize a session in us-west-2 that the SDK will use to load
    // credentials from the shared credentials file ~/.aws/credentials.
    sess, err := session.NewSession(&aws.Config{
        Region: aws.String("us-west-2")},
    )

    // Create CodeBuild service client
    svc := codebuild.New(sess)

    // Get the list of builds
    names, err := svc.ListBuilds(&codebuild.ListBuildsInput{SortOrder:
 aws.String("ASCENDING")})

    if err != nil {
        fmt.Println("Got error listing builds: ", err)
        os.Exit(1)
    }

    // Get information about each build
```

```
    builds, err := svc.BatchGetBuilds(&codebuild.BatchGetBuildsInput{Ids: names.Ids})

    if err != nil {
        fmt.Println("Got error getting builds: ", err)
        os.Exit(1)
    }

    for _, build := range builds.Builds {
        fmt.Printf("Project: %s\n", aws.StringValue(build.ProjectName))
        fmt.Printf("Phase:   %s\n", aws.StringValue(build.CurrentPhase))
        fmt.Printf("Status:  %s\n", aws.StringValue(build.BuildStatus))
        fmt.Println("")
    }
}
```

Choose Copy to save the code locally. See the complete example on GitHub.

Amazon DynamoDB Examples Using the AWS SDK for Go

Amazon DynamoDB is a fully managed NoSQL database service that provides fast and predictable performance with seamless scalability. The AWS SDK for Go examples can integrate Amazon DynamoDB into your Go applications. The examples assume you have already set up and configured the SDK (that is, you have imported all required packages and set your credentials and region). For more information, see Getting Started with the AWS SDK for Go and Configuring the AWS SDK for Go.

The topic also provides a link to a downloadable version of DynamoDB, which includes an interactive web interface so you can experiment with DynamoDB offline.

You can download complete versions of these example files from the aws-doc-sdk-examples repository on GitHub.

Topics

- Listing all Amazon DynamoDB Tables
- Creating an Amazon DynamoDB Table
- Creating an Amazon DynamoDB Table Item
- Creating Amazon DynamoDB Table Items from a JSON File
- Reading an Amazon DynamoDB Table Item
- Getting Amazon DynamoDB Table Items Using Expression Builder

- Updating an Amazon DynamoDB Table Item
- Deleting an Amazon DynamoDB Table Item

Listing all Amazon DynamoDB Tables

The following example uses the DynamoDB ListTables operation to list all of the tables in your default region.

Create the file *DynamoDBListTables.go*. Add the following statements to import the Go and AWS SDK for Go packages used in the example.

```
import (
    "github.com/aws/aws-sdk-go/aws/awserr"
    "github.com/aws/aws-sdk-go/aws/session"
    "github.com/aws/aws-sdk-go/service/dynamodb"

    "fmt"
)
```

Initialize a session that the SDK will use to load credentials from the shared credentials file `~/.aws/credentials` and region from the shared configuration file `~/.aws/config` and create a new DynamoDB service client.

```
// Initialize a session that the SDK will use to load
// credentials from the shared credentials file ~/.aws/credentials
// and region from the shared configuration file ~/.aws/config.
sess := session.Must(session.NewSessionWithOptions(session.Options{
    SharedConfigState: session.SharedConfigEnable,
}))

// Create DynamoDB client
svc := dynamodb.New(sess)
```

Call **ListTables**. If an error occurs, print the error and exit. If no error occurs, loop through the tables, printing the name of each table.

```
// create the input configuration instance
input := &dynamodb.ListTablesInput{}
```

```
fmt.Printf("Tables:\n")

for {
    // Get the list of tables
    result, err := svc.ListTables(input)
    if err != nil {
        if aerr, ok := err.(awserr.Error); ok {
            switch aerr.Code() {
            case dynamodb.ErrCodeInternalServerError:
                fmt.Println(dynamodb.ErrCodeInternalServerError, aerr.Error())
            default:
                fmt.Println(aerr.Error())
            }
        } else {
            // Print the error, cast err to awserr.Error to get the Code and
            // Message from an error.
            fmt.Println(err.Error())
        }
        return
    }

    for _, n := range result.TableNames {
        fmt.Println(*n)
    }

    // assign the last read tablename as the start for our next call to the ListTables
 function
    // the maximum number of table names returned in a call is 100 (default), which
 requires us to make
    // multiple calls to the ListTables function to retrieve all table names
    input.ExclusiveStartTableName = result.LastEvaluatedTableName

    if result.LastEvaluatedTableName == nil {
        break
    }
}
```

See the complete example on GitHub.

Creating an Amazon DynamoDB Table

The following example uses the DynamoDB CreateTable operation to create the table **Music** your default region.

Create the file *DynamoDBCreateTable.go*. Add the following statements to import the Go and AWS SDK for Go packages used in the example.

```
import (
    "github.com/aws/aws-sdk-go/aws"
    "github.com/aws/aws-sdk-go/aws/session"
    "github.com/aws/aws-sdk-go/service/dynamodb"

    "fmt"
    "log"
)
```

Initialize a session that the SDK will use to load credentials from the shared credentials file `~/.aws/credentials` and region from the shared configuration file `~/.aws/config`, and create the DynamoDB client.

```
// Initialize a session that the SDK will use to load
// credentials from the shared credentials file ~/.aws/credentials
// and region from the shared configuration file ~/.aws/config.
sess := session.Must(session.NewSessionWithOptions(session.Options{
    SharedConfigState: session.SharedConfigEnable,
}))

// Create DynamoDB client
svc := dynamodb.New(sess)
```

Call **CreateTable**. If an error occurs, print the error and exit. If no error occurs, print an message that the table was created.

```
// Create table Movies
tableName := "Movies"

input := &dynamodb.CreateTableInput{
    AttributeDefinitions: []*dynamodb.AttributeDefinition{
        {
            AttributeName: aws.String("Year"),
            AttributeType: aws.String("N"),
        },
        {
            AttributeName: aws.String("Title"),
            AttributeType: aws.String("S"),
```

```
        },
    },
    KeySchema: []*dynamodb.KeySchemaElement{
        {
            AttributeName: aws.String("Year"),
            KeyType:       aws.String("HASH"),
        },
        {
            AttributeName: aws.String("Title"),
            KeyType:       aws.String("RANGE"),
        },
    },
    ProvisionedThroughput: &dynamodb.ProvisionedThroughput{
        ReadCapacityUnits:  aws.Int64(10),
        WriteCapacityUnits: aws.Int64(10),
    },
    TableName: aws.String(tableName),
}

_, err := svc.CreateTable(input)
if err != nil {
    log.Fatalf("Got error calling CreateTable: %s", err)
}

fmt.Println("Created the table", tableName)
```

See the complete example on GitHub.

Creating an Amazon DynamoDB Table Item

The following example uses the DynamoDB PutItem operation to create the table item with the `year` **2015** and `title` **The Big New Movie** in the `Movies` table in your default region.

Create the file *DynamoDBCreateItem.go*. Add the following statements to import the Go and AWS SDK for Go packages used in the example.

```
import (
    "github.com/aws/aws-sdk-go/aws"
    "github.com/aws/aws-sdk-go/aws/session"
    "github.com/aws/aws-sdk-go/service/dynamodb"
    "github.com/aws/aws-sdk-go/service/dynamodb/dynamodbattribute"

    "fmt"
```

```
    "log"
    "strconv"
)
```

Create the data structure we use to containing the information about the table item.

```
// Create struct to hold info about new item
type Item struct {
    Year   int
    Title  string
    Plot   string
    Rating float64
}
```

Initialize a session that the SDK will use to load credentials from the shared credentials file `~/.aws/credentials` and region from the shared configuration file `~/.aws/config`, and create the DynamoDB client.

```
// Initialize a session that the SDK will use to load
// credentials from the shared credentials file ~/.aws/credentials
// and region from the shared configuration file ~/.aws/config.
sess := session.Must(session.NewSessionWithOptions(session.Options{
    SharedConfigState: session.SharedConfigEnable,
}))

// Create DynamoDB client
svc := dynamodb.New(sess)
```

Create a struct with the movie data and marshall that data into a map of **AttributeValue** objects.

```
item := Item{
    Year:   2015,
    Title:  "The Big New Movie",
    Plot:   "Nothing happens at all.",
    Rating: 0.0,
}

av, err := dynamodbattribute.MarshalMap(item)
if err != nil {
    log.Fatalf("Got error marshalling new movie item: %s", err)
}
```

Create the input for **PutItem** and call it. If an error occurs, print the error and exit. If no error occurs, print an message that the item was added to the table.

```
// Create item in table Movies
tableName := "Movies"

input := &dynamodb.PutItemInput{
    Item:      av,
    TableName: aws.String(tableName),
}

_, err = svc.PutItem(input)
if err != nil {
    log.Fatalf("Got error calling PutItem: %s", err)
}

year := strconv.Itoa(item.Year)

fmt.Println("Successfully added '" + item.Title + "' (" + year + ") to table " +
 tableName)
```

See the complete example on GitHub.

Creating Amazon DynamoDB Table Items from a JSON File

The following example uses the DynamoDB PutItem operation in a loop to create the items defined in *movie_data.json* file in the `Movies` table in your default region.

Create the file *DynamoDBLoadItems.go*. Add the following statements to import the Go and AWS SDK for Go packages used in the example.

```
import (
    "github.com/aws/aws-sdk-go/aws"
    "github.com/aws/aws-sdk-go/aws/session"
    "github.com/aws/aws-sdk-go/service/dynamodb"
    "github.com/aws/aws-sdk-go/service/dynamodb/dynamodbattribute"

    "encoding/json"
    "fmt"
    "log"
    "io/ioutil"
    "strconv"
```

```
)
```

Create the data structure we use to contain the information about the table item.

```
// Create struct to hold info about new item
type Item struct {
    Year   int
    Title  string
    Plot   string
    Rating float64
}
```

Create a function to get the table items from the JSON file.

```
// Get table items from JSON file
func getItems() []Item {
    raw, err := ioutil.ReadFile("./.movie_data.json")
    if err != nil {
        log.Fatalf("Got error reading file: %s", err)
    }

    var items []Item
    json.Unmarshal(raw, &items)
    return items
}
```

Initialize a session that the SDK will use to load credentials from the shared credentials file `~/.aws/credentials` and region from the shared configuration file `~/.aws/config` and create a new DynamoDB service client.

```
// Initialize a session that the SDK will use to load
// credentials from the shared credentials file ~/.aws/credentials
// and region from the shared configuration file ~/.aws/config.
sess := session.Must(session.NewSessionWithOptions(session.Options{
    SharedConfigState: session.SharedConfigEnable,
}))

// Create DynamoDB client
svc := dynamodb.New(sess)
```

Call **getItems** to get the items. Loop through each item, marshall that data into a map of **AttributeValue** objects, add the item to the `Movies` table, and print out the title and year of the movie added to the table.

```
// Get table items from .movie_data.json
items := getItems()

// Add each item to Movies table:
tableName := "Movies"

for _, item := range items {
    av, err := dynamodbattribute.MarshalMap(item)
    if err != nil {
        log.Fatalf("Got error marshalling map: %s", err)
    }

    // Create item in table Movies
    input := &dynamodb.PutItemInput{
        Item:      av,
        TableName: aws.String(tableName),
    }

    _, err = svc.PutItem(input)
    if err != nil {
        log.Fatalf("Got error calling PutItem: %s", err)
    }

    year := strconv.Itoa(item.Year)

    fmt.Println("Successfully added '" + item.Title + "' (" + year + ") to table " +
 tableName)
```

See the complete example and a sample JSON file on GitHub.

Reading an Amazon DynamoDB Table Item

The following example uses the DynamoDB GetItem operation to retrieve information about the item with the `year` **2015** and `title` **The Big New Movie** in the `movies` table in your default region.

Create the file *DynamoDBReadItem.go*. Add the following statements to import the Go and AWS SDK for Go packages used in the example.

```
import (
    "github.com/aws/aws-sdk-go/aws"
    "github.com/aws/aws-sdk-go/aws/session"
    "github.com/aws/aws-sdk-go/service/dynamodb"
    "github.com/aws/aws-sdk-go/service/dynamodb/dynamodbattribute"

    "fmt"
    "log"
)
```

Create the data structure we use to contain the information about the table item.

```
// Create struct to hold info about new item
type Item struct {
    Year   int
    Title  string
    Plot   string
    Rating float64
}
```

Initialize a session that the SDK will use to load credentials from the shared credentials file `~/.aws/credentials` and region from the shared configuration file `~/.aws/config` and create a new DynamoDB service client.

```
// Initialize a session that the SDK will use to load
// credentials from the shared credentials file ~/.aws/credentials
// and region from the shared configuration file ~/.aws/config.
sess := session.Must(session.NewSessionWithOptions(session.Options{
    SharedConfigState: session.SharedConfigEnable,
}))

// Create DynamoDB client
svc := dynamodb.New(sess)
```

Call **GetItem** to get the item from the table. If we encounter an error, print the error message. Otherwise, display information about the item.

```
tableName := "Movies"
movieName := "The Big New Movie"
movieYear := "2015"
```

```
result, err := svc.GetItem(&dynamodb.GetItemInput{
    TableName: aws.String(tableName),
    Key: map[string]*dynamodb.AttributeValue{
        "Year": {
            N: aws.String(movieYear),
        },
        "Title": {
            S: aws.String(movieName),
        },
    },
})
if err != nil {
    log.Fatalf("Got error calling GetItem: %s", err)
}
```

If an item was returned, unmarshall the return value and display the year, title, plot, and rating.

```
if result.Item == nil {
    msg := "Could not find '" + *title + "'"
    return nil, errors.New(msg)
}

item := Item{}

err = dynamodbattribute.UnmarshalMap(result.Item, &item)
if err != nil {
    panic(fmt.Sprintf("Failed to unmarshal Record, %v", err))
}

fmt.Println("Found item:")
fmt.Println("Year:  ", item.Year)
fmt.Println("Title: ", item.Title)
fmt.Println("Plot:  ", item.Plot)
fmt.Println("Rating:", item.Rating)
```

See the complete example on GitHub.

Getting Amazon DynamoDB Table Items Using Expression Builder

The following example uses the DynamoDB Scan operation to get items with a `rating` greater than **4.0** in the `year` **2013** in the `Movies` table in your default region.

The example uses the Expression Builder package released in version 1.11.0 of the AWS SDK for Go in September 2017.

Create the file *DynamoDBScanItem.go*. Add the following statements to import the Go and AWS SDK for Go packages used in the example.

```
import (
    "github.com/aws/aws-sdk-go/aws"
    "github.com/aws/aws-sdk-go/aws/session"
    "github.com/aws/aws-sdk-go/service/dynamodb"
    "github.com/aws/aws-sdk-go/service/dynamodb/dynamodbattribute"
    "github.com/aws/aws-sdk-go/service/dynamodb/expression"

    "fmt"
    "log"
)
```

Create the data structure we use to contain the information about the table item.

```
// Create struct to hold info about new item
type Item struct {
    Year   int
    Title  string
    Plot   string
    Rating float64
}
```

Initialize a session that the SDK will use to load credentials from the shared credentials file `~/.aws/credentials` and region from the shared configuration file `~/.aws/config` and create a new DynamoDB service client.

```
// Initialize a session that the SDK will use to load
// credentials from the shared credentials file ~/.aws/credentials
// and region from the shared configuration file ~/.aws/config.
sess := session.Must(session.NewSessionWithOptions(session.Options{
    SharedConfigState: session.SharedConfigEnable,
}))

// Create DynamoDB client
svc := dynamodb.New(sess)
```

Create variables for the minimum rating and year for the table items to retrieve.

```
tableName := "Movies"
minRating := 4.0
year := 2013
```

Create the expression defining the year for which we filter the table items to retrieve, and the projection so we get the title, year, and rating for each retrieved item. Then build the expression.

```
// Create the Expression to fill the input struct with.
// Get all movies in that year; we'll pull out those with a higher rating later
filt := expression.Name("Year").Equal(expression.Value(year))

// Or we could get by ratings and pull out those with the right year later
//    filt := expression.Name("info.rating").GreaterThan(expression.Value(min_rating))

// Get back the title, year, and rating
proj := expression.NamesList(expression.Name("Title"), expression.Name("Year"),
 expression.Name("Rating"))

expr, err := expression.NewBuilder().WithFilter(filt).WithProjection(proj).Build()
if err != nil {
    log.Fatalf("Got error building expression: %s", err)
}
```

Create the inputs for and call **Scan** to retrieve the items from the table (the movies made in 2013).

```
// Build the query input parameters
params := &dynamodb.ScanInput{
    ExpressionAttributeNames:  expr.Names(),
    ExpressionAttributeValues: expr.Values(),
    FilterExpression:          expr.Filter(),
    ProjectionExpression:      expr.Projection(),
    TableName:                 aws.String(tableName),
}

// Make the DynamoDB Query API call
result, err := svc.Scan(params)
if err != nil {
    log.Fatalf("Query API call failed: %s", err)
}
```

Loop through the movies from 2013 and display the title and rating for those where the rating is greater than 4.0

```
numItems := 0

for _, i := range result.Items {
    item := Item{}

    err = dynamodbattribute.UnmarshalMap(i, &item)

    if err != nil {
        log.Fatalf("Got error unmarshalling: %s", err)
    }

    // Which ones had a higher rating than minimum?
    if item.Rating > minRating {
        // Or it we had filtered by rating previously:
        //   if item.Year == year {
        numItems++

        fmt.Println("Title: ", item.Title)
        fmt.Println("Rating:", item.Rating)
        fmt.Println()
    }
}

fmt.Println("Found", numItems, "movie(s) with a rating above", minRating, "in", year)
```

See the complete example on GitHub.

Updating an Amazon DynamoDB Table Item

The following example uses the DynamoDB UpdateItem operation to update the rating to **0.5** for the item with the `year` **2015** and `title` **The Big New Movie** in the `Movies` table in your default region.

Create the file *DynamoDBUpdateItem.go*. Add the following statements to import the Go and AWS SDK for Go packages used in the example.

```
import (
    "github.com/aws/aws-sdk-go/aws"
    "github.com/aws/aws-sdk-go/aws/session"
    "github.com/aws/aws-sdk-go/service/dynamodb"

    "fmt"
```

```
    "log"
)
```

Initialize a session that the SDK will use to load credentials from the shared credentials file `~/.aws/credentials` and region from the shared configuration file `~/.aws/config` and create a new DynamoDB service client.

```
// Initialize a session that the SDK will use to load
// credentials from the shared credentials file ~/.aws/credentials
// and region from the shared configuration file ~/.aws/config.
sess := session.Must(session.NewSessionWithOptions(session.Options{
    SharedConfigState: session.SharedConfigEnable,
}))

// Create DynamoDB client
svc := dynamodb.New(sess)
```

Call **UpdateItem** to add the item to the table. If we encounter an error, print the error message. Otherwise, display a message that the item was updated.

```
// Update item in table Movies
tableName := "Movies"
movieName := "The Big New Movie"
movieYear := "2015"
movieRating := "0.5"

input := &dynamodb.UpdateItemInput{
    ExpressionAttributeValues: map[string]*dynamodb.AttributeValue{
        ":r": {
            N: aws.String(movieRating),
        },
    },
    TableName: aws.String(tableName),
    Key: map[string]*dynamodb.AttributeValue{
        "Year": {
            N: aws.String(movieYear),
        },
        "Title": {
            S: aws.String(movieName),
        },
    },
    ReturnValues:     aws.String("UPDATED_NEW"),
```

```
    UpdateExpression: aws.String("set Rating = :r"),
}

_, err := svc.UpdateItem(input)
if err != nil {
    log.Fatalf("Got error calling UpdateItem: %s", err)
}

fmt.Println("Successfully updated '" + movieName + "' (" + movieYear + ") rating to " +
 movieRating)
```

See the complete example on GitHub.

Deleting an Amazon DynamoDB Table Item

The following example uses the DynamoDB DeleteItem operation to delete the item with the `year` **2015** and `title` **The Big New Movie** from the `Movies` table in your default region.

Create the file *DynamoDBUpdateItem.go*. Add the following statements to import the Go and AWS SDK for Go packages used in the example.

```
import (
    "github.com/aws/aws-sdk-go/aws"
    "github.com/aws/aws-sdk-go/aws/session"
    "github.com/aws/aws-sdk-go/service/dynamodb"

    "fmt"
    "log"
)
```

Initialize a session that the SDK will use to load credentials from the shared credentials file `~/.aws/credentials` and region from the shared configuration file `~/.aws/config` and create a new DynamoDB service client.

```
// Initialize a session that the SDK will use to load
// credentials from the shared credentials file ~/.aws/credentials
// and region from the shared configuration file ~/.aws/config.
sess := session.Must(session.NewSessionWithOptions(session.Options{
    SharedConfigState: session.SharedConfigEnable,
}))

// Create DynamoDB client
```

```
svc := dynamodb.New(sess)
```

Call **DeleteItem** to delete the item from the table. If we encounter an error, print the error message. Otherwise, display a message that the item was deleted.

```
tableName := "Movies"
movieName := "The Big New Movie"
movieYear := "2015"

input := &dynamodb.DeleteItemInput{
    Key: map[string]*dynamodb.AttributeValue{
        "Year": {
            N: aws.String(movieYear),
        },
        "Title": {
            S: aws.String(movieName),
        },
    },
    TableName: aws.String(tableName),
}

_, err := svc.DeleteItem(input)
if err != nil {
    log.Fatalf("Got error calling DeleteItem: %s", err)
}

fmt.Println("Deleted '" + movieName + "' (" + movieYear + ") from table " + tableName)
```

See the complete example on GitHub.

Amazon EC2 Examples Using the AWS SDK for Go

Amazon Elastic Compute Cloud (Amazon EC2) is a web service that provides resizable computing capacity—literally servers in Amazon's data centers—that you use to build and host your software systems. The AWS SDK for Go examples can integrate Amazon EC2 into your Go applications. The examples assume you have already set up and configured the SDK (that is, you have imported all required packages and set your credentials and region). For more information, see Getting Started with the AWS SDK for Go and Configuring the AWS SDK for Go.

You can download complete versions of these example files from the aws-doc-sdk-examples repository on GitHub.

Topics

- Creating Amazon EC2 Instances with Tags or without Block Devices
- Managing Amazon EC2 Instances
- Working with Amazon EC2 Key Pairs
- Using Regions and Availability Zones with Amazon EC2
- Working with Security Groups in Amazon EC2
- Using Elastic IP Addresses in Amazon EC2

Creating Amazon EC2 Instances with Tags or without Block Devices

This Go example shows you how to:

- Create an Amazon EC2 instance with tags or set up an instance without a block device

You can download complete versions of these example files from the aws-doc-sdk-examples repository on GitHub.

Scenarios

In these examples, you use a series of Go routines to create Amazon EC2 instances with tags or set up an instance without a block device.

The routines use the AWS SDK for Go to perform these tasks by using these methods of the EC2 type:

- BlockDeviceMapping
- RunInstances
- CreateTags

Prerequisites

- You have set up and configured the AWS SDK for Go.

Create an Instance with Tags

The Amazon EC2 service has an operation for creating instances (RunInstances) and another for attaching tags to instances (CreateTags). To create an instance with tags, call both of these operations in succession. The following example creates an instance and then adds a `Name` tag to it. The Amazon EC2 console displays the value of the `Name` tag in its list of instances.

```
package main

import (
    "github.com/aws/aws-sdk-go/aws"
    "github.com/aws/aws-sdk-go/aws/session"
    "github.com/aws/aws-sdk-go/service/ec2"

    "fmt"
    "log"
)

func main() {
    sess, err := session.NewSession(&aws.Config{
        Region: aws.String("us-west-2")},
    )

    // Create EC2 service client
    svc := ec2.New(sess)

    // Specify the details of the instance that you want to create.
    runResult, err := svc.RunInstances(&ec2.RunInstancesInput{
        // An Amazon Linux AMI ID for t2.micro instances in the us-west-2 region
        ImageId:      aws.String("ami-e7527ed7"),
        InstanceType: aws.String("t2.micro"),
        MinCount:     aws.Int64(1),
        MaxCount:     aws.Int64(1),
    })

    if err != nil {
        fmt.Println("Could not create instance", err)
        return
    }

    fmt.Println("Created instance", *runResult.Instances[0].InstanceId)

    // Add tags to the created instance
```

```
    _, errtag := svc.CreateTags(&ec2.CreateTagsInput{
        Resources: []*string{runResult.Instances[0].InstanceId},
        Tags: []*ec2.Tag{
            {
                Key:   aws.String("Name"),
                Value: aws.String("MyFirstInstance"),
            },
        },
    })
    if errtag != nil {
        log.Println("Could not create tags for instance",
 runResult.Instances[0].InstanceId, errtag)
        return
    }

    fmt.Println("Successfully tagged instance")
}
```

You can add up to 10 tags to an instance in a single `CreateTags` operation.

See the complete example on GitHub.

Create an Image without a Block Device

Sometimes when you create an Amazon EC2 image, you might want to explicitly exclude certain block devices. To do this, you can use the `NoDevice` parameter in BlockDeviceMapping. When this parameter is set to an empty string `""`, the named device isn't mapped.

The `NoDevice` parameter is compatible only with `DeviceName`, not with any other field in `BlockDeviceMapping`. The request will fail if other parameters are present.

```
package main

import (
    "github.com/aws/aws-sdk-go/aws"
    "github.com/aws/aws-sdk-go/aws/session"
    "github.com/aws/aws-sdk-go/service/ec2"

    "fmt"
)

func main() {
    // Load session from shared config
```

```
    sess := session.Must(session.NewSessionWithOptions(session.Options{
        SharedConfigState: session.SharedConfigEnable,
    }))

    // Create EC2 service client
    svc := ec2.New(sess)

    opts := &ec2.CreateImageInput{
        Description: aws.String("image description"),
        InstanceId:  aws.String("i-abcdef12"),
        Name:        aws.String("image name"),
        BlockDeviceMappings: []*ec2.BlockDeviceMapping{
            {
                DeviceName: aws.String("/dev/sda1"),
                NoDevice:    aws.String(""),
            },
            {
                DeviceName: aws.String("/dev/sdb"),
                NoDevice:    aws.String(""),
            },
            {
                DeviceName: aws.String("/dev/sdc"),
                NoDevice:    aws.String(""),
            },
        },
    }
    resp, err := svc.CreateImage(opts)
    if err != nil {
        fmt.Println(err)
        return
    }

    fmt.Println("ID: ", resp.ImageId)
}
```

See the complete example on GitHub.

Managing Amazon EC2 Instances

These Go examples show you how to:

- Describe Amazon EC2 instances
- Manage Amazon EC2 instance monitoring

- Start and stop Amazon EC2 instances
- Reboot Amazon EC2 instances

You can download complete versions of these example files from the aws-doc-sdk-examples repository on GitHub.

Scenario

In these examples, you use a series of Go routines to perform several basic instance management operations.

The routines use the AWS SDK for Go to perform the operations by using these methods of the Amazon EC2 client class:

- DescribeInstances
- MonitorInstances
- UnmonitorInstances
- StartInstances
- StopInstances
- RebootInstances

Prerequisites

- You have set up and configured the AWS SDK for Go.
- You are familiar with the lifecycle of Amazon EC2 instances. To learn more, see Instance Lifecycle in the Amazon EC2 User Guide.

Describe Your Instances

Create a new Go file named `describing_instances.go`.

The Amazon EC2 service has an operation for describing instances, DescribeInstances.

Import the required AWS SDK for Go packages.

```
package main
```

```
import (
    "github.com/aws/aws-sdk-go/aws/session"
    "github.com/aws/aws-sdk-go/service/ec2"

    "fmt"
)
```

Use the following code to create a session and Amazon EC2 client.

```
// Load session from shared config
sess := session.Must(session.NewSessionWithOptions(session.Options{
    SharedConfigState: session.SharedConfigEnable,
}))

// Create new EC2 client
ec2Svc := ec2.New(sess)
```

Call `DescribeInstances` to get detailed information for each instance.

```
// Call to get detailed information on each instance
result, err := ec2Svc.DescribeInstances(nil)
if err != nil {
    fmt.Println("Error", err)
} else {
    fmt.Println("Success", result)
}
```

See the complete example on GitHub.

Manage Instance Monitoring

Create a new Go file named `monitoring_instances.go`.

Import the required AWS SDK for Go packages.

```
package main

import (
    "fmt"
    "os"
```

```
    "github.com/aws/aws-sdk-go/aws"
    "github.com/aws/aws-sdk-go/aws/awserr"
    "github.com/aws/aws-sdk-go/aws/session"
    "github.com/aws/aws-sdk-go/service/ec2"
)
```

To access Amazon EC2, create an EC2 client.

```
func main() {
    // Load session from shared config
    sess := session.Must(session.NewSessionWithOptions(session.Options{
        SharedConfigState: session.SharedConfigEnable,
    }))

    // Create new EC2 client
    svc := ec2.New(sess)
```

Based on the value of a command-line argument (ON or OFF), call either the `MonitorInstances` method of the Amazon EC2 service object to begin detailed monitoring of the specified instances, or the `UnmonitorInstances` method. Before you try to change the monitoring of these instances, use the `DryRun` parameter to test whether you have permission to change instance monitoring.

```
    if os.Args[1] == "ON" {
        input := &ec2.MonitorInstancesInput{
            InstanceIds: []*string{
                aws.String(os.Args[2]),
            },
            DryRun: aws.Bool(true),
        }
        result, err := svc.MonitorInstances(input)
        awsErr, ok := err.(awserr.Error)

        if ok && awsErr.Code() == "DryRunOperation" {
            input.DryRun = aws.Bool(false)
            result, err = svc.MonitorInstances(input)
            if err != nil {
                fmt.Println("Error", err)
            } else {
                fmt.Println("Success", result.InstanceMonitorings)
            }
```

```
        } else {
            fmt.Println("Error", err)
        }
    } else if os.Args[1] == "OFF" { // Turn monitoring off
        input := &ec2.UnmonitorInstancesInput{
            InstanceIds: []*string{
                aws.String(os.Args[2]),
            },
            DryRun: aws.Bool(true),
        }
        result, err := svc.UnmonitorInstances(input)
        awsErr, ok := err.(awserr.Error)
        if ok && awsErr.Code() == "DryRunOperation" {
            input.DryRun = aws.Bool(false)
            result, err = svc.UnmonitorInstances(input)
            if err != nil {
                fmt.Println("Error", err)
            } else {
                fmt.Println("Success", result.InstanceMonitorings)
            }
        } else {
            fmt.Println("Error", err)
        }
    }

}
```

See the complete example on GitHub.

Start and Stop Instances

Create a new Go file named `start_stop_instances.go`.

Import the required AWS SDK for Go packages.

```
package main

import (
    "fmt"
    "os"

    "github.com/aws/aws-sdk-go/aws"
    "github.com/aws/aws-sdk-go/aws/awserr"
```

```
    "github.com/aws/aws-sdk-go/aws/session"
    "github.com/aws/aws-sdk-go/service/ec2"
)
```

To access Amazon EC2, create an EC2 client. The user will pass in a state value of START or STOP and the instance ID.

```
func main() {
    // Load session from shared config
    sess := session.Must(session.NewSessionWithOptions(session.Options{
        SharedConfigState: session.SharedConfigEnable,
    }))

    // Create new EC2 client
    svc := ec2.New(sess)
```

Based on the value of a command-line argument (START or STOP), call either the `StartInstances` method of the Amazon EC2 service object to start the specified instances, or the `StopInstances` method to stop them. Before you try to start or stop the selected instances, use the `DryRun` parameter to test whether you have permission to start or stop them.

```
    if os.Args[1] == "START" {
        input := &ec2.StartInstancesInput{
            InstanceIds: []*string{
                aws.String(os.Args[2]),
            },
            DryRun: aws.Bool(true),
        }
        result, err := svc.StartInstances(input)
        awsErr, ok := err.(awserr.Error)

        if ok && awsErr.Code() == "DryRunOperation" {
            // Let's now set dry run to be false. This will allow us to start the
 instances
            input.DryRun = aws.Bool(false)
            result, err = svc.StartInstances(input)
            if err != nil {
                fmt.Println("Error", err)
            } else {
                fmt.Println("Success", result.StartingInstances)
            }
        } else { // This could be due to a lack of permissions
```

```
            fmt.Println("Error", err)
        }
    } else if os.Args[1] == "STOP" { // Turn instances off
        input := &ec2.StopInstancesInput{
            InstanceIds: []*string{
                aws.String(os.Args[2]),
            },
            DryRun: aws.Bool(true),
        }
        result, err := svc.StopInstances(input)
        awsErr, ok := err.(awserr.Error)
        if ok && awsErr.Code() == "DryRunOperation" {
            input.DryRun = aws.Bool(false)
            result, err = svc.StopInstances(input)
            if err != nil {
                fmt.Println("Error", err)
            } else {
                fmt.Println("Success", result.StoppingInstances)
            }
        } else {
            fmt.Println("Error", err)
        }
    }
}
```

See the complete example on GitHub.

Reboot Instances

Create a new Go file named `reboot_instances.go`.

Import the required AWS SDK for Go packages.

```
package main

import (
    "fmt"
    "os"

    "github.com/aws/aws-sdk-go/aws"
    "github.com/aws/aws-sdk-go/aws/awserr"
    "github.com/aws/aws-sdk-go/aws/session"
    "github.com/aws/aws-sdk-go/service/ec2"
```

```
)
```

To access Amazon EC2, create an EC2 client. The user will pass in a state value of START or STOP and the instance ID.

```
    sess := session.Must(session.NewSessionWithOptions(session.Options{
        SharedConfigState: session.SharedConfigEnable,
    }))

    // Create new EC2 client
    svc := ec2.New(sess)
```

Based on the value of a command-line argument (START or STOP), call either the `StartInstances` method of the Amazon EC2 service object to start the specified instances, or the `StopInstances` method to stop them. Before you try to reboot the selected instances, use the `DryRun` parameter to test whether the instance exists and that you have permission to reboot it.

```
    input := &ec2.RebootInstancesInput{
        InstanceIds: []*string{
            aws.String(os.Args[1]),
        },
        DryRun: aws.Bool(true),
    }
    result, err := svc.RebootInstances(input)
    awsErr, ok := err.(awserr.Error)
```

If the error code is `DryRunOperation`, it means that you do have the permissions you need to reboot the instance.

```
    if ok && awsErr.Code() == "DryRunOperation" {
        input.DryRun = aws.Bool(false)
        result, err = svc.RebootInstances(input)
        if err != nil {
            fmt.Println("Error", err)
        } else {
            fmt.Println("Success", result)
        }
    } else { // This could be due to a lack of permissions
        fmt.Println("Error", err)
    }
```

```
}
```

See the complete example on GitHub.

Working with Amazon EC2 Key Pairs

These Go examples show you how to:

- Describe an Amazon EC2 key pair
- Create an Amazon EC2 key pair
- Delete an Amazon EC2 key pair

You can download complete versions of these example files from the aws-doc-sdk-examples repository on GitHub.

Scenario

Amazon EC2 uses public–key cryptography to encrypt and decrypt login information. Public–key cryptography uses a public key to encrypt data, then the recipient uses the private key to decrypt the data. The public and private keys are known as a key pair.

The routines use the AWS SDK for Go to perform these tasks by using these methods of the EC2 type:

- CreateKeyPair
- DeleteKeyPair
- DescribeKeyPairs

Prerequisites

- You have set up and configured the SDK.
- You are familiar with Amazon EC2 key pairs. To learn more, see Amazon EC2 Key Pairs in the Amazon EC2 User Guide.

Describe Your Key Pairs

Create a new Go file named `ec2_describe_keypairs.go`.

Import the required AWS SDK for Go packages.

```
package main

import (
    "github.com/aws/aws-sdk-go/aws"
    "github.com/aws/aws-sdk-go/aws/session"
    "github.com/aws/aws-sdk-go/service/ec2"

    "fmt"
    "os"
)
```

Use the following code to create a session and Amazon EC2 client.

```
//    go run ec2_describe_keypairs.go
    // credentials from the shared credentials file ~/.aws/credentials.
    sess, err := session.NewSession(&aws.Config{
        Region: aws.String("us-west-2")},
    )

    // Create an EC2 service client.
```

Call `DescribeKeyPairs` to get a list of key pairs and print them out.

```
    //  Returns a list of key pairs
    result, err := svc.DescribeKeyPairs(nil)
    if err != nil {
        exitErrorf("Unable to get key pairs, %v", err)
    }

    fmt.Println("Key Pairs:")
    for _, pair := range result.KeyPairs {
        fmt.Printf("%s: %s\n", *pair.KeyName, *pair.KeyFingerprint)
    }
```

The routine uses the following utility function.

```

func exitErrorf(msg string, args ...interface{}) {
    fmt.Fprintf(os.Stderr, msg+"\n", args...)
```

```
    os.Exit(1)
```

See the complete example on GitHub.

Create a Key Pair

Create a new Go file named `ec2_create_keypair.go`.

Import the required AWS SDK for Go packages.

```
package main

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

    "github.com/aws/aws-sdk-go/aws"
    "github.com/aws/aws-sdk-go/aws/awserr"
    "github.com/aws/aws-sdk-go/aws/session"
    "github.com/aws/aws-sdk-go/service/ec2"
)
```

Get the key pair name passed in to the code and, to access Amazon EC2, create an EC2 client.

```
func main() {
    if len(os.Args) != 2 {
        exitErrorf("pair name required\nUsage: %s key_pair_name",
            filepath.Base(os.Args[0]))
    }
    pairName := os.Args[1]
    sess, err := session.NewSession(&aws.Config{
        Region: aws.String("us-west-2")},
    )

    // Create an EC2 service client.
    svc := ec2.New(sess)
```

Create a new key pair with the provided name.

```
    result, err := svc.CreateKeyPair(&ec2.CreateKeyPairInput{
        KeyName: aws.String(pairName),
```

```
    })
    if err != nil {
        if aerr, ok := err.(awserr.Error); ok && aerr.Code() ==
 "InvalidKeyPair.Duplicate" {
            exitErrorf("Keypair %q already exists.", pairName)
        }
        exitErrorf("Unable to create key pair: %s, %v.", pairName, err)
    }

    fmt.Printf("Created key pair %q %s\n%s\n",
        *result.KeyName, *result.KeyFingerprint,
        *result.KeyMaterial)
}
```

The routine uses the following utility function.

```
func exitErrorf(msg string, args ...interface{}) {
    fmt.Fprintf(os.Stderr, msg+"\n", args...)
    os.Exit(1)
}
```

See the complete example on GitHub.

Delete a Key Pair

Create a new Go file named `ec2_delete_keypair.go`.

Import the required AWS SDK for Go packages.

```
package main

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

    "github.com/aws/aws-sdk-go/aws"
    "github.com/aws/aws-sdk-go/aws/awserr"
    "github.com/aws/aws-sdk-go/aws/session"
    "github.com/aws/aws-sdk-go/service/ec2"
)
```

Get the key pair name passed in to the code and, to access Amazon EC2, create an EC2 client.

```
func main() {
    if len(os.Args) != 2 {
        exitErrorf("pair name required\nUsage: %s key_pair_name",
            filepath.Base(os.Args[0]))
    }
    pairName := os.Args[1]

    sess, err := session.NewSession(&aws.Config{
        Region: aws.String("us-west-2")},
    )

    // Create an EC2 service client.
    svc := ec2.New(sess)
```

Delete the key pair with the provided name.

```
    _, err = svc.DeleteKeyPair(&ec2.DeleteKeyPairInput{
        KeyName: aws.String(pairName),
    })
    if err != nil {
        if aerr, ok := err.(awserr.Error); ok && aerr.Code() ==
 "InvalidKeyPair.Duplicate" {
            exitErrorf("Key pair %q does not exist.", pairName)
        }
        exitErrorf("Unable to delete key pair: %s, %v.", pairName, err)
    }

    fmt.Printf("Successfully deleted %q key pair\n", pairName)
}
```

The routine uses the following utility function.

```
func exitErrorf(msg string, args ...interface{}) {
    fmt.Fprintf(os.Stderr, msg+"\n", args...)
    os.Exit(1)
}
```

See the complete example on GitHub.

Using Regions and Availability Zones with Amazon EC2

These Go examples show you how to retrieve details about AWS Regions and Availability Zones.

The code in this example uses the AWS SDK for Go to perform these tasks by using these methods of the Amazon EC2 client class:

- DescribeRegions
- DescribeAvailabilityZones

You can download complete versions of these example files from the aws-doc-sdk-examples repository on GitHub.

Scenario

Amazon EC2 is hosted in multiple locations worldwide. These locations are composed of AWS Regions and Availability Zones. Each region is a separate geographic area with multiple, isolated locations known as Availability Zones. Amazon EC2 provides the ability to place instances and data in these multiple locations.

In this example, you use Go code to retrieve details about regions and Availability Zones. The code uses the AWS SDK for Go tomanage instances by using the following methods of the Amazon EC2 client class:

- DescribeAvailabilityZones
- DescribeRegions

Prerequisites

- You have set up and configured the AWS SDK for Go.
- You are familiar with AWS Regions and Availability Zones. To learn more, see Regions and Availability Zones in the Amazon EC2 User Guide or Regions and Availability Zones in the Amazon EC2 User Guide.

List the Regions

This example lists the regions in which Amazon EC2 is available.

To get started, create a new Go file named `regions_and_availability.go`.

You must import the relevant Go and AWS SDK for Go packages by adding the following lines.

```
package main

import (
    "fmt"

    "github.com/aws/aws-sdk-go/aws/session"
    "github.com/aws/aws-sdk-go/service/ec2"
)
```

In the `main` function, create a session with credentials from the shared credentials file, `~/.aws/credentials`, and create a new EC2 client.

```
func main() {
    // Load session from shared config
    sess := session.Must(session.NewSessionWithOptions(session.Options{
        SharedConfigState: session.SharedConfigEnable,
    }))

    // Create new EC2 client
    svc := ec2.New(sess)
```

Print out the list of regions that work with Amazon EC2 that are returned by calling `DescribeRegions`.

```
    resultRegions, err := svc.DescribeRegions(nil)
    if err != nil {
        fmt.Println("Error", err)
        return
    }
```

Add a call that retrieves Availability Zones only for the region of the EC2 service object.

```
    resultAvalZones, err := svc.DescribeAvailabilityZones(nil)
    if err != nil {
        fmt.Println("Error", err)
        return
    }

    fmt.Println("Success", resultAvalZones.AvailabilityZones)
}
```

See the complete example on GitHub.

Working with Security Groups in Amazon EC2

These Go examples show you how to:

- Retrieve information about your security groups
- Create a security group to access an Amazon EC2 instance
- Delete an existing security group

You can download complete versions of these example files from the aws-doc-sdk-examples repository on GitHub.

Scenario

An Amazon EC2 security group acts as a virtual firewall that controls the traffic for one or more instances. You add rules to each security group to allow traffic to or from its associated instances. You can modify the rules for a security group at any time; the new rules are automatically applied to all instances that are associated with the security group.

The code in this example uses the AWS SDK for Go to perform these tasks by using these methods of the Amazon EC2 client class:

- DescribeSecurityGroups
- AuthorizeSecurityGroupIngress
- CreateSecurityGroup
- DescribeVpcs
- DeleteSecurityGroup

Prerequisites

- You have set up and configured the AWS SDK for Go.
- You are familiar with Amazon EC2 security groups. To learn more, see Amazon EC2 Amazon Security Groups for Linux Instances in the Amazon EC2 User Guide or Amazon EC2 Amazon Security Groups for Windows Instances in the Amazon EC2 User Guide.

Describing Your Security Groups

This example describes the security groups by IDs that are passed into the routine. It takes a space separated list of group IDs as input.

To get started, create a new Go file named `ec2_describe_security_groups.go`.

You must import the relevant Go and AWS SDK for Go packages by adding the following lines.

```
import (
    "fmt"
    "os"
    "path/filepath"

    "github.com/aws/aws-sdk-go/aws"
    "github.com/aws/aws-sdk-go/aws/awserr"
    "github.com/aws/aws-sdk-go/aws/session"
    "github.com/aws/aws-sdk-go/service/ec2"
)
```

In the `main` function, get the security group ID that is passed in.

```
if len(os.Args) < 2 {
    exitErrorf("Security Group ID required\nUsage: %s group_id ...",
        filepath.Base(os.Args[0]))
}
groupIds := os.Args[1:]
```

Initialize a session and create an EC2 service client.

```
sess, err := session.NewSession(&aws.Config{
    Region: aws.String("us-west-2")},
)

// Create an EC2 service client.
svc := ec2.New(sess)
```

Obtain and print out the security group descriptions. You will explicitly check for errors caused by an invalid group ID.

```
result, err := svc.DescribeSecurityGroups(&ec2.DescribeSecurityGroupsInput{
    GroupIds: aws.StringSlice(groupIds),
```

```
})
if err != nil {
    if aerr, ok := err.(awserr.Error); ok {
        switch aerr.Code() {
        case "InvalidGroupId.Malformed":
            fallthrough
        case "InvalidGroup.NotFound":
            exitErrorf("%s.", aerr.Message())
        }
    }
    exitErrorf("Unable to get descriptions for security groups, %v", err)
}

fmt.Println("Security Group:")
for _, group := range result.SecurityGroups {
    fmt.Println(group)
}
```

The following utility function is used by this example to display errors.

```
func exitErrorf(msg string, args ...interface{}) {
    fmt.Fprintf(os.Stderr, msg+"\n", args...)
    os.Exit(1)
}
```

See the complete example on GitHub.

Creating a Security Group

You can create new Amazon EC2 security groups. To do this, you use the CreateSecurityGroup method.

This example creates a new security group with the given name and description for access to open ports 80 and 22. If a VPC ID is not provided, it associates the security group with the first VPC in the account.

You must import the relevant Go and AWS SDK for Go packages by adding the following lines.

```
import (
    "flag"
    "fmt"
    "os"
```

```
    "github.com/aws/aws-sdk-go/aws"
    "github.com/aws/aws-sdk-go/aws/awserr"
    "github.com/aws/aws-sdk-go/aws/session"
    "github.com/aws/aws-sdk-go/service/ec2"
)
```

Get the parameters (name, description, and optional ID of the VPC) that are passed in to the routine.

```
namePtr := flag.String("n", "", "Group Name")
descPtr := flag.String("d", "", "Group Description")
vpcIDPtr := flag.String("vpc", "", "(Optional) VPC ID to associate security group
 with")

flag.Parse()

if *namePtr == "" || *descPtr == "" {
    flag.PrintDefaults()
    exitErrorf("Group name and description require")
}
```

Create a session and Amazon EC2 client.

```
sess := session.Must(session.NewSessionWithOptions(session.Options{
    SharedConfigState: session.SharedConfigEnable,
}))

svc := ec2.New(sess)
```

Create an Amazon EC2 client. If the VPC ID was not provided, retrieve the first one in the account.

```
if *vpcIDPtr == "" {
    // Get a list of VPCs so we can associate the group with the first VPC.
    result, err := svc.DescribeVpcs(nil)
    if err != nil {
        exitErrorf("Unable to describe VPCs, %v", err)
    }
    if len(result.Vpcs) == 0 {
        exitErrorf("No VPCs found to associate security group with.")
    }

    *vpcIDPtr = aws.StringValue(result.Vpcs[0].VpcId)
```

```
}
```

Create the security group with the VPC ID, name, and description.

```
createRes, err := svc.CreateSecurityGroup(&ec2.CreateSecurityGroupInput{
    GroupName:   aws.String(*namePtr),
    Description: aws.String(*descPtr),
    VpcId:       aws.String(*vpcIDPtr),
})
if err != nil {
    if aerr, ok := err.(awserr.Error); ok {
        switch aerr.Code() {
        case "InvalidVpcID.NotFound":
            exitErrorf("Unable to find VPC with ID %q.", *vpcIDPtr)
        case "InvalidGroup.Duplicate":
            exitErrorf("Security group %q already exists.", *namePtr)
        }
    }
    exitErrorf("Unable to create security group %q, %v", *namePtr, err)
}

fmt.Printf("Created security group %s with VPC %s.\n",
    aws.StringValue(createRes.GroupId), *vpcIDPtr)
```

Add permissions to the security group.

```
_, err = svc.AuthorizeSecurityGroupIngress(&ec2.AuthorizeSecurityGroupIngressInput{
    GroupName: aws.String(*namePtr),
    IpPermissions: []*ec2.IpPermission{
        // Can use setters to simplify seting multiple values without the
        // needing to use aws.String or associated helper utilities.
        (&ec2.IpPermission{}).
            SetIpProtocol("tcp").
            SetFromPort(80).
            SetToPort(80).
            SetIpRanges([]*ec2.IpRange{
                {CidrIp: aws.String("0.0.0.0/0")},
            }),
        (&ec2.IpPermission{}).
            SetIpProtocol("tcp").
            SetFromPort(22).
            SetToPort(22).
            SetIpRanges([]*ec2.IpRange{
```

```
                (&ec2.IpRange{}).
                    SetCidrIp("0.0.0.0/0"),
            }),
    },
})
if err != nil {
    exitErrorf("Unable to set security group %q ingress, %v", *namePtr, err)
}

fmt.Println("Successfully set security group ingress")
```

The following utility function is used by this example.

```
func exitErrorf(msg string, args ...interface{}) {
    fmt.Fprintf(os.Stderr, msg+"\n", args...)
    os.Exit(1)
}
```

See the complete example on GitHub.

Deleting a Security Group

You can delete an Amazon EC2 security group in code. To do this, you use the DeleteSecurityGroup method.

This example deletes a security group with the given group ID.

You must import the relevant Go and AWS SDK for Go packages by adding the following lines.

```
import (
    "fmt"
    "os"
    "path/filepath"

    "github.com/aws/aws-sdk-go/aws"
    "github.com/aws/aws-sdk-go/aws/awserr"
    "github.com/aws/aws-sdk-go/aws/session"
    "github.com/aws/aws-sdk-go/service/ec2"
)
```

Get the group ID that is passed in to the routine.

```
if len(os.Args) != 2 {
```

```
    exitErrorf("Security Group ID required\nUsage: %s group_id",
        filepath.Base(os.Args[0]))
}
groupID := os.Args[1]
```

Create a session.

```
sess, err := session.NewSession(&aws.Config{
    Region: aws.String("us-west-2")},
)

// Create an EC2 service client.
svc := ec2.New(sess)
```

Then delete the security group with the group ID that is passed in.

```
_, err = svc.DeleteSecurityGroup(&ec2.DeleteSecurityGroupInput{
    GroupId: aws.String(groupID),
})
if err != nil {
    if aerr, ok := err.(awserr.Error); ok {
        switch aerr.Code() {
        case "InvalidGroupId.Malformed":
            fallthrough
        case "InvalidGroup.NotFound":
            exitErrorf("%s.", aerr.Message())
        }
    }
    exitErrorf("Unable to get descriptions for security groups, %v.", err)
}

fmt.Printf("Successfully delete security group %q.\n", groupID)
```

This example uses the following utility function.

```
func exitErrorf(msg string, args ...interface{}) {
    fmt.Fprintf(os.Stderr, msg+"\n", args...)
    os.Exit(1)
}
```

See the complete example on GitHub.

Using Elastic IP Addresses in Amazon EC2

These Go examples show you how to:

- Describe Amazon EC2 instance IP addresses
- Allocate addresses to Amazon EC2 instances
- Release Amazon EC2 instance IP addresses

You can download complete versions of these example files from the aws-doc-sdk-examples repository on GitHub.

Scenario

An Elastic IP address is a static IP address designed for dynamic cloud computing that is associated with your AWS account. It is a public IP address, reachable from the Internet. If your instance doesn't have a public IP address, you can associate an Elastic IP address with the instance to enable communication with the Internet.

In this example, you use a series of Go routines to perform several Amazon EC2 operations involving Elastic IP addresses. The routines use the AWS SDK for Go to manage Elastic IP addresses by using these methods of the Amazon EC2 client class:

- DescribeAddresses
- AllocateAddress
- AssociateAddress
- ReleaseAddress

Prerequisites

- You have set up and configured the AWS SDK for Go.
- You are familiar with Elastic IP addresses in Amazon EC2. To learn more, see Elastic IP Addresses in the Amazon EC2 User Guide or Elastic IP Addresses in the Amazon EC2 User Guide.

Describe Instance IP Addresses

Create a new Go file named `ec2_describe_addresses.go`.

You must import the relevant Go and AWS SDK for Go packages by adding the following lines.

```
package main

import (
    "fmt"
    "os"

    "github.com/aws/aws-sdk-go/aws"
    "github.com/aws/aws-sdk-go/aws/session"
    "github.com/aws/aws-sdk-go/service/ec2"
)
```

Get the Address Descriptions

This routine prints out the Elastic IP Addresses for the account's VPC. Initialize a session that the SDK will use to load credentials from the shared credentials file, `~/.aws/credentials`, and create a new EC2 service client.

```
func main() {
    sess, err := session.NewSession(&aws.Config{
        Region: aws.String("us-west-2")},
    )

    // Create an EC2 service client.
    svc := ec2.New(sess)
```

Make the API request to EC2 filtering for the addresses in the account's VPC.

```
    result, err := svc.DescribeAddresses(&ec2.DescribeAddressesInput{
        Filters: []*ec2.Filter{
            {
                Name:   aws.String("domain"),
                Values: aws.StringSlice([]string{"vpc"}),
            },
        },
    })
    if err != nil {
        exitErrorf("Unable to elastic IP address, %v", err)
    }

    // Printout the IP addresses if there are any.
```

```
    if len(result.Addresses) == 0 {
        fmt.Printf("No elastic IPs for %s region\n", *svc.Config.Region)
    } else {
        fmt.Println("Elastic IPs")
        for _, addr := range result.Addresses {
            fmt.Println("*", fmtAddress(addr))
        }
    }
}
```

The `fmtAddress` and `exitErrorf` functions are utility functions used in the example.

```
func fmtAddress(addr *ec2.Address) string {
    out := fmt.Sprintf("IP: %s,  allocation id: %s",
        aws.StringValue(addr.PublicIp), aws.StringValue(addr.AllocationId))
    if addr.InstanceId != nil {
        out += fmt.Sprintf(", instance-id: %s", *addr.InstanceId)
    }
    return out
}

func exitErrorf(msg string, args ...interface{}) {
    fmt.Fprintf(os.Stderr, msg+"\n", args...)
    os.Exit(1)
}
```

See the complete example on GitHub.

Allocate Addresses to Instances

Create a new Go file named `ec2_allocate_address.go`.

You must import the relevant Go and AWS SDK for Go packages by adding the following lines.

```
package main

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

    "github.com/aws/aws-sdk-go/aws"
```

```
    "github.com/aws/aws-sdk-go/aws/session"
    "github.com/aws/aws-sdk-go/service/ec2"
)
```

This routine attempts to allocate a VPC Elastic IP address for the current region. The IP address requires and will be associated with the instance ID that is passed in.

```
func main() {
    if len(os.Args) != 2 {
        exitErrorf("instance ID required\nUsage: %s instance_id",
            filepath.Base(os.Args[0]))
    }
    instanceID := os.Args[1]
```

You will need to initialize a session that the SDK will use to load credentials from the shared credentials file, `~/.aws/credentials`, and create a new Amazon EC2 service client.

```
    sess, err := session.NewSession(&aws.Config{
        Region: aws.String("us-west-2")},
    )

    // Create an EC2 service client.
    svc := ec2.New(sess)
```

Call AllocateAddress, passing in “vpc” as the Domain value.

```
    allocRes, err := svc.AllocateAddress(&ec2.AllocateAddressInput{
        Domain: aws.String("vpc"),
    })
    if err != nil {
        exitErrorf("Unable to allocate IP address, %v", err)
    }
```

Call AssociateAddress to associate the new Elastic IP address with an existing Amazon EC2 instance, and print out the results.

```
    assocRes, err := svc.AssociateAddress(&ec2.AssociateAddressInput{
        AllocationId: allocRes.AllocationId,
        InstanceId:   aws.String(instanceID),
    })
    if err != nil {
```

```
        exitErrorf("Unable to associate IP address with %s, %v",
            instanceID, err)
    }

    fmt.Printf("Successfully allocated %s with instance %s.\n\tallocation id: %s,
 association id: %s\n",
        *allocRes.PublicIp, instanceID, *allocRes.AllocationId,
 *assocRes.AssociationId)
}
```

This example also uses the `exitErrorf` utility function.

```
func exitErrorf(msg string, args ...interface{}) {
    fmt.Fprintf(os.Stderr, msg+"\n", args...)
    os.Exit(1)
}
```

See the complete example on GitHub.

Release Instance IP Addresses

This routine releases an Elastic IP address allocation ID. If the address is associated with an Amazon EC2 instance, the association is removed.

Create a new Go file named `ec2_release_address.go`.

You must import the relevant Go and AWS SDK for Go packages by adding the following lines.

```
package main

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

    "github.com/aws/aws-sdk-go/aws"
    "github.com/aws/aws-sdk-go/aws/awserr"
    "github.com/aws/aws-sdk-go/aws/session"
    "github.com/aws/aws-sdk-go/service/ec2"
)
```

The routine requires that the user pass in the allocation ID of the Elastic IP address.

```
func main() {
    if len(os.Args) != 2 {
        exitErrorf("allocation ID required\nUsage: %s allocation_id",
            filepath.Base(os.Args[0]))
    }
    allocationID := os.Args[1]
```

Initialize a session that the SDK will use to load credentials from the shared credentials file, `~/.aws/credentials`, and create a new EC2 service client.

```
    sess, err := session.NewSession(&aws.Config{
        Region: aws.String("us-west-2")},
    )

    // Create an EC2 service client.
    svc := ec2.New(sess)
```

Attempt to release the Elastic IP address by using the allocation ID.

```
    _, err = svc.ReleaseAddress(&ec2.ReleaseAddressInput{
        AllocationId: aws.String(allocationID),
    })
    if err != nil {
        if aerr, ok := err.(awserr.Error); ok && aerr.Code() ==
 "InvalidAllocationID.NotFound" {
            exitErrorf("Allocation ID %s does not exist", allocationID)
        }
        exitErrorf("Unable to release IP address for allocation %s, %v",
            allocationID, err)
    }

    fmt.Printf("Successfully released allocation ID %s\n", allocationID)
}
```

This example uses the `fmtAddress` and `exitErrorf` utility functions.

```
func fmtAddress(addr *ec2.Address) string {
    out := fmt.Sprintf("IP: %s,  allocation id: %s",
        aws.StringValue(addr.PublicIp), aws.StringValue(addr.AllocationId))
    if addr.InstanceId != nil {
        out += fmt.Sprintf(", instance-id: %s", *addr.InstanceId)
```

```
    }
    return out
}

func exitErrorf(msg string, args ...interface{}) {
    fmt.Fprintf(os.Stderr, msg+"\n", args...)
    os.Exit(1)
}
```

See the complete example on GitHub.

Amazon S3 Glacier Examples Using the AWS SDK for Go

Amazon S3 Glacier is a secure, durable, and extremely low-cost cloud storage service for data archiving and long-term backup. The AWS SDK for Go examples can integrate Amazon S3 Glacier into your applications. The examples assume you have already set up and configured the SDK (that is, you've imported all required packages and set your credentials and region). For more information, see Getting Started with the AWS SDK for Go and Configuring the AWS SDK for Go.

You can download complete versions of these example files from the aws-doc-sdk-examples repository on GitHub.

The Scenario

Amazon S3 Glacier is a secure cloud storage service for data archiving and long-term backup. The service is optimized for infrequently accessed data where a retrieval time of several hours is suitable. These examples show you how to create a vault and upload an archive with Go. The methods used include:

- CreateVault
- UploadArchive

Prerequisites

- You have set up and configured the AWS SDK for Go.
- You are familiar with the Amazon S3 Glacier data model. To learn more, see Amazon Glacier Data Model in the Amazon S3 Glacier Developer Guide.

Create a Vault

The following example uses the Amazon S3 GlacierCreateVault operation to create a vault named YOUR_VAULT_NAME.

```
import (
    "log"

    "github.com/aws/aws-sdk-go/aws"
    "github.com/aws/aws-sdk-go/aws/session"
    "github.com/aws/aws-sdk-go/service/glacier"
)

func main() {
    // Initialize a session that the SDK uses to load
    // credentials from the shared credentials file ~/.aws/credentials
    // and configuration from the shared configuration file ~/.aws/config.
    sess := session.Must(session.NewSessionWithOptions(session.Options{
        SharedConfigState: session.SharedConfigEnable,
    }))

    // Create Glacier client in default region
    svc := glacier.New(sess)

    // start snippet
    _, err := svc.CreateVault(&glacier.CreateVaultInput{
        VaultName: aws.String("YOUR_VAULT_NAME"),
    })
    if err != nil {
        log.Println(err)
        return
    }

    log.Println("Created vault!")
    // end snippet
}
```

Upload an Archive

The following example assumes you have a vault named YOUR_VAULT_NAME. It uses the Amazon S3 GlacierUploadArchive operation to upload a single reader object as an entire archive. The AWS SDK for Go automatically computes the tree hash checksum for the data to be uploaded.

```
import (
    "bytes"
    "log"

    "github.com/aws/aws-sdk-go/aws/session"
    "github.com/aws/aws-sdk-go/service/glacier"
)

func main() {
    // Initialize a session that the SDK uses to load
    // credentials from the shared credentials file ~/.aws/credentials
    // and configuration from the shared configuration file ~/.aws/config.
    sess := session.Must(session.NewSessionWithOptions(session.Options{
        SharedConfigState: session.SharedConfigEnable,
    }))

    // Create Glacier client in default region
    svc := glacier.New(sess)

    // start snippet
    vaultName := "YOUR_VAULT_NAME"

    result, err := svc.UploadArchive(&glacier.UploadArchiveInput{
        VaultName: &vaultName,
        Body:      bytes.NewReader(make([]byte, 2*1024*1024)), // 2 MB buffer
    })
    if err != nil {
        log.Println("Error uploading archive.", err)
        return
    }

    log.Println("Uploaded to archive", *result.ArchiveId)
    // end snippet
}
```

IAM Examples Using the AWS SDK for Go

AWS Identity and Access Management (IAM) is a web service that enables AWS customers to manage users and user permissions in AWS. The service is targeted at organizations with multiple users or systems in the cloud that use AWS products. With IAM, you can centrally manage users, security credentials such as access keys, and permissions that control which AWS resources users can access.

The examples assume you have already set up and configured the SDK (that is, you've imported all required packages and set your credentials and region). For more information, see Getting Started with the AWS SDK for Go and Configuring the AWS SDK for Go.

You can download complete versions of these example files from the aws-doc-sdk-examples repository on GitHub.

Topics

- Managing IAM Users
- Managing IAM Access Keys
- Managing IAM Account Aliases
- Working with IAM Policies
- Working with IAM Server Certificates

Managing IAM Users

This Go example shows you how to create, update, view, and delete IAM users. You can download complete versions of these example files from the aws-doc-sdk-examples repository on GitHub.

Scenario

In this example, you use a series of Go routines to manage users in IAM. The routines use the AWS SDK for Go IAM client methods that follow:

- CreateUser
- ListUsers
- UpdateUser
- GetUser
- DeleteUser

- GetAccountAuthorizationDetails

Prerequisites

- You have set up and configured the AWS SDK for Go.
- You are familiar with IAM users. To learn more, see IAM Users in the IAM User Guide.

Create a New IAM User

This code creates a new IAM user.

Create a new Go file named `iam_createuser.go`. You must import the relevant Go and AWS SDK for Go packages by adding the following lines.

```
import (
    "fmt"
    "os"

    "github.com/aws/aws-sdk-go/aws"
    "github.com/aws/aws-sdk-go/aws/awserr"
    "github.com/aws/aws-sdk-go/aws/session"
    "github.com/aws/aws-sdk-go/service/iam"
)
```

The code takes the new user name as an argument, and then calls `GetUser` with the user name.

```
// Initialize a session in us-west-2 that the SDK will use to load
// credentials from the shared credentials file ~/.aws/credentials.
sess, err := session.NewSession(&aws.Config{
    Region: aws.String("us-west-2")},
)

// Create a IAM service client.
svc := iam.New(sess)

_, err = svc.GetUser(&iam.GetUserInput{
    UserName: &os.Args[1],
})
```

If you receive a `NoSuchEntity` error, call `CreateUser` because the user doesn't exist.

```
if awserr, ok := err.(awserr.Error); ok && awserr.Code() ==
 iam.ErrCodeNoSuchEntityException {
    result, err := svc.CreateUser(&iam.CreateUserInput{
        UserName: &os.Args[1],
    })

    if err != nil {
        fmt.Println("CreateUser Error", err)
        return
    }

    fmt.Println("Success", result)
} else {
    fmt.Println("GetUser Error", err)
}
```

See the complete example on GitHub.

List IAM Users in Your Account

You can get a list of the users in your account and print the list to the console.

Create a new Go file named `iam_listusers.go`. You must import the relevant Go and AWS SDK for Go packages by adding the following lines.

```
import (
    "fmt"

    "github.com/aws/aws-sdk-go/aws"
    "github.com/aws/aws-sdk-go/aws/session"
    "github.com/aws/aws-sdk-go/service/iam"
)
```

Set up a new IAM client.

```
// Initialize a session in us-west-2 that the SDK will use to load
// credentials from the shared credentials file ~/.aws/credentials.
sess, err := session.NewSession(&aws.Config{
    Region: aws.String("us-west-2")},
)

// Create a IAM service client.
```

```
svc := iam.New(sess)
```

Call `ListUsers` and print the results.

```
result, err := svc.ListUsers(&iam.ListUsersInput{
    MaxItems: aws.Int64(10),
})

if err != nil {
    fmt.Println("Error", err)
    return
}

for i, user := range result.Users {
    if user == nil {
        continue
    }
    fmt.Printf("%d user %s created %v\n", i, *user.UserName, user.CreateDate)
}
```

See the complete example on GitHub.

Update a User's Name

In this example, you change the name of an IAM user to a new value.

Create a new Go file named `iam_updateuser.go`. You must import the relevant Go and AWS SDK for Go packages by adding the following lines.

```
import (
    "fmt"
    "os"

    "github.com/aws/aws-sdk-go/aws"
    "github.com/aws/aws-sdk-go/aws/session"
    "github.com/aws/aws-sdk-go/service/iam"
)
```

Set up a new IAM client.

```
// Initialize a session in us-west-2 that the SDK will use to load
```

```
// credentials from the shared credentials file ~/.aws/credentials.
sess, err := session.NewSession(&aws.Config{
    Region: aws.String("us-west-2")},
)

// Create a IAM service client.
svc := iam.New(sess)
```

Call `UpdateUser`, passing in the original user name and the new name, and print the results.

```
result, err := svc.UpdateUser(&iam.UpdateUserInput{
    UserName:    &os.Args[1],
    NewUserName: &os.Args[2],
})

if err != nil {
    fmt.Println("Error", err)
    return
}

fmt.Println("Success", result)
```

See the complete example on GitHub.

Delete an IAM User

In this example, you delete an IAM user.

Create a new Go file named `iam_deleteuser.go`. You must import the relevant Go and AWS SDK for Go packages by adding the following lines.

```
import (
    "fmt"
    "os"

    "github.com/aws/aws-sdk-go/aws"
    "github.com/aws/aws-sdk-go/aws/awserr"
    "github.com/aws/aws-sdk-go/aws/session"
    "github.com/aws/aws-sdk-go/service/iam"
)
```

Set up a new IAM client.

```
    sess, err := session.NewSession(&aws.Config{
        Region: aws.String("us-west-2")},
    )
    svc := iam.New(sess)
```

Call `DeleteUser`, passing in the user name, and print the results. If the user doesn't exist, display an error.

```
    _, err = svc.DeleteUser(&iam.DeleteUserInput{
        UserName: &os.Args[1],
    })

    // If the user does not exist than we will log an error.
    if awserr, ok := err.(awserr.Error); ok && awserr.Code() ==
iam.ErrCodeNoSuchEntityException {
        fmt.Printf("User %s does not exist\n", os.Args[1])
        return
    } else if err != nil {
        fmt.Println("Error", err)
        return
    }

    fmt.Printf("User %s has been deleted\n", os.Args[1])
```

See the complete example on GitHub.

List the IAM Users who have Administrator Privileges

In this example, you list the IAM users who have administrator privileges (a policy or attached policy of the user or a group to which the user belongs has the name **AdministratorAccess**.

Create a new Go file named `IamGetAdmins.go`. Import the following packages.

```
import (
    "github.com/aws/aws-sdk-go/aws"
    "github.com/aws/aws-sdk-go/aws/session"
    "github.com/aws/aws-sdk-go/service/iam"

    "fmt"
    "os"
)
```

Create a method to determine whether a user has a policy that has administrator privileges.

```
func UserPolicyHasAdmin(user *iam.UserDetail, admin string) bool {
    for _, policy := range user.UserPolicyList {
        if *policy.PolicyName == admin {
            return true
        }
    }

    return false
}
```

Create a method to determine whether a user has an attached policy that has administrator privileges.

```
func AttachedUserPolicyHasAdmin(user *iam.UserDetail, admin string) bool {
    for _, policy := range user.AttachedManagedPolicies {
        if *policy.PolicyName == admin {
            return true
        }
    }

    return false
}
```

Create a method that determines whether a group has a policy that has adminstrator privileges.

```
func GroupPolicyHasAdmin(svc *iam.IAM, group *iam.Group, admin string) bool {
    input := &iam.ListGroupPoliciesInput{
        GroupName: group.GroupName,
    }

    result, err := svc.ListGroupPolicies(input)
    if err != nil {
        fmt.Println("Got error calling ListGroupPolicies for group", group.GroupName)
    }

    // Wade through policies
    for _, policyName := range result.PolicyNames {
        if
        *policyName == admin {
            return true
```

```
        }
    }

    return false
}
```

Create a method that determines whether a group has an attached policy that has administrator privileges.

```
func AttachedGroupPolicyHasAdmin(svc *iam.IAM, group *iam.Group, admin string) bool {
    input := &iam.ListAttachedGroupPoliciesInput{GroupName: group.GroupName}
    result, err := svc.ListAttachedGroupPolicies(input)
    if err != nil {
        fmt.Println("Got error getting attached group policies:")
        fmt.Println(err.Error())
        os.Exit(1)
    }

    for _, policy := range result.AttachedPolicies {
        if *policy.PolicyName == admin {
            return true
        }
    }

    return false
}
```

Create a method that determines whether any group to which the user belongs has administrator privileges.

```
func UsersGroupsHaveAdmin(svc *iam.IAM, user *iam.UserDetail, admin string) bool {
    input := &iam.ListGroupsForUserInput{UserName: user.UserName}
    result, err := svc.ListGroupsForUser(input)
    if err != nil {
        fmt.Println("Got error getting groups for user:")
        fmt.Println(err.Error())
        os.Exit(1)
    }

    for _, group := range result.Groups {
        groupPolicyHasAdmin := GroupPolicyHasAdmin(svc, group, admin)
```

```
        if groupPolicyHasAdmin {
            return true
        }

        attachedGroupPolicyHasAdmin := AttachedGroupPolicyHasAdmin(svc, group, admin)

        if attachedGroupPolicyHasAdmin {
            return true
        }
    }

    return false
}
```

Create a method that determines whether a user has administrator privileges.

```
func IsUserAdmin(svc *iam.IAM, user *iam.UserDetail, admin string) bool {
    // Check policy, attached policy, and groups (policy and attached policy)
    policyHasAdmin := UserPolicyHasAdmin(user, admin)
    if policyHasAdmin {
        return true
    }

    attachedPolicyHasAdmin := AttachedUserPolicyHasAdmin(user, admin)
    if attachedPolicyHasAdmin {
        return true
    }

    userGroupsHaveAdmin := UsersGroupsHaveAdmin(svc, user, admin)
    if userGroupsHaveAdmin {
        return true
    }

    return false
}
```

Create a main method with an IAM client in `us-west-2`. Create variables to keep track of how many users we have and how many of those have administrator privileges.

```
sess, err := session.NewSession()
if err != nil {
    fmt.Println("Got error creating new session")
    fmt.Println(err.Error())
```

```
    os.Exit(1)
}

svc := iam.New(sess, &aws.Config{Region: aws.String("us-west-2")})

numUsers := 0
numAdmins := 0
```

Create the input for and call `GetAccountAuthorizationDetails`. If there is an error, print an error message and quit.

```
user := "User"
input := &iam.GetAccountAuthorizationDetailsInput{Filter: []*string{&user}}
resp, err := svc.GetAccountAuthorizationDetails(input)
if err != nil {
    fmt.Println("Got error getting account details")
    fmt.Println(err.Error())
    os.Exit(1)
}
```

Loop through the users. If a user has adminstrator privileges, print their name and increment the number of users who have adminstrator privileges.

```
adminName := "AdministratorAccess"

// Wade through resulting users
for _, user := range resp.UserDetailList {
    numUsers += 1

    isAdmin := IsUserAdmin(svc, user, adminName)

    if isAdmin {
        fmt.Println(*user.UserName)
        numAdmins += 1
    }
}
```

If we did not get all of the users in the first call to `GetAccountAuthorizationDetails`, loop through the next set of users and determine which of those have adminstrator privileges.

```
for *resp.IsTruncated {
```

```
    input := &iam.GetAccountAuthorizationDetailsInput{Filter: []*string{&user}, Marker:
 resp.Marker}
    resp, err = svc.GetAccountAuthorizationDetails(input)
    if err != nil {
        fmt.Println("Got error getting account details")
        fmt.Println(err.Error())
        os.Exit(1)
    }

    // Wade through resulting users
    for _, user := range resp.UserDetailList {
        numUsers += 1

        isAdmin := IsUserAdmin(svc, user, adminName)

        if isAdmin {
            fmt.Println(*user.UserName)
            numAdmins += 1
        }
    }
}
```

Finally, display the number of users who have administrator access.

```
fmt.Println("")
fmt.Println("Found", numAdmins, "admin(s) out of", numUsers, "user(s).")
```

See the complete example on GitHub.

Managing IAM Access Keys

This Go example shows you how to create, modify, view, or rotate IAM access keys. You can download complete versions of these example files from the aws-doc-sdk-examples repository on GitHub.

Scenario

Users need their own access keys to make programmatic calls to the AWS SDK for Go. To fill this need, you can create, modify, view, or rotate access keys (access key IDs and secret access keys) for IAM users. By default, when you create an access key its status is Active, which means the user can use the access key for API calls.

In this example, you use a series of Go routines to manage access keys in IAM. The routines use the AWS SDK for Go IAM client methods that follow:

- CreateAccessKey
- ListAccessKeys
- GetAccessKeyLastUsed
- UpdateAccessKey
- DeleteAccessKey

Prerequisites

- You have set up and configured the AWS SDK for Go.
- You are familiar with IAM access keys. To learn more, see Managing Access Keys for IAM Users in the IAM User Guide.

Create a New IAM Access Key

This code creates a new IAM access key for the IAM user named IAM_USER_NAME.

Create a new Go file named `iam_createaccesskey.go`. You must import the relevant Go and AWS SDK for Go packages by adding the following lines.

```
package main

import (
    "fmt"

    "github.com/aws/aws-sdk-go/aws"
    "github.com/aws/aws-sdk-go/aws/session"
    "github.com/aws/aws-sdk-go/service/iam"
)
```

Set up the session.

```
func main() {
    sess, err := session.NewSession(&aws.Config{
        Region: aws.String("us-west-2")},
    )
```

```
    // Create a IAM service client.
    svc := iam.New(sess)
```

Call `CreateAccessKey` and print the results.

```
    result, err := svc.CreateAccessKey(&iam.CreateAccessKeyInput{
        UserName: aws.String("IAM_USER_NAME"),
    })

    if err != nil {
        fmt.Println("Error", err)
        return
    }

    fmt.Println("Success", *result.AccessKey)
}
```

See the complete example on GitHub.

List a User's Access Keys

In this example, you get a list of the access keys for a user and print the list to the console.

Create a new Go file named `iam_listaccesskeys.go`. You must import the relevant Go and AWS SDK for Go packages by adding the following lines.

```
package main

import (
    "fmt"

    "github.com/aws/aws-sdk-go/aws"
    "github.com/aws/aws-sdk-go/aws/session"
    "github.com/aws/aws-sdk-go/service/iam"
)
```

Set up a new IAM client.

```
func main() {
    sess, err := session.NewSession(&aws.Config{
        Region: aws.String("us-west-2")},
```

```
    )

    // Create a IAM service client.
    svc := iam.New(sess)
```

Call `ListAccessKeys` and print the results.

```
    result, err := svc.ListAccessKeys(&iam.ListAccessKeysInput{
        MaxItems: aws.Int64(5),
        UserName: aws.String("IAM_USER_NAME"),
    })

    if err != nil {
        fmt.Println("Error", err)
        return
    }

    fmt.Println("Success", result)
}
```

See the complete example on GitHub.

Get the Last Use for an Access Key

In this example, you find out when an access key was last used.

Create a new Go file named `iam_accesskeylastused.go`. You must import the relevant Go and AWS SDK for Go packages by adding the following lines.

```
package main

import (
    "fmt"

    "github.com/aws/aws-sdk-go/aws"
    "github.com/aws/aws-sdk-go/aws/session"
    "github.com/aws/aws-sdk-go/service/iam"
)
```

Set up a new IAM client.

```
func main() {
```

```
    sess, err := session.NewSession(&aws.Config{
        Region: aws.String("us-west-2")},
    )

    // Create a IAM service client.
    svc := iam.New(sess)
```

Call `GetAccessKeyLastUsed`, passing in the access key ID, and print the results.

```
    result, err := svc.GetAccessKeyLastUsed(&iam.GetAccessKeyLastUsedInput{
        AccessKeyId: aws.String("ACCESS_KEY_ID"),
    })

    if err != nil {
        fmt.Println("Error", err)
        return
    }

    fmt.Println("Success", *result.AccessKeyLastUsed)
}
```

See the complete example on GitHub.

Update Access Key Status

In this example, you delete an IAM user.

Create a new Go file with the name `iam_updateaccesskey.go`. You must import the relevant Go and AWS SDK for Go packages by adding the following lines.

```
package main

import (
    "fmt"

    "github.com/aws/aws-sdk-go/aws"
    "github.com/aws/aws-sdk-go/aws/session"
    "github.com/aws/aws-sdk-go/service/iam"
)
```

Set up a new IAM client.

```
func main() {
    sess, err := session.NewSession(&aws.Config{
        Region: aws.String("us-west-2")},
    )

    // Create a IAM service client.
    svc := iam.New(sess)
```

Call `UpdateAccessKey`, passing in the access key ID, status (making it active in this case), and user name.

```
    _, err = svc.UpdateAccessKey(&iam.UpdateAccessKeyInput{
        AccessKeyId: aws.String("ACCESS_KEY_ID"),
        Status:      aws.String(iam.StatusTypeActive),
        UserName:    aws.String("USER_NAME"),
    })

    if err != nil {
        fmt.Println("Error", err)
        return
    }

    fmt.Println("Access Key updated")
}
```

See the complete example on GitHub.

Delete an Access Key

In this example, you delete an access key.

Create a new Go file named `iam_deleteaccesskey.go`. You must import the relevant Go and AWS SDK for Go packages by adding the following lines.

```
package main

import (
    "fmt"

    "github.com/aws/aws-sdk-go/aws"
    "github.com/aws/aws-sdk-go/aws/session"
```

```
    "github.com/aws/aws-sdk-go/service/iam"
)
```

Set up a new IAM client.

```
func main() {
    sess, err := session.NewSession(&aws.Config{
        Region: aws.String("us-west-2")},
    )

    // Create a IAM service client.
    svc := iam.New(sess)
```

Call `DeleteAccessKey`, passing in the access key ID and user name.

```
    result, err := svc.DeleteAccessKey(&iam.DeleteAccessKeyInput{
        AccessKeyId: aws.String("ACCESS_KEY_ID"),
        UserName:    aws.String("USER_NAME"),
    })

    if err != nil {
        fmt.Println("Error", err)
        return
    }

    fmt.Println("Success", result)
}
```

See the complete example on GitHub.

Managing IAM Account Aliases

This Go example shows you how to create, list, and delete IAM account aliases. You can download complete versions of these example files from the aws-doc-sdk-examples repository on GitHub.

Scenario

You can use a series of Go routines to manage aliases in IAM. The routines use the AWS SDK for Go IAM client methods that follow:

- CreateAccountAlias

- ListAccountAliases
- DeleteAccountAlias

Prerequisites

- You have set up and configured the AWS SDK for Go.
- You are familiar with IAM account aliases. To learn more, see Your AWS account ID and Its Alias in the IAM User Guide.

Create a New IAM Account Alias

This code creates a new IAM user.

Create a new Go file named `iam_createaccountalias.go`. You must import the relevant Go and AWS SDK for Go packages by adding the following lines.

```
package main

import (
    "fmt"
    "os"

    "github.com/aws/aws-sdk-go/aws"
    "github.com/aws/aws-sdk-go/aws/session"
    "github.com/aws/aws-sdk-go/service/iam"
)
```

Set up a session and an IAM client.

```
func main() {
    sess, err := session.NewSession(&aws.Config{
        Region: aws.String("us-west-2")},
    )

    // Create a IAM service client.
    svc := iam.New(sess)
```

The code takes the new alias as an argument, and then calls `CreateAccountAlias` with the alias name.

```
    _, err = svc.CreateAccountAlias(&iam.CreateAccountAliasInput{
        AccountAlias: &os.Args[1],
    })

    if err != nil {
        fmt.Println("Error", err)
        return
    }

    fmt.Printf("Account alias %s has been created\n", os.Args[1])
```

See the complete example on GitHub.

List IAM Account Aliases

This code lists the aliases for your IAM account.

Create a new Go file named `iam_listaccountaliases.go`. You must import the relevant Go and AWS SDK for Go packages by adding the following lines.

```
package main

import (
    "fmt"

    "github.com/aws/aws-sdk-go/aws"
    "github.com/aws/aws-sdk-go/aws/session"
    "github.com/aws/aws-sdk-go/service/iam"
)
```

Set up a session and an IAM client.

```
func main() {
    sess, err := session.NewSession(&aws.Config{
        Region: aws.String("us-west-2")},
    )

    // Create a IAM service client.
    svc := iam.New(sess)
```

The code calls `ListAccountAliases`, specifying to return a maximum of 10 items.

```
    result, err := svc.ListAccountAliases(&iam.ListAccountAliasesInput{
        MaxItems: aws.Int64(10),
    })

    if err != nil {
        fmt.Println("Error", err)
        return
    }

    for i, alias := range result.AccountAliases {
        if alias == nil {
            continue
        }
        fmt.Printf("Alias %d: %s\n", i, *alias)
    }
}
```

See the complete example on GitHub.

Delete an IAM Account Alias

This code deletes a specified IAM account alias.

Create a new Go file with the name `iam_deleteaccountalias.go`. You must import the relevant Go and AWS SDK for Go packages by adding the following lines.

```
package main

import (
    "fmt"
    "os"

    "github.com/aws/aws-sdk-go/aws"
    "github.com/aws/aws-sdk-go/aws/session"
    "github.com/aws/aws-sdk-go/service/iam"
)
```

Set up a session and an IAM client.

```
func main() {
    // Initialize a session in us-west-2 that the SDK will use to load
    // credentials from the shared credentials file ~/.aws/credentials.
```

```
    sess, err := session.NewSession(&aws.Config{
        Region: aws.String("us-west-2")},
    )

    // Create a IAM service client.
    svc := iam.New(sess)
```

The code calls `ListAccountAliases`, specifying to return a maximum of 10 items.

```
    _, err = svc.DeleteAccountAlias(&iam.DeleteAccountAliasInput{
        AccountAlias: &os.Args[1],
    })

    if err != nil {
        fmt.Println("Error", err)
        return
    }

    fmt.Printf("Alias %s has been deleted\n", os.Args[1])
```

See the complete example on GitHub.

Working with IAM Policies

This Go example shows you how to create, get, attach, and detach IAM policies. You can download complete versions of these example files from the aws-doc-sdk-examples repository on GitHub.

Scenario

You grant permissions to a user by creating a policy, which is a document that lists the actions that a user can perform and the resources those actions can affect. Any actions or resources that are not explicitly allowed are denied by default. Policies can be created and attached to users, groups of users, roles assumed by users, and resources.

In this example, you use a series of Go routines to manage policies in IAM. The routines use the AWS SDK for Go IAM client methods that follow:

- CreatePolicy
- GetPolicy
- ListAttachedRolePolicies

- AttachRolePolicy
- DetachRolePolicy

Prerequisites

- You have set up and configured the AWS SDK for Go.
- You are familiar with IAM policies. To learn more, see Overview of Access Management: Permissions and Policies in the IAM User Guide.

Create an IAM Policy

This code creates a new IAM Policy. Create a new Go file named `iam_createpolicy.go`.

You must import the relevant Go and AWS SDK for Go packages by adding the following lines.

```
package main

import (
    "encoding/json"
    "fmt"

    "github.com/aws/aws-sdk-go/aws"
    "github.com/aws/aws-sdk-go/aws/session"
    "github.com/aws/aws-sdk-go/service/iam"
)
```

Define two structs. The first is the definition of the policies to upload to IAM.

```
type PolicyDocument struct {
    Version   string
    Statement []StatementEntry
}
```

The second dictates what this policy will allow or disallow.

```
type StatementEntry struct {
    Effect   string
    Action   []string
    Resource string
```

```
}
```

Set up the session and IAM client.

```
func main() {
    sess, err := session.NewSession(&aws.Config{
        Region: aws.String("us-west-2")},
    )

    // Create a IAM service client.
    svc := iam.New(sess)
```

Build the policy document using the structures defined earlier.

```
    policy := PolicyDocument{
        Version: "2012-10-17",
        Statement: []StatementEntry{
            StatementEntry{
                Effect: "Allow",
                Action: []string{
                    "logs:CreateLogGroup", // Allow for creating log groups
                },
                Resource: "RESOURCE ARN FOR logs:*",
            },
            StatementEntry{
                Effect: "Allow",
                // Allows for DeleteItem, GetItem, PutItem, Scan, and UpdateItem
                Action: []string{
                    "dynamodb:DeleteItem",
                    "dynamodb:GetItem",
                    "dynamodb:PutItem",
                    "dynamodb:Scan",
                    "dynamodb:UpdateItem",
                },
                Resource: "RESOURCE ARN FOR dynamodb:*",
            },
        },
    }
```

Marshal the policy to JSON and pass to `CreatePolicy`.

```
    b, err := json.Marshal(&policy)
```

```
    if err != nil {
        fmt.Println("Error marshaling policy", err)
        return
    }

    result, err := svc.CreatePolicy(&iam.CreatePolicyInput{
        PolicyDocument: aws.String(string(b)),
        PolicyName:     aws.String("myDynamodbPolicy"),
    })

    if err != nil {
        fmt.Println("Error", err)
        return
    }

    fmt.Println("New policy", result)
}
```

See the complete example on GitHub.

Get an IAM Policy

In this example, you retrieve an existing policy from IAM. Create a new Go file named `iam_getpolicy.go`.

You must import the relevant Go and AWS SDK for Go packages by adding the following lines.

```
package main

import (
    "fmt"

    "github.com/aws/aws-sdk-go/aws"
    "github.com/aws/aws-sdk-go/aws/session"
    "github.com/aws/aws-sdk-go/service/iam"
)
```

Set up a new IAM client.

```
func main() {
    // Initialize a session in us-west-2 that the SDK will use to load
    // credentials from the shared credentials file ~/.aws/credentials.
    sess, err := session.NewSession(&aws.Config{
```

```
        Region: aws.String("us-west-2")},
    )

    // Create a IAM service client.
    svc := iam.New(sess)
```

Call `GetPolicy`, passing in the ARN for the policy (which is hard coded in this example), and print the results.

```
    arn := "arn:aws:iam::aws:policy/AWSLambdaExecute"
    result, err := svc.GetPolicy(&iam.GetPolicyInput{
        PolicyArn: &arn,
    })

    if err != nil {
        fmt.Println("Error", err)
        return
    }

    fmt.Printf("%s - %s\n", arn, *result.Policy.Description)
```

See the complete example on GitHub.

Attach a Managed Role Policy

In this example, you attach an IAM managed role policy. Create a new Go file named `iam_attachuserpolicy.go`. You'll call the `ListAttachedRolePolicies` method of the IAM service object, which returns an array of managed policies.

Then, you'll check the array members to see if the policy you want to attach to the role is already attached. If the policy isn't attached, you'll call the `AttachRolePolicy` method to attach it.

You must import the relevant Go and AWS SDK for Go packages by adding the following lines.

```
package main

import (
    "fmt"
    "os"

    "github.com/aws/aws-sdk-go/aws"
    "github.com/aws/aws-sdk-go/aws/session"
```

```
    "github.com/aws/aws-sdk-go/service/iam"
)
```

Set up a new IAM client.

```
func main() {
    // Initialize a session in us-west-2 that the SDK will use to load
    // credentials from the shared credentials file ~/.aws/credentials.
    sess, err := session.NewSession(&aws.Config{
        Region: aws.String("us-west-2")},
    )

    // Create a IAM service client.
    svc := iam.New(sess)
```

Declare variables to hold the name and ARN of the policy.

```
    var pageErr error
    policyName := "AmazonDynamoDBFullAccess"
    policyArn := "arn:aws:iam::aws:policy/AmazonDynamoDBFullAccess"
```

Paginate through all the role policies. If your role exists on any role policy, you set the `pageErr` and return `false`, stopping the pagination.

```
    err = svc.ListAttachedRolePoliciesPages(
        &iam.ListAttachedRolePoliciesInput{
            RoleName: &os.Args[1],
        },
        func(page *iam.ListAttachedRolePoliciesOutput, lastPage bool) bool {
            if page != nil && len(page.AttachedPolicies) > 0 {
                for _, policy := range page.AttachedPolicies {
                    if *policy.PolicyName == policyName {
                        pageErr = fmt.Errorf("%s is already attached to this role",
 policyName)
                        return false
                    }
                }
                // We should keep paginating because we did not find our role
                return true
            }
            return false
        },
```

```
    )
```

If your role policy is not attached already, call `AttachRolePolicy`.

```
    if pageErr != nil {
        fmt.Println("Error", pageErr)
        return
    }

    if err != nil {
        fmt.Println("Error", err)
        return
    }

    _, err = svc.AttachRolePolicy(&iam.AttachRolePolicyInput{
        PolicyArn: &policyArn,
        RoleName:  &os.Args[1],
    })

    if err != nil {
        fmt.Println("Unable to attach role policy to role")
        return
    }
    fmt.Println("Role attached successfully")
```

See the complete example on GitHub.

Detach a Managed Role Policy

In this example, you detach a role policy. Once again, you call the `ListAttachedRolePolicies` method of the IAM service object, which returns an array of managed policies.

Then, check the array members to see if the policy you want to detach from the role is attached. If the policy is attached, call the `DetachRolePolicy` method to detach it.

Create a new Go file named `iam_detachuserpolicy.go`. You must import the relevant Go and AWS SDK for Go packages by adding the following lines.

```
package main

import (
```

```
    "fmt"
    "os"

    "github.com/aws/aws-sdk-go/aws"
    "github.com/aws/aws-sdk-go/aws/session"
    "github.com/aws/aws-sdk-go/service/iam"
)
```

Set up a new IAM client.

```
func main() {
    // Initialize a session in us-west-2 that the SDK will use to load
    // credentials from the shared credentials file ~/.aws/credentials.
    sess, err := session.NewSession(&aws.Config{
        Region: aws.String("us-west-2")},
    )

    // Create a IAM service client.
    svc := iam.New(sess)
```

Declare variables to hold the name and ARN of the policy.

```
    foundPolicy := false
    policyName := "AmazonDynamoDBFullAccess"
    policyArn := "arn:aws:iam::aws:policy/AmazonDynamoDBFullAccess"
```

Paginate through all the role policies. If the role exists on any role policy, you stop iterating and detach the role.

```
    err = svc.ListAttachedRolePoliciesPages(
        &iam.ListAttachedRolePoliciesInput{
            RoleName: &os.Args[1],
        },
        func(page *iam.ListAttachedRolePoliciesOutput, lastPage bool) bool {
            if page != nil && len(page.AttachedPolicies) > 0 {
                for _, policy := range page.AttachedPolicies {
                    if *policy.PolicyName == policyName {
                        foundPolicy = true
                        return false
                    }
                }
                return true
```

```
            }
            return false
        },
    )

    if err != nil {
        fmt.Println("Error", err)
        return
    }

    if !foundPolicy {
        fmt.Println("Policy was not attached to role")
        return
    }

    _, err = svc.DetachRolePolicy(&iam.DetachRolePolicyInput{
        PolicyArn: &policyArn,
        RoleName:  &os.Args[1],
    })

    if err != nil {
        fmt.Println("Unable to detach role policy to role")
        return
    }
    fmt.Println("Role detached successfully")
```

See the complete example on GitHub.

Working with IAM Server Certificates

This Go example shows you how to carry out basic tasks for managing server certificate HTTPS connections with the AWS SDK for Go.

You can download complete versions of these example files from the aws-doc-sdk-examples repository on GitHub.

Scenario

To enable HTTPS connections to your website or application on AWS, you need an SSL/TLS server certificate. To use a certificate that you obtained from an external provider with your website or application on AWS, you must upload the certificate to IAM or import it into AWS Certificate Manager.

In this example, you use a series of Go routines to manage policies in IAM. The routines use the AWS SDK for GoIAM client methods that follow:

- ListServerCertificates
- GetServerCertificate
- UpdateServerCertificate
- DeleteServerCertificate

Prerequisites

- You have set up and configured the AWS SDK for Go.
- You are familiar with server certificates. To learn more, see Working with Server Certificates in the IAM User Guide.

List Your Server Certificates

This code lists your certificates. Create a new Go file named `iam_listservercerts.go`.

You must import the relevant Go and AWS SDK for Go packages by adding the following lines.

```
import (
    "fmt"

    "github.com/aws/aws-sdk-go/aws"
    "github.com/aws/aws-sdk-go/aws/session"
    "github.com/aws/aws-sdk-go/service/iam"
)
```

Set up the session and IAM client.

```
func main() {
    // Initialize a session in us-west-2 that the SDK will use to load
    // credentials from the shared credentials file ~/.aws/credentials.
    sess, err := session.NewSession(&aws.Config{
        Region: aws.String("us-west-2")},
    )

    // Create a IAM service client.
    svc := iam.New(sess)
```

Call `ListServerCertificates` and print the details.

```
    result, err := svc.ListServerCertificates(nil)
    if err != nil {
        fmt.Println("Error", err)
        return
    }

    for i, metadata := range result.ServerCertificateMetadataList {
        if metadata == nil {
            continue
        }

        fmt.Printf("Metadata %d: %v\n", i, metadata)
    }
```

See the complete example on GitHub.

Get a Server Certificate

In this example, you retrieve an existing server certificate.

Create a new Go file named `iam_getservercert.go`. You must import the relevant Go and AWS SDK for Go packages by adding the following lines.

```
package main

import (
    "fmt"

    "github.com/aws/aws-sdk-go/aws"
    "github.com/aws/aws-sdk-go/aws/session"
    "github.com/aws/aws-sdk-go/service/iam"
)
```

Set up a new IAM client.

```
func main() {
    sess, err := session.NewSession(&aws.Config{
        Region: aws.String("us-west-2")},
    )
```

```
    // Create a IAM service client.
    svc := iam.New(sess)
```

Call `GetServerCertificate`, passing the name of the certificate, and print the results.

```
    result, err := svc.GetServerCertificate(&iam.GetServerCertificateInput{
        ServerCertificateName: aws.String("CERTIFICATE_NAME"),
    })
    if err != nil {
        fmt.Println("Error", err)
        return
    }

    fmt.Println("ServerCertificate:", result)
}
```

See the complete example on GitHub.

Update a Server Certificate

In this example, you update an existing server certificate.

Create a new Go file named `iam_updateservercert.go`. You call the `UpdateServerCertificate` method of the IAM service object to change the name of the certificate.

You must import the relevant Go and AWS SDK for Go packages by adding the following lines.

```
package main

import (
    "fmt"

    "github.com/aws/aws-sdk-go/aws"
    "github.com/aws/aws-sdk-go/aws/session"
    "github.com/aws/aws-sdk-go/service/iam"
)
```

Set up a new IAM client.

```
func main() {
```

```
    sess, err := session.NewSession(&aws.Config{
        Region: aws.String("us-west-2")},
    )

    // Create a IAM service client.
    svc := iam.New(sess)
```

Update the certificate name.

```
    _, err = svc.UpdateServerCertificate(&iam.UpdateServerCertificateInput{
        ServerCertificateName:    aws.String("CERTIFICATE_NAME"),
        NewServerCertificateName: aws.String("NEW_CERTIFICATE_NAME"),
    })
    if err != nil {
        fmt.Println("Error", err)
        return
    }

    fmt.Println("Server certificate updated")
}
```

See the complete example on GitHub.

Delete a Server Certificate

In this example, you delete an existing server certificate.

Create a new Go file named `iam_deleteservercert.go`. You call the `DeleteServerCertificate` method of the IAM service object to change the name of the certificate.

You must import the relevant Go and AWS SDK for Go packages by adding the following lines.

```
package main

import (
    "fmt"

    "github.com/aws/aws-sdk-go/aws"
    "github.com/aws/aws-sdk-go/aws/session"
    "github.com/aws/aws-sdk-go/service/iam"
)
```

Set up a new IAM client.

```
func main() {
    sess, err := session.NewSession(&aws.Config{
        Region: aws.String("us-west-2")},
    )

    // Create a IAM service client.
    svc := iam.New(sess)
```

Call the method to delete the certificate, specifying the name of certificate.

```
    _, err = svc.DeleteServerCertificate(&iam.DeleteServerCertificateInput{
        ServerCertificateName: aws.String("CERTIFICATE_NAME"),
    })
    if err != nil {
        fmt.Println("Error", err)
        return
    }

    fmt.Println("Server certificate deleted")
}
```

See the complete example on GitHub.

AWS Key Management Service Examples Using the AWS SDK for Go

You can use the following examples to access AWS Key Management Service (AWS KMS) using the AWS SDK for Go. For more information about AWS KMS, see the AWS KMS documentation. For reference information about the AWS KMS client, see the New function.

Examples

Topics

- Creating a CMK in AWS Key Management Service
- Encrypting Data with AWS Key Management Service
- Decrypting a Data Blob in AWS Key Management Service
- Re-encrypting a Data Blob in AWS Key Management Service

Creating a CMK in AWS Key Management Service

The following example uses the AWS SDK for GoCreateKey method, which implements the CreateKey operation, to create a customer master key (CMK). Since the example only encrypts a small amount of data, a CMK is fine for our purposes. For larger amounts of data, use the CMK to encrypt a data encryption key (DEK).

```
import (
    "github.com/aws/aws-sdk-go/aws"
    "github.com/aws/aws-sdk-go/aws/session"
    "github.com/aws/aws-sdk-go/service/kms"

    "fmt"
    "os"
)

// Create an AWS KMS key (KMS key)
// Since we are only encrypting small amounts of data (4 KiB or less) directly,
// a KMS key is fine for our purposes.
// For larger amounts of data,
// use the KMS key to encrypt a data encryption key (DEK).

func main() {
    // Initialize a session in us-west-2 that the SDK will use to load
    // credentials from the shared credentials file ~/.aws/credentials.
    sess, err := session.NewSession(&aws.Config{
        Region: aws.String("us-west-2")},
    )

    // Create KMS service client
    svc := kms.New(sess)

    // Create the key
    result, err := svc.CreateKey(&kms.CreateKeyInput{
        Tags: []*kms.Tag{
            {
                TagKey:   aws.String("CreatedBy"),
                TagValue: aws.String("ExampleUser"),
            },
        },
    })

    if err != nil {
```

```
        fmt.Println("Got error creating key: ", err)
        os.Exit(1)
    }

    fmt.Println(*result.KeyMetadata.KeyId)
}
```

Choose Copy to save the code locally. See the complete example on GitHub.

Encrypting Data with AWS Key Management Service

The following example uses the AWS SDK for GoEncrypt method, which implements the Encrypt operation, to encrypt the string “1234567890”. The example displays a readable version of the resulting encrypted blob.

```
import (
    "github.com/aws/aws-sdk-go/aws"
    "github.com/aws/aws-sdk-go/aws/session"
    "github.com/aws/aws-sdk-go/service/kms"

    "fmt"
    "os"
)

func main() {
    // Initialize a session in us-west-2 that the SDK will use to load
    // credentials from the shared credentials file ~/.aws/credentials.
    sess, err := session.NewSession(&aws.Config{
        Region: aws.String("us-west-2")},
    )

    // Create KMS service client
    svc := kms.New(sess)

    // Encrypt data key
    //
    // Replace the fictitious key ARN with a valid key ID

    keyId := "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab"

    text := "1234567890"
```

```
    // Encrypt the data
    result, err := svc.Encrypt(&kms.EncryptInput{
        KeyId: aws.String(keyId),
        Plaintext: []byte(text),
    })

    if err != nil {
        fmt.Println("Got error encrypting data: ", err)
        os.Exit(1)
    }

    fmt.Println("Blob (base-64 byte array):")
    fmt.Println(result.CiphertextBlob)
}
```

Choose Copy to save the code locally. See the complete example on GitHub.

Decrypting a Data Blob in AWS Key Management Service

The following example uses the AWS SDK for GoDecrypt method, which implements the Decrypt operation, to decrypt the provided string and emits the result.

```
import (
    "github.com/aws/aws-sdk-go/aws/session"
    "github.com/aws/aws-sdk-go/service/kms"

    "fmt"
    "os"
)

func main() {
    // Initialize a session that the SDK uses to load
    // credentials from the shared credentials file ~/.aws/credentials
    // and configuration from the shared configuration file ~/.aws/config.
    sess := session.Must(session.NewSessionWithOptions(session.Options{
        SharedConfigState: session.SharedConfigEnable,
    }))

    // Create KMS service client
    svc := kms.New(sess)

    // Encrypted data
    blob := []byte("1234567890")
```

```
    // Decrypt the data
    result, err := svc.Decrypt(&kms.DecryptInput{CiphertextBlob: blob})

    if err != nil {
        fmt.Println("Got error decrypting data: ", err)
        os.Exit(1)
    }

    blob_string := string(result.Plaintext)

    fmt.Println(blob_string)
```

Choose Copy to save the code locally. See the complete example on GitHub.

Re-encrypting a Data Blob in AWS Key Management Service

The following example uses the AWS SDK for GoReEncrypt method, which implements the ReEncrypt operation, to decrypt encrypted data and then immediately re-encrypt data under a new customer master key (CMK). The operations are performed entirely on the server side within AWS KMS, so they never expose your plaintext outside of AWS KMS. The example displays a readable version of the resulting re-encrypted blob.

```
import (
    "github.com/aws/aws-sdk-go/aws/session"
    "github.com/aws/aws-sdk-go/service/kms"

    "fmt"
    "os"
)

func main() {
    // Initialize a session that the SDK uses to load
    // credentials from the shared credentials file ~/.aws/credentials
    // and configuration from the shared configuration file ~/.aws/config.
    sess := session.Must(session.NewSessionWithOptions(session.Options{
        SharedConfigState: session.SharedConfigEnable,
    }))

    // Create KMS service client
    svc := kms.New(sess)
```

```
    // Encrypt data key
    //
    // Replace the fictitious key ARN with a valid key ID

    keyId := "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab"

    // Encrypted data
    blob := []byte("1234567890")

    // Re-encrypt the data key
    result, err := svc.ReEncrypt(&kms.ReEncryptInput{CiphertextBlob: blob,
 DestinationKeyId: &keyId})

    if err != nil {
        fmt.Println("Got error re-encrypting data: ", err)
        os.Exit(1)
    }

    fmt.Println("Blob (base-64 byte array):")
    fmt.Println(result.CiphertextBlob)
```

Choose Copy to save the code locally. See the complete example on GitHub.

AWS Lambda Examples Using the AWS SDK for Go

AWS Lambda (Lambda) is a zero-administration compute platform for backend web developers that runs your code for you in the AWS Cloud, and provides you with a fine-grained pricing structure. You can use the following examples to access AWS Lambda (Lambda) using the AWS SDK for Go. For more information about Lambda, see the Lambda documentation.

You can download complete versions of these example files from the aws-doc-sdk-examples repository on GitHub.

Topics

- Displaying Information about All Lambda Functions
- Creating a Lambda Function
- Running a Lambda Function
- Configuring a Lambda Function to Receive Notifications

Displaying Information about All Lambda Functions

First import the packages we use in this example.

```
import (
    "github.com/aws/aws-sdk-go/aws"
    "github.com/aws/aws-sdk-go/aws/session"
    "github.com/aws/aws-sdk-go/service/lambda"

    "fmt"
    "os"
)
```

Next, create the session and Lambda client.

```
sess := session.Must(session.NewSessionWithOptions(session.Options{
    SharedConfigState: session.SharedConfigEnable,
}))

svc := lambda.New(sess, &aws.Config{Region: aws.String("us-west-2")})
```

Next, call `ListFunctions` and exit if there is an error.

```
result, err := svc.ListFunctions(nil)
if err != nil {
    fmt.Println("Cannot list functions")
    os.Exit(0)
}
```

Finally, display the names and descriptions of the Lambda functions.

```
fmt.Println("Functions:")

for _, f := range result.Functions {
    fmt.Println("Name:        " + aws.StringValue(f.FunctionName))
    fmt.Println("Description: " + aws.StringValue(f.Description))
    fmt.Println("")
}
```

See the complete example on GitHub.

Creating a Lambda Function

The following example creates the Lambda function `functionName` in the `us-west-2` region using the following values:

- Role ARN: `resourceArn`. In most cases, you need to attach only the `AWSLambdaExecute` managed policy to the policy for this role.
- Function entry point: `handler`
- Runtime: `runtime`
- Zip file: `zipFileName` + `.zip`
- Bucket: `amzn-s3-demo-bucket`
- Key: `zipFileName`

The first step is to import the packages we use in the example.

```
import (
    "flag"
    "fmt"
    "io/ioutil"

    "github.com/aws/aws-sdk-go/aws/session"
    "github.com/aws/aws-sdk-go/service/lambda"
)
```

Get the zip file name, bucket name, function name, handler, ARN, and runtime values from the command line. If any are missing, print and error message and quit.

```
zipFile := flag.String("z", "", "The name of the ZIP file, without the .zip
 extension.")
bucket := flag.String("b", "", "the name of bucket to which the ZIP file is uploaded.")
function := flag.String("f", "", "The name of the Lambda function.")
handler := flag.String("h", "main", "The name of the package.class handling the call.")
roleARN := flag.String("a", "", "The ARN of the role that calls the function.")
runtime := flag.String("r", "go1.x", "The runtime for the function.")

flag.Parse()

if *zipFile == "" || *bucket == "" || *function == "" || *handler == "" || *roleARN ==
 "" || *runtime == "" {
```

```
    fmt.Println("You must supply a zip file name, bucket name, function name, handler
 (package) name, role ARN, and runtime value.")
    return
}
```

Create the session and Lambda client and get the contents of the zip file. If you cannot read the zip file contents, print an error message and quit.

```
sess := session.Must(session.NewSessionWithOptions(session.Options{
    SharedConfigState: session.SharedConfigEnable,
}))
```

Next, create the structures for the input argument to the `CreateFunction` function.

```
createCode := &lambda.FunctionCode{
    //      S3Bucket:        bucket,
    //      S3Key:           zipFile,
    //      S3ObjectVersion: aws.String("1"),
    ZipFile: contents,
}

createArgs := &lambda.CreateFunctionInput{
    Code:         createCode,
    FunctionName: function,
    Handler:      handler,
    Role:         role,
    Runtime:      runtime,
}
```

Finally, call `CreateFunction` and display a message with the result of the call.

```
result, err := svc.CreateFunction(createArgs)
```

Running a Lambda Function

The following example runs the Lambda function `MyGetitemsFunction` in the `us-west-2` region. This Node.js function returns a list of items from a database. The input JSON looks like the following.

```
{
```

```
   "SortBy": "name|time",
   "SortOrder": "ascending|descending",
   "Number": 50
}
```

Where:

- `SortBy` is the criteria for sorting the results. Our example uses `time`, which means the returned items are sorted in the order in which they were added to the database.
- `SortOrder` is the order of sorting. Our example uses `descending`, which means the most-recent item is last in the list.
- `Number` is the maximum number of items to retrieve (the default is 50). Our example uses `10`, which means get the 10 most-recent items.

The output JSON looks like the following when the function succeeds and two items are returned.

```
{
   "statusCode": 200,
   "body": {
      "result": "success",
      "error": ""
      "data": [
         {
            "item": "item1"
         },
         {
            "item": "item2"
         }
      ]
   }
}
```

Where:

- `statusCode`– An HTTP status code; `200` means the call was successful.
- body– The body of the returned JSON.
- `result`– The result of the call, either `success` or `failure`.
- `error`– An error message if `result` is `failure`; otherwise, an empty string.
- `data`– The returned results if `result` is `success`; otherwise, nil.

- `item`– An item from the list of results.

The first step is to import the packages we use.

```
import (
    "github.com/aws/aws-sdk-go/aws"
    "github.com/aws/aws-sdk-go/aws/session"
    "github.com/aws/aws-sdk-go/service/lambda"

    "encoding/json"
    "fmt"
    "os"
    "strconv"
)
```

Next create session and Lambda client we use to invoke the Lambda function.

```
sess := session.Must(session.NewSessionWithOptions(session.Options{
    SharedConfigState: session.SharedConfigEnable,
}))

client := lambda.New(sess, &aws.Config{Region: aws.String("us-west-2")})
```

Next, create the request and payload, and call `MyGetItemsFunction`. If there is an error, display a message and quit.

```
request := getItemsRequest{"time", "descending", 10}

payload, err := json.Marshal(request)
if err != nil {
    fmt.Println("Error marshalling MyGetItemsFunction request")
    os.Exit(0)
}

result, err := client.Invoke(&lambda.InvokeInput{FunctionName:
 aws.String("MyGetItemsFunction"), Payload: payload})
if err != nil {
    fmt.Println("Error calling MyGetItemsFunction")
    os.Exit(0)
}
```

Finally, parse the response, and if successful, print out the items.

```
var resp getItemsResponse

err = json.Unmarshal(result.Payload, &resp)
if err != nil {
    fmt.Println("Error unmarshalling MyGetItemsFunction response")
    os.Exit(0)
}

// If the status code is NOT 200, the call failed
if resp.StatusCode != 200 {
    fmt.Println("Error getting items, StatusCode: " + strconv.Itoa(resp.StatusCode))
    os.Exit(0)
}

// If the result is failure, we got an error
if resp.Body.Result == "failure" {
    fmt.Println("Failed to get items")
    os.Exit(0)
}

// Print out items
if len(resp.Body.Data) > 0 {
    for i := range resp.Body.Data {
        fmt.Println(resp.Body.Data[i].Item)
    }
} else {
    fmt.Println("There were no items")
}
```

See the complete example on GitHub.

Note

The complete example includes the structures for marshaling the JSON request and unmarshaling the JSON response.

Configuring a Lambda Function to Receive Notifications

The following example configures the Lambda function `functionName` to accept notifications from the resource with the ARN `sourceArn`.

Import the packages we use.

```
import (
    "flag"
    "fmt"
    "os"

    "github.com/aws/aws-sdk-go/aws"
    "github.com/aws/aws-sdk-go/aws/session"
    "github.com/aws/aws-sdk-go/service/lambda"
)
```

Get the name of the function and ARN of the entity invoking the function. If either is missing, print an error message and quit.

```
functionPtr := flag.String("f", "", "The name of the Lambda function")
sourcePtr := flag.String("a", "", "The ARN of the entity invoking the function")
flag.Parse()

if *functionPtr == "" || *sourcePtr == "" {
    fmt.Println("You must supply the name of the function and of the entity invoking
 the function")
    flag.PrintDefaults()
    os.Exit(1)
}
```

Create the session and Lambda client.

```
sess := session.Must(session.NewSessionWithOptions(session.Options{
    SharedConfigState: session.SharedConfigEnable,
}))

svc := lambda.New(sess)
```

Create the structure for the input argument to the `AddPermission` function.

```
permArgs := &lambda.AddPermissionInput{
```

```
    Action:       aws.String("lambda:InvokeFunction"),
    FunctionName: functionPtr,
    Principal:    aws.String("s3.amazonaws.com"),
    SourceArn:    sourcePtr,
    StatementId:  aws.String("lambda_s3_notification"),
}
```

Finally, call `AddPermission` and display a message with the result of the call.

```
result, err := svc.AddPermission(permArgs)
if err != nil {
    fmt.Println("Cannot configure function for notifications")
    os.Exit(0)
}

fmt.Println(result)
```

See the complete example on GitHub.

Amazon Polly Examples Using the AWS SDK for Go

Amazon Polly is a cloud service that converts text into lifelike speech. The AWS SDK for Go examples can integrate Amazon Polly into your applications. The examples assume you have already set up and configured the SDK (that is, you've imported all required packages and set your credentials and region). For more information, see Getting Started with the AWS SDK for Go and Configuring the AWS SDK for Go.

You can download complete versions of these example files from the aws-doc-sdk-examples repository on GitHub.

Topics

- Getting a List of Voices
- Getting a List of Lexicons
- Synthesizing Speech

Getting a List of Voices

This example uses the DescribeVoices operation to get the list of voices in the `us-west-2` region.

Choose Copy to save the code locally.

Create the file *pollyDescribeVoices.go*. Import the packages used in the example.

```
import (
    "github.com/aws/aws-sdk-go/aws"
    "github.com/aws/aws-sdk-go/aws/session"
    "github.com/aws/aws-sdk-go/service/polly"

    "fmt"
    "os"
)
```

Initialize a session that the SDK will use to load credentials from the shared credentials file `~/.aws/credentials`, load your configuration from the shared configuration file `~/.aws/config`, and create an Amazon Polly client.

```
sess := session.Must(session.NewSessionWithOptions(session.Options{
    SharedConfigState: session.SharedConfigEnable,
}))

svc := polly.New(sess)
```

Create the input for and call `DescribeVoices`.

```
input := &polly.DescribeVoicesInput{LanguageCode: aws.String("en-US")}
resp, err := svc.DescribeVoices(input)
```

Display the name and gender of the voices.

```
for _, v := range resp.Voices {
    fmt.Println("Name:   " + *v.Name)
    fmt.Println("Gender: " + *v.Gender)
    fmt.Println("")
}
```

See the complete example on GitHub.

Getting a List of Lexicons

This example uses the ListLexicons operation to get the list of lexicons in the `us-west-2` region.

Choose Copy to save the code locally.

Create the file *pollyListLexicons.go*. Import the packages used in the example.

```
import (
    "github.com/aws/aws-sdk-go/aws/session"
    "github.com/aws/aws-sdk-go/service/polly"

    "fmt"
    "os"
)
```

Initialize a session that the SDK will use to load credentials from the shared credentials file `~/.aws/credentials`, load your configuration from the shared configuration file `~/.aws/config`, and create an Amazon Polly client.

```
sess := session.Must(session.NewSessionWithOptions(session.Options{
    SharedConfigState: session.SharedConfigEnable,
}))

svc := polly.New(sess)
```

Call `ListLexicons` and display the name, alphabet, and language code of each lexicon.

```
resp, err := svc.ListLexicons(nil)

for _, l := range resp.Lexicons {
    fmt.Println(*l.Name)
    fmt.Println("  Alphabet: " + *l.Attributes.Alphabet)
    fmt.Println("  Language: " + *l.Attributes.LanguageCode)
    fmt.Println("")
}
```

See the complete example on GitHub.

Synthesizing Speech

This example uses the SynthesizeSpeech operation to get the text from a file and produce an MP3 file containing the synthesized speech.

Choose Copy to save the code locally.

Create the file *pollySynthesizeSpeech.go*. Import the packages used in the example.

```
import (
    "github.com/aws/aws-sdk-go/aws"
    "github.com/aws/aws-sdk-go/aws/session"
    "github.com/aws/aws-sdk-go/service/polly"

    "fmt"
    "os"
    "strings"
    "io"
    "io/ioutil"
)
```

Get the name of the text file from the command line.

```
if len(os.Args) != 2 {
    fmt.Println("You must supply an alarm name")
    os.Exit(1)
}

fileName := os.Args[1]
```

Open the text file and read the contents as a string.

```
contents, err := ioutil.ReadFile(fileName)
s := string(contents[:])
```

Initialize a session that the SDK will use to load credentials from the shared credentials file `~/.aws/credentials`, load your configuration from the shared configuration file `~/.aws/config`, and create an Amazon Polly client.

```
sess := session.Must(session.NewSessionWithOptions(session.Options{
    SharedConfigState: session.SharedConfigEnable,
}))

svc := polly.New(sess)
```

Create the input for and call `SynthesizeSpeech`.

```
input := &polly.SynthesizeSpeechInput{OutputFormat: aws.String("mp3"), Text:
 aws.String(s), VoiceId: aws.String("Joanna")}

output, err := svc.SynthesizeSpeech(input)
```

Save the resulting synthesized speech as an MP3 file.

```
names := strings.Split(fileName, ".")
name := names[0]
mp3File := name + ".mp3"

outFile, err := os.Create(mp3File)
defer outFile.Close()
```

> **Note**
>
> The resulting MP3 file is in the MPEG-2 format.

See the complete example on GitHub.

Amazon S3 Examples Using the AWS SDK for Go

Amazon Simple Storage Service (Amazon S3) is storage for the internet. The AWS SDK for Go examples can integrate Amazon S3 into your applications. The examples assume you have already set up and configured the SDK (that is, you've imported all required packages and set your credentials and region). For more information, see Getting Started with the AWS SDK for Go and Configuring the AWS SDK for Go.

You can download complete versions of these example files from the aws-doc-sdk-examples repository on GitHub.

Topics

- Performing Basic Amazon S3 Bucket Operations
- Creating Pre-Signed URLs for Amazon S3 Buckets
- Using an Amazon S3 Bucket as a Static Web Host
- Working with Amazon S3 CORS Permissions

- Working with Amazon S3 Bucket Policies
- Working with Amazon S3 Bucket ACLs
- Encrypting Amazon S3 Bucket Items

Performing Basic Amazon S3 Bucket Operations

These AWS SDK for Go examples show you how to perform the following operations on Amazon S3 buckets and bucket items:

- List the buckets in your account
- Create a bucket
- List the items in a bucket
- Upload a file to a bucket
- Download a bucket item
- Copy a bucket item to another bucket
- Delete a bucket item
- Delete all the items in a bucket
- Restore a bucket item
- Delete a bucket
- List the users with administrator privileges

You can download complete versions of these example files from the aws-doc-sdk-examples repository on GitHub.

Scenario

In these examples, a series of Go routines are used to perform operations on your Amazon S3 buckets. The routines use the AWS SDK for Go to perform Amazon S3 bucket operations using the following methods of the Amazon S3 client class, unless otherwise noted:

- ListBuckets
- CreateBucket
- ListObjects

- Upload (from the **s3manager.NewUploader** class)
- Download (from the **s3manager.NewDownloader** class)
- CopyObject
- DeleteObject
- DeleteObjects
- RestoreObject
- DeleteBucket

Prerequisites

- You have set up and configured the AWS SDK for Go.
- You are familiar with buckets. To learn more, see Working with Amazon S3 Buckets in the Amazon S3 Developer Guide.

List Buckets

The ListBuckets function lists the buckets in your account.

The following example lists the buckets in your account. There are no command line arguments.

Create the file *s3_list_buckets.go*. Add the following statements to import the Go and AWS SDK for Go packages used in the example.

```
import (
    "github.com/aws/aws-sdk-go/aws"
    "github.com/aws/aws-sdk-go/aws/session"
    "github.com/aws/aws-sdk-go/service/s3"
    "fmt"
    "os"
)
```

Create a function we use to display errors and exit.

```
func exitErrorf(msg string, args ...interface{}) {
    fmt.Fprintf(os.Stderr, msg+"\n", args...)
    os.Exit(1)
}
```

Initialize the session that the SDK uses to load credentials from the shared credentials file `~/.aws/credentials`, and create a new Amazon S3 service client.

```
sess, err := session.NewSession(&aws.Config{
    Region: aws.String("us-west-2")},
)

// Create S3 service client
svc := s3.New(sess)
```

Call ListBuckets. Passing `nil` means no filters are applied to the returned list. If an error occurs, call `exitErrorf`. If no error occurs, loop through the buckets, printing the name and creation date of each bucket.

```
result, err := svc.ListBuckets(nil)
if err != nil {
    exitErrorf("Unable to list buckets, %v", err)
}

fmt.Println("Buckets:")

for _, b := range result.Buckets {
    fmt.Printf("* %s created on %s\n",
        aws.StringValue(b.Name), aws.TimeValue(b.CreationDate))
}
```

See the complete example on GitHub.

Create a Bucket

The CreateBucket function creates a bucket in your account.

The following example creates a bucket with the name specified as a command line argument. You must specify a globally unique name for the bucket.

Create the file *s3_create_bucket.go*. Import the following Go and AWS SDK for Go packages.

```
import (
    "github.com/aws/aws-sdk-go/aws"
    "github.com/aws/aws-sdk-go/aws/session"
    "github.com/aws/aws-sdk-go/service/s3"
```

```
    "fmt"
    "os"
)
```

Create a function we use to display errors and exit.

```
func exitErrorf(msg string, args ...interface{}) {
    fmt.Fprintf(os.Stderr, msg+"\n", args...)
    os.Exit(1)
}
```

The program requires one argument, the name of the bucket to create.

```
if len(os.Args) != 2 {
    exitErrorf("Bucket name missing!\nUsage: %s bucket_name", os.Args[0])
}

bucket := os.Args[1]
```

Initialize the session that the SDK uses to load credentials from the shared credentials file `~/.aws/credentials`, and create a new S3 service client.

```
sess, err := session.NewSession(&aws.Config{
    Region: aws.String("us-west-2")},
)

// Create S3 service client
svc := s3.New(sess)
```

Call CreateBucket, passing in the bucket name defined previously. If an error occurs, call `exitErrorf`. If there are no errors, wait for a notification that the bucket was created.

```
_, err = svc.CreateBucket(&s3.CreateBucketInput{
    Bucket: aws.String(bucket),
})
if err != nil {
    exitErrorf("Unable to create bucket %q, %v", bucket, err)
}

// Wait until bucket is created before finishing
fmt.Printf("Waiting for bucket %q to be created...\n", bucket)
```

```
err = svc.WaitUntilBucketExists(&s3.HeadBucketInput{
    Bucket: aws.String(bucket),
})
```

If the `WaitUntilBucketExists` call returns an error, call `exitErrorf`. If there are no errors, notify the user of success.

```
if err != nil {
    exitErrorf("Error occurred while waiting for bucket to be created, %v", bucket)
}

fmt.Printf("Bucket %q successfully created\n", bucket)
```

See the complete example on GitHub.

List Bucket Items

The ListObjects function lists the items in a bucket.

The following example lists the items in the bucket with the name specified as a command line argument.

Create the file *s3_list_objects.go*. Add the following statements to import the Go and AWS SDK for Go packages used in the example.

```
import (
    "github.com/aws/aws-sdk-go/aws"
    "github.com/aws/aws-sdk-go/aws/session"
    "github.com/aws/aws-sdk-go/service/s3"
    "fmt"
    "os"
)
```

Create a function we use to display errors and exit.

```
func exitErrorf(msg string, args ...interface{}) {
    fmt.Fprintf(os.Stderr, msg+"\n", args...)
    os.Exit(1)
}
```

The program requires one command line argument, the name of the bucket.

```
if len(os.Args) != 2 {
    exitErrorf("Bucket name required\nUsage: %s bucket_name",
        os.Args[0])
}

bucket := os.Args[1]
```

Initialize the session that the SDK uses to load credentials from the shared credentials file `~/.aws/credentials`, and create a new Amazon S3 service client.

```
sess, err := session.NewSession(&aws.Config{
    Region: aws.String("us-west-2")},
)

// Create S3 service client
svc := s3.New(sess)
```

Call ListObjects, passing in the name of the bucket. If an error occurs, call `exitErrorf`. If no error occurs, loop through the items, printing the name, last modified date, size, and storage class of each item.

```
resp, err := svc.ListObjectsV2(&s3.ListObjectsV2Input{Bucket: aws.String(bucket)})
if err != nil {
    exitErrorf("Unable to list items in bucket %q, %v", bucket, err)
}

for _, item := range resp.Contents {
    fmt.Println("Name:         ", *item.Key)
    fmt.Println("Last modified:", *item.LastModified)
    fmt.Println("Size:         ", *item.Size)
    fmt.Println("Storage class:", *item.StorageClass)
    fmt.Println("")
}
```

See the complete example on GitHub.

Upload a File to a Bucket

The Upload function uploads an object to a bucket.

The following example uploads a file to a bucket with the names specified as command line arguments.

Create the file *s3_upload_object.go*. Add the following statements to import the Go and AWS SDK for Go packages used in the example.

```
import (
    "github.com/aws/aws-sdk-go/aws"
    "github.com/aws/aws-sdk-go/aws/session"
    "github.com/aws/aws-sdk-go/service/s3/s3manager"
    "fmt"
    "os"
)
```

Create a function we use to display errors and exit.

```
func exitErrorf(msg string, args ...interface{}) {
    fmt.Fprintf(os.Stderr, msg+"\n", args...)
    os.Exit(1)
}
```

Get the bucket and file name from the command line arguments, open the file, and defer the file closing until we are done with it. If an error occurs, call `exitErrorF`.

```
if len(os.Args) != 3 {
    exitErrorf("bucket and file name required\nUsage: %s bucket_name filename",
        os.Args[0])
}

bucket := os.Args[1]
filename := os.Args[2]

file, err := os.Open(filename)
if err != nil {
    exitErrorf("Unable to open file %q, %v", err)
}

defer file.Close()
```

Initialize the session that the SDK uses to load credentials from the shared credentials file `~/.aws/credentials`, and create a `NewUploader` object.

```
sess, err := session.NewSession(&aws.Config{
    Region: aws.String("us-west-2")},
)

// Setup the S3 Upload Manager. Also see the SDK doc for the Upload Manager
// for more information on configuring part size, and concurrency.
//
// http://docs.aws.amazon.com/sdk-for-go/api/service/s3/s3manager/#NewUploader
uploader := s3manager.NewUploader(sess)
```

Upload the file to the bucket. If an error occurs, call `exitErrorF`. Otherwise, notify the user that the upload succeeded.

```
_, err = uploader.Upload(&s3manager.UploadInput{
    Bucket: aws.String(bucket),
    Key: aws.String(filename),
    Body: file,
})
if err != nil {
    // Print the error and exit.
    exitErrorf("Unable to upload %q to %q, %v", filename, bucket, err)
}

fmt.Printf("Successfully uploaded %q to %q\n", filename, bucket)
```

See the complete example on GitHub.

Download a File from a Bucket

The Download function downloads an object from a bucket.

The following example downloads an item from a bucket with the names specified as command line arguments.

Create the file *s3_download_object.go*. Add the following statements to import the Go and AWS SDK for Go packages used in the example.

```
import (
    "github.com/aws/aws-sdk-go/aws"
    "github.com/aws/aws-sdk-go/aws/session"
    "github.com/aws/aws-sdk-go/service/s3"
    "github.com/aws/aws-sdk-go/service/s3/s3manager"
```

```
    "fmt"
    "os"
)
```

Create a function we use to display errors and exit.

```
func exitErrorf(msg string, args ...interface{}) {
    fmt.Fprintf(os.Stderr, msg+"\n", args...)
    os.Exit(1)
}
```

Get the bucket and file name from the command line arguments. If there aren't two arguments, call `exitErrorf`. Otherwise, create the file and defer file closing until we are done downloading. If an error occurs while creating the file, call `exitErrorf`.

```
if len(os.Args) != 3 {
    exitErrorf("Bucket and item names required\nUsage: %s bucket_name item_name",
        os.Args[0])
}

bucket := os.Args[1]
item := os.Args[2]
```

Initialize the session in us-west-2 that the SDK uses to load credentials from the shared credentials file `~/.aws/credentials`, and create a `NewDownloader` object.

```
sess, _ := session.NewSession(&aws.Config{
    Region: aws.String("us-west-2")},
)

downloader := s3manager.NewDownloader(sess)
```

Download the item from the bucket. If an error occurs, call `exitErrorf`. Otherwise, notify the user that the download succeeded.

```
numBytes, err := downloader.Download(file,
    &s3.GetObjectInput{
        Bucket: aws.String(bucket),
        Key:    aws.String(item),
```

```
    })
if err != nil {
    exitErrorf("Unable to download item %q, %v", item, err)
}

fmt.Println("Downloaded", file.Name(), numBytes, "bytes")
```

See the complete example on GitHub.

Copy an Item from one Bucket to Another

The CopyObject function copies an object from one bucket to another.

The following example copies an item from one bucket to another with the names specified as command line arguments.

Create the file *s3_copy_object.go*. Add the following statements to import the Go and AWS SDK for Go packages used in the example.

```
import (
    "github.com/aws/aws-sdk-go/aws"
    "github.com/aws/aws-sdk-go/aws/session"
    "github.com/aws/aws-sdk-go/service/s3"
    "fmt"
    "net/url"
    "os"
)
```

Create a function we use to display errors and exit.

```
func exitErrorf(msg string, args ...interface{}) {
    fmt.Fprintf(os.Stderr, msg+"\n", args...)
    os.Exit(1)
}
```

Get the names of the bucket containing the item, the item to copy, and the name of the bucket to which the item is copied. If there aren't four command line arguments, call `exitErrorf`.

```
if len(os.Args) != 4 {
    exitErrorf("Bucket, item, and other bucket names required\nUsage: go run
 s3_copy_object bucket item other-bucket")
```

```
}

bucket := os.Args[1]
item := os.Args[2]
other := os.Args[3]

source := bucket + "/" + item
```

Initialize the session that the SDK uses to load credentials from the shared credentials file `~/.aws/credentials`, and create a new Amazon S3 service client.

```
sess, err := session.NewSession(&aws.Config{
    Region: aws.String("us-west-2")},
)

// Create S3 service client
svc := s3.New(sess)
```

Call CopyObject, with the names of the bucket containing the item, the item to copy, and the name of the bucket to which the item is copied. If an error occurs, call `exitErrorf`. If no error occurs, wait for the item to be copied.

```
// Copy the item
_, err = svc.CopyObject(&s3.CopyObjectInput{Bucket: aws.String(other),
    CopySource: aws.String(url.PathEscape(source)), Key: aws.String(item)})
if err != nil {
    exitErrorf("Unable to copy item from bucket %q to bucket %q, %v", bucket, other,
 err)
}
```

If the `WaitUntilObjectExists` call returns an error, call `exitErrorf`. Otherwise, notify the user that the copy succeeded.

```
// Wait to see if the item got copied
err = svc.WaitUntilObjectExists(&s3.HeadObjectInput{Bucket: aws.String(other), Key:
 aws.String(item)})
if err != nil {
    exitErrorf("Error occurred while waiting for item %q to be copied to bucket %q,
 %v", bucket, item, other, err)
}
```

```
fmt.Printf("Item %q successfully copied from bucket %q to bucket %q\n", item, bucket,
 other)
```

See the complete example on GitHub.

Delete an Item in a Bucket

The DeleteObject function deletes an object from a bucket.

The following example deletes an item from a bucket with the names specified as command line arguments.

Create the file *s3_delete_object.go*. Add the following statements to import the Go and AWS SDK for Go packages used in the example.

```
import (
    "github.com/aws/aws-sdk-go/aws"
    "github.com/aws/aws-sdk-go/aws/session"
    "github.com/aws/aws-sdk-go/service/s3"
    "fmt"
    "os"
)
```

Create a function we use to display errors and exit.

```
func exitErrorf(msg string, args ...interface{}) {
    fmt.Fprintf(os.Stderr, msg+"\n", args...)
    os.Exit(1)
}
```

Get the name of the bucket and object to delete.

```
if len(os.Args) != 3 {
    exitErrorf("Bucket and object name required\nUsage: %s bucket_name object_name",
        os.Args[0])
}

bucket := os.Args[1]
obj := os.Args[2]
```

Initialize the session that the SDK uses to load credentials from the shared credentials file `~/.aws/credentials`, and create a new Amazon S3 service client.

```
sess, err := session.NewSession(&aws.Config{
    Region: aws.String("us-west-2")},
)

// Create S3 service client
svc := s3.New(sess)
```

Call DeleteObject, passing in the names of the bucket and object to delete. If an error occurs, call `exitErrorf`. If no error occurs, wait until the object is deleted.

```
_, err = svc.DeleteObject(&s3.DeleteObjectInput{Bucket: aws.String(bucket), Key:
 aws.String(obj)})
if err != nil {
    exitErrorf("Unable to delete object %q from bucket %q, %v", obj, bucket, err)
}

err = svc.WaitUntilObjectNotExists(&s3.HeadObjectInput{
    Bucket: aws.String(bucket),
    Key:    aws.String(obj),
})
```

If `WaitUntilObjectNotExists` returns an error, call `exitErrorf`. Otherwise, inform the user that the object was successfully deleted.

```

fmt.Printf("Object %q successfully deleted\n", obj)
```

See the complete example on GitHub.

Delete All the Items in a Bucket

The DeleteObjects function deletes objects from a bucket.

The following example deletes all the items from a bucket with the bucket name specified as a command line argument.

Create the file *s3_delete_objects.go*. Add the following statements to import the Go and AWS SDK for Go packages used in the example.

```
import (
    "github.com/aws/aws-sdk-go/aws"
    "github.com/aws/aws-sdk-go/aws/session"
    "github.com/aws/aws-sdk-go/service/s3"
    "github.com/aws/aws-sdk-go/service/s3/s3manager"

    "fmt"
    "os"
)
```

Create a function we use to display errors and exit.

```
func exitErrorf(msg string, args ...interface{}) {
    fmt.Fprintf(os.Stderr, msg+"\n", args...)
    os.Exit(1)
}
```

Get the name of the bucket.

```
if len(os.Args) != 2 {
    exitErrorf("Bucket name required\nUsage: %s BUCKET", os.Args[0])
}

bucket := os.Args[1]
```

Initialize the session that the SDK uses to load credentials from the shared credentials file `~/.aws/credentials`, and create a new Amazon S3 service client.

```
sess, _ := session.NewSession(&aws.Config{
    Region: aws.String("us-west-2")},
)
svc := s3.New(sess)
```

Create a list iterator to iterate through the list of bucket objects, deleting each object. If an error occurs, call `exitErrorf`.

```
iter := s3manager.NewDeleteListIterator(svc, &s3.ListObjectsInput{
    Bucket: aws.String(bucket),
})
```

```
if err := s3manager.NewBatchDeleteWithClient(svc).Delete(aws.BackgroundContext(),
 iter); err != nil {
    exitErrorf("Unable to delete objects from bucket %q, %v", bucket, err)
}
```

Once all of the items in the bucket have been deleted, inform the user that the objects were deleted.

```
fmt.Printf("Deleted object(s) from bucket: %s", bucket)
```

See the complete example on GitHub.

Restore a Bucket Item

The RestoreObject function restores an item in a bucket.

The following example restores the items in a bucket with the names specified as command line arguments.

Create the file *s3_restore_object.go*. Add the following statements to import the Go and AWS SDK for Go packages used in the example.

```
import (
    "github.com/aws/aws-sdk-go/aws"
    "github.com/aws/aws-sdk-go/aws/session"
    "github.com/aws/aws-sdk-go/service/s3"
    "fmt"
    "os"
)
```

Create a function we use to display errors and exit.

```
func exitErrorf(msg string, args ...interface{}) {
    fmt.Fprintf(os.Stderr, msg+"\n", args...)
    os.Exit(1)
}
```

The program requires two arguments, the names of the bucket and object to restore.

```
if len(os.Args) != 3 {
```

```
    exitErrorf("Bucket name and object name required\nUsage: %s bucket_name
 object_name",
        os.Args[0])
}

bucket := os.Args[1]
obj := os.Args[2]
```

Initialize the session that the SDK uses to load credentials from the shared credentials file `~/.aws/credentials`, and create a new Amazon S3 service client.

```
sess, err := session.NewSession(&aws.Config{
    Region: aws.String("us-west-2")},
)

// Create S3 service client
svc := s3.New(sess)
```

Call RestoreObject, passing in the bucket and object names and the number of days to temporarily restore. If an error occurs, call `exitErrorf`. Otherwise, inform the user that the bucket should be restored in the next four hours or so.

```
_, err = svc.RestoreObject(&s3.RestoreObjectInput{Bucket: aws.String(bucket), Key:
 aws.String(obj), RestoreRequest: &s3.RestoreRequest{Days: aws.Int64(30)}})
if err != nil {
    exitErrorf("Could not restore %s in bucket %s, %v", obj, bucket, err)
}

fmt.Printf("%q should be restored to %q in about 4 hours\n", obj, bucket)
```

See the complete example on GitHub.

Delete a Bucket

The DeleteBucket function deletes a bucket.

The following example deletes the bucket with the name specified as a command line argument.

Create the file *s3_delete_bucket.go*. Import the following Go and AWS SDK for Go packages.

```
import (
```

```
    "github.com/aws/aws-sdk-go/aws"
    "github.com/aws/aws-sdk-go/aws/session"
    "github.com/aws/aws-sdk-go/service/s3"
    "fmt"
    "os"
)
```

Create a function we use to display errors and exit.

```
func exitErrorf(msg string, args ...interface{}) {
    fmt.Fprintf(os.Stderr, msg+"\n", args...)
    os.Exit(1)
}
```

The program requires one argument, the name of the bucket to delete. If the argument is not supplied, call `exitErrorf`.

```
if len(os.Args) != 2 {
    exitErrorf("bucket name required\nUsage: %s bucket_name", os.Args[0])
}

bucket := os.Args[1]
```

Initialize the session that the SDK uses to load credentials from the shared credentials file `~/.aws/credentials`, and create a new S3 service client.

```
sess, err := session.NewSession(&aws.Config{
    Region: aws.String("us-west-2")},
)

// Create S3 service client
svc := s3.New(sess)
```

Call DeleteBucket, passing in the bucket name. If an error occurs, call `exitErrorf`. If there are no errors, wait for a notification that the bucket was deleted.

```
_, err = svc.DeleteBucket(&s3.DeleteBucketInput{
    Bucket: aws.String(bucket),
})
if err != nil {
    exitErrorf("Unable to delete bucket %q, %v", bucket, err)
```

```
}

// Wait until bucket is deleted before finishing
fmt.Printf("Waiting for bucket %q to be deleted...\n", bucket)

err = svc.WaitUntilBucketNotExists(&s3.HeadBucketInput{
    Bucket: aws.String(bucket),
})
```

If `WaitUntilBucketNotExists` returns an error, call `exitErrorf`. Otherwise, inform the user that the bucket was successfully deleted.

```
if err != nil {
    exitErrorf("Error occurred while waiting for bucket to be deleted, %v", bucket)
}

fmt.Printf("Bucket %q successfully deleted\n", bucket)
```

See the complete example on GitHub.

Creating Pre-Signed URLs for Amazon S3 Buckets

This Go example shows you how to obtain a pre-signed URL for an Amazon S3 bucket. You can download complete versions of these example files from the aws-doc-sdk-examples repository on GitHub.

Scenario

In this example, a series of Go routines are used to obtain a pre-signed URL for an Amazon S3 bucket using either GetObject or a PUT operation. A pre-signed URL allows you to grant temporary access to users who don't have permission to directly run AWS operations in your account. A pre-signed URL is signed with your credentials and can be used by any user.

- Presign

Prerequisites

- You have set up and configured the SDK.
- You are familiar with pre-signed URLs. To learn more, see Uploading Objects Using Pre-Signed URLs in the Amazon S3 Developer Guide.

Generate a Pre-Signed URL for a GetObject Operation

To generate a pre-signed URL, use the Presign method on the `request` object. You must set an expiration value because the AWS SDK for Go doesn't set one by default.

The following example generates a pre-signed URL that enables you to temporarily share a file without making it public. Anyone with access to the URL can view the file.

```
package main

import (
    "github.com/aws/aws-sdk-go/aws"
    "github.com/aws/aws-sdk-go/aws/session"
    "github.com/aws/aws-sdk-go/service/s3"
    "log"
    "time"
)

// Downloads an item from an S3 Bucket
//
// Usage:
//    go run s3_download.go
func main() {
    // Initialize a session in us-west-2 that the SDK will use to load
    // credentials from the shared credentials file ~/.aws/credentials.
    sess, err := session.NewSession(&aws.Config{
        Region: aws.String("us-west-2")},
    )

    // Create S3 service client
    svc := s3.New(sess)

    req, _ := svc.GetObjectRequest(&s3.GetObjectInput{
        Bucket: aws.String("amzn-s3-demo-bucket"),
        Key:    aws.String("myKey"),
    })
    urlStr, err := req.Presign(15 * time.Minute)

    if err != nil {
        log.Println("Failed to sign request", err)
    }

    log.Println("The URL is", urlStr)
```

```
}
```

If the SDK has not retrieved your credentials before calling `Presign`, it will get them to generate the pre-signed URL.

Generate a Pre-Signed URL for an Amazon S3 PUT Operation with a Specific Payload

You can generate a pre-signed URL for a PUT operation that checks whether users upload the correct content. When the SDK pre-signs a request, it computes the checksum of the request body and generates an MD5 checksum that is included in the pre-signed URL. Users must upload the same content that produces the same MD5 checksum generated by the SDK; otherwise, the operation fails. This is not the Content-MD5, but the signature. To enforce Content-MD5, simply add the header to the request.

The following example adds a `Body` field to generate a pre-signed PUT operation that requires a specific payload to be uploaded by users.

```
package main

import (
    "github.com/aws/aws-sdk-go/aws"
    "github.com/aws/aws-sdk-go/aws/session"
    "github.com/aws/aws-sdk-go/service/s3"
    "log"
    "strings"
    "time"
)

func main() {
    // Initialize a session in us-west-2 that the SDK will use to load
    // credentials from the shared credentials file ~/.aws/credentials.
    sess, err := session.NewSession(&aws.Config{
        Region: aws.String("us-west-2")},
    )

    // Create S3 service client
    svc := s3.New(sess)

    req, _ := svc.PutObjectRequest(&s3.PutObjectInput{
        Bucket: aws.String("amzn-s3-demo-bucket"),
```

```
        Key:    aws.String("myKey"),
        Body:   strings.NewReader("EXPECTED CONTENTS"),
    })
    str, err := req.Presign(15 * time.Minute)

    log.Println("The URL is:", str, " err:", err)
}
```

If you omit the Body field, users can write any contents to the given object.

The following example shows the enforcing of Content-MD5.

```
package main

import (
    "github.com/aws/aws-sdk-go/aws"
    "github.com/aws/aws-sdk-go/aws/session"
    "github.com/aws/aws-sdk-go/service/s3"
    "encoding/base64"
    "fmt"
    "crypto/md5"
    "strings"
    "time"
    "net/http"
)

// Downloads an item from an S3 Bucket in the region configured in the shared config
// or AWS_REGION environment variable.
//
// Usage:
//    go run s3_download.go
func main() {
    h := md5.New()
    content := strings.NewReader("")
    content.WriteTo(h)

    // Initialize a session in us-west-2 that the SDK will use to load
    // credentials from the shared credentials file ~/.aws/credentials.
    sess, err := session.NewSession(&aws.Config{
        Region: aws.String("us-west-2")},
    )

    // Create S3 service client
    svc := s3.New(sess)
```

```
    resp, _ := svc.PutObjectRequest(&s3.PutObjectInput{
        Bucket: aws.String("amzn-s3-demo-bucket"),
        Key:    aws.String("testKey"),
    })

    md5s := base64.StdEncoding.EncodeToString(h.Sum(nil))
    resp.HTTPRequest.Header.Set("Content-MD5", md5s)

    url, err := resp.Presign(15 * time.Minute)
    if err != nil {
        fmt.Println("error presigning request", err)
        return
    }

    req, err := http.NewRequest("PUT", url, strings.NewReader(""))
    req.Header.Set("Content-MD5", md5s)
    if err != nil {
        fmt.Println("error creating request", url)
        return
    }

    defClient, err := http.DefaultClient.Do(req)
    fmt.Println(defClient, err)
}
```

Using an Amazon S3 Bucket as a Static Web Host

This AWS SDK for Go example shows you how to configure an Amazon S3 bucket to act as a static web host. You can download complete versions of these example files from the aws-doc-sdk-examples repository on GitHub.

Scenario

In this example, you use a series of Go routines to configure any of your buckets to act as a static web host. The routines use the AWS SDK for Go to configure a selected Amazon S3 bucket using these methods of the Amazon S3 client class:

- GetBucketWebsite
- PutBucketWebsite
- DeleteBucketWebsite

For more information about using an Amazon S3 bucket as a static web host, see Hosting a Static Website on Amazon S3 in the Amazon S3 Developer Guide.

Prerequisites

- You have set up, and configured the AWS SDK for Go.
- You are familiar with hosting static websites on Amazon S3. To learn more, see Hosting a Static Website on Amazon S3 in the Amazon S3 Developer Guide.

Retrieve a Bucket's Website Configuration

Create a new Go file named `s3_get_bucket_website.go`. You must import the relevant Go and AWS SDK for Go packages by adding the following lines.

```
import (
    "github.com/aws/aws-sdk-go/aws"
    "github.com/aws/aws-sdk-go/aws/awserr"
    "github.com/aws/aws-sdk-go/aws/session"
    "github.com/aws/aws-sdk-go/service/s3"
    "fmt"
    "os"
)
```

This routine requires you to pass in an argument containing the name of the bucket for which you want to get website configuration.

```
if len(os.Args) != 2 {
    exitErrorf("bucket name required\nUsage: %s bucket_name", os.Args[0])
}

bucket := os.Args[1]
```

Initialize a session that the SDK will use to load credentials from the shared credentials file, `~/.aws/credentials`, and create a new S3 service client.

```
sess, err := session.NewSession(&aws.Config{
    Region: aws.String("us-west-2")},
)

// Create S3 service client
```

```
svc := s3.New(sess)
```

Call GetBucketWebsite to get the bucket configuration. You should check for the `NoSuchWebsiteConfiguration` error code, which tells you that the bucket doesn't have a website configured.

```
result, err := svc.GetBucketWebsite(&s3.GetBucketWebsiteInput{
    Bucket: aws.String(bucket),
})
if err != nil {
    // Check for the NoSuchWebsiteConfiguration error code telling us
    // that the bucket does not have a website configured.
    if awsErr, ok := err.(awserr.Error); ok && awsErr.Code() ==
 "NoSuchWebsiteConfiguration" {
        exitErrorf("Bucket %s does not have website configuration\n", bucket)
    }
    exitErrorf("Unable to get bucket website config, %v", err)
}
```

Print the bucket's website configuration.

```
fmt.Println("Bucket Website Configuration:")
fmt.Println(result)
```

Set a Bucket's Website Configuration

Create a Go file named `s3_set_bucket_website.go` and add the code below. The Amazon S3PutBucketWebsite operation sets the website configuration on a bucket, replacing any existing configuration.

Create a new Go file named `s3_create_bucket.go`. You must import the relevant Go and AWS SDK for Go packages by adding the following lines.

```
import (
    "github.com/aws/aws-sdk-go/aws"
    "github.com/aws/aws-sdk-go/aws/session"
    "github.com/aws/aws-sdk-go/service/s3"
    "fmt"
    "os"
    "path/filepath"
)
```

This routine requires you to pass in an argument containing the name of the bucket and the index suffix page.

```
if len(os.Args) != 4 {
    exitErrorf("bucket name and index suffix page required\nUsage: %s bucket_name
 index_page [error_page]",
        filepath.Base(os.Args[0]))
}

bucket := fromArgs(os.Args, 1)
indexSuffix := fromArgs(os.Args, 2)
errorPage := fromArgs(os.Args, 3)
```

Initialize a session that the SDK will use to load configuration, credentials, and region information from the shared credentials file, `~/.aws/credentials`, and create a new S3 service client.

```
sess, err := session.NewSession(&aws.Config{
    Region: aws.String("us-west-2")},
)

// Create S3 service client
svc := s3.New(sess)
```

Create the parameters to be passed in to PutBucketWebsite, including the bucket name and index document details. If the user passed in an error page when calling the routine, also add that to the parameters.

```
params := s3.PutBucketWebsiteInput{
    Bucket: aws.String(bucket),
    WebsiteConfiguration: &s3.WebsiteConfiguration{
        IndexDocument: &s3.IndexDocument{
            Suffix: aws.String(indexSuffix),
        },
    },
}

// Add the error page if set on CLI
if len(errorPage) > 0 {
    params.WebsiteConfiguration.ErrorDocument = &s3.ErrorDocument{
        Key: aws.String(errorPage),
    }
```

```
}
```

Call `PutBucketWebsite`, passing in the parameters you just defined. If an error occurs, print the errordetails and exit the routine.

```
_, err = svc.PutBucketWebsite(&params)
if err != nil {
    exitErrorf("Unable to set bucket %q website configuration, %v",
        bucket, err)
}

fmt.Printf("Successfully set bucket %q website configuration\n", bucket)
```

Delete Website Configuration on a Bucket

Create a Go file named `s3_delete_bucket_website.go`. Import the relevant Go and AWS SDK for Go packages.

```
import (
    "github.com/aws/aws-sdk-go/aws"
    "github.com/aws/aws-sdk-go/aws/session"
    "github.com/aws/aws-sdk-go/service/s3"
    "fmt"
    "os"
    "path/filepath"
)
```

This routine requires you to pass in the name of the bucket for which you want to delete the website configuration.

```
if len(os.Args) != 2 {
    exitErrorf("bucket name required\nUsage: %s bucket_name",
        filepath.Base(os.Args[0]))
}

bucket := os.Args[1]
```

Initialize a session that the SDK will use to load configuration, credentials, and region information from the shared credentials file, `~/.aws/credentials`, and create a new S3 service client.

```
sess, err := session.NewSession(&aws.Config{
```

```
    Region: aws.String("us-west-2")},
)

// Create S3 service client
svc := s3.New(sess)
```

Call `DeleteBucketWebsite` and pass in the name of the bucket to complete the action.

```
_, err = svc.DeleteBucketWebsite(&s3.DeleteBucketWebsiteInput{
    Bucket: aws.String(bucket),
})
if err != nil {
    exitErrorf("Unable to delete bucket %q website configuration, %v",
        bucket, err)
}

fmt.Printf("Successfully delete bucket %q website configuration\n", bucket)
```

Working with Amazon S3 CORS Permissions

This AWS SDK for Go example shows you how to list Amazon S3 buckets and configure CORS and bucket logging. You can download complete versions of these example files from the aws-doc-sdk-examples repository on GitHub.

Scenario

In this example, a series of Go routines are used to list your Amazon S3 buckets and to configure CORS and bucket logging. The routines use the AWS SDK for Go to configure a selected Amazon S3 bucket using these methods of the Amazon S3 client class:

- GetBucketCors
- PutBucketCors

If you are unfamiliar with using CORS configuration with an Amazon S3 bucket, it's worth your time to read Cross-Origin Resource Sharing (CORS) in the Amazon S3 Developer Guide before proceeding.

Prerequisites

- You have set up and configured the AWS SDK for Go.

- You are familiar with using CORS configuration with an Amazon S3 bucket. To learn more, see Cross-Origin Resource Sharing (CORS) in the Amazon S3 Developer Guide.

Configure CORS on the Bucket

Create a new Go file named `s3_set_cors.go`. You must import the relevant Go and AWS SDK for Go packages by adding the following lines.

```
import (
    "flag"
    "fmt"
    "os"
    "strings"

    "github.com/aws/aws-sdk-go/aws"
    "github.com/aws/aws-sdk-go/aws/session"
    "github.com/aws/aws-sdk-go/service/s3"
)
```

This routine configures CORS rules for a bucket by setting the allowed HTTP methods.

It requires the bucket name and can also take a space-separated list of HTTP methods. Using the Go Language's `flag` package, it parses the input and validates the bucket name.

```
bucketPtr := flag.String("b", "", "Bucket to set CORS on, (required)")

flag.Parse()

if *bucketPtr == "" {
    exitErrorf("-b <bucket> Bucket name required")
}

methods := filterMethods(flag.Args())
```

Notice the helper method, `filterMethods`, which ensures the methods passed in are uppercase.

```
func filterMethods(methods []string) []string {
    filtered := make([]string, 0, len(methods))
    for _, m := range methods {
        v := strings.ToUpper(m)
```

```
        switch v {
        case "POST", "GET", "PUT", "PATCH", "DELETE":
            filtered = append(filtered, v)
        }
    }

    return filtered
}
```

Initialize a session that the SDK will use to load credentials, from the shared credentials file, `~/.aws/credentials`, and create a new S3 service client.

```
sess := session.Must(session.NewSessionWithOptions(session.Options{
    SharedConfigState: session.SharedConfigEnable,
}))

svc := s3.New(sess)
```

Create a new CORSRule to set up the CORS configuration.

```
rule := s3.CORSRule{
    AllowedHeaders: aws.StringSlice([]string{"Authorization"}),
    AllowedOrigins: aws.StringSlice([]string{"*"}),
    MaxAgeSeconds:  aws.Int64(3000),

    // Add HTTP methods CORS request that were specified in the CLI.
    AllowedMethods: aws.StringSlice(methods),
}
```

Add the `CORSRule` to the `PutBucketCorsInput` structure, call `PutBucketCors` with that structure, and print a success or error message.

```
params := s3.PutBucketCorsInput{
    Bucket: bucketPtr,
    CORSConfiguration: &s3.CORSConfiguration{
        CORSRules: []*s3.CORSRule{&rule},
    },
}

_, err := svc.PutBucketCors(&params)
if err != nil {
```

```
    // Print the error message
    exitErrorf("Unable to set Bucket %q's CORS, %v", *bucketPtr, err)
}

// Print the updated CORS config for the bucket
fmt.Printf("Updated bucket %q CORS for %v\n", *bucketPtr, methods)
```

The example uses the following error function.

```
func exitErrorf(msg string, args ...interface{}) {
    fmt.Fprintf(os.Stderr, msg+"\n", args...)
    os.Exit(1)
}
```

Working with Amazon S3 Bucket Policies

This AWS SDK for Go example shows you how to retrieve, display, and set Amazon S3 bucket polices. You can download complete versions of these example files from the aws-doc-sdk-examples repository on GitHub.

Scenario

In this example, you use a series of Go routines to retrieve or set a bucket policy on an Amazon S3 bucket. The routines use the AWS SDK for Go to configure policy for a selected Amazon S3 bucket using these methods of the Amazon S3 client class:

- GetBucketPolicy
- PutBucketPolicy
- DeleteBucketPolicy

For more information about bucket policies for Amazon S3 buckets, see Using Bucket Policies and User Policies in the Amazon S3 Developer Guide.

Prerequisites

- You have set up, and configured the AWS SDK for Go.
- You are familiar with Amazon S3 bucket polices. To learn more, see Using Bucket Policies and User Policies in the Amazon S3 Developer Guide.

Retrieve and Display a Bucket Policy

Create a new Go file named `s3_get_bucket_policy.go`. You must import the relevant Go and AWS SDK for Go packages by adding the following lines.

```
import (
    "github.com/aws/aws-sdk-go/aws"
    "github.com/aws/aws-sdk-go/aws/awserr"
    "github.com/aws/aws-sdk-go/aws/session"
    "github.com/aws/aws-sdk-go/service/s3"
    "bytes"
    "encoding/json"
    "fmt"
    "os"
    "path/filepath"
)
```

Create the `exitError` function to deal with errors.

```
func exitErrorf(msg string, args ...interface{}) {
    fmt.Fprintf(os.Stderr, msg+"\n", args...)
    os.Exit(1)
}
```

This routine prints the policy for a bucket. If the bucket doesn't exist, or there was an error, an error message is printed instead. It requires the bucket name as input.

```
if len(os.Args) != 2 {
    exitErrorf("bucket name required\nUsage: %s bucket_name",
        filepath.Base(os.Args[0]))
}

bucket := os.Args[1]
```

Initialize a session that the SDK will use to load credentials from the shared credentials file, `~/.aws/credentials`, and create a new S3 service client.

```
sess, err := session.NewSession(&aws.Config{
    Region: aws.String("us-west-2")},
)
```

```
svc := s3.New(sess)
```

Call GetBucketPolicy to fetch the policy, then display any errors.

```
result, err := svc.GetBucketPolicy(&s3.GetBucketPolicyInput{
    Bucket: aws.String(bucket),
})
if err != nil {
    // Special error handling for the when the bucket doesn't
    // exists so we can give a more direct error message from the CLI.
    if aerr, ok := err.(awserr.Error); ok {
        switch aerr.Code() {
        case s3.ErrCodeNoSuchBucket:
            exitErrorf("Bucket %q does not exist.", bucket)
        case "NoSuchBucketPolicy":
            exitErrorf("Bucket %q does not have a policy.", bucket)
        }
    }
    exitErrorf("Unable to get bucket %q policy, %v.", bucket, err)
}
```

Use Go's JSON package to print the Policy JSON returned by the call.

```
out := bytes.Buffer{}
policyStr := aws.StringValue(result.Policy)
if err := json.Indent(&out, []byte(policyStr), "", "  "); err != nil {
    exitErrorf("Failed to pretty print bucket policy, %v.", err)
}

fmt.Printf("%q's Bucket Policy:\n", bucket)
fmt.Println(out.String())
```

See the complete example on GitHub.

Set Bucket Policy

This routine sets the policy for a bucket. If the bucket doesn't exist, or there was an error, an error message will be printed instead. It requires the bucket name as input. It also requires the same Go and AWS SDK for Go packages as the previous example, except for the `bytes` Go package.

```
import (
```

```
    "github.com/aws/aws-sdk-go/aws"
    "github.com/aws/aws-sdk-go/aws/awserr"
    "github.com/aws/aws-sdk-go/aws/session"
    "github.com/aws/aws-sdk-go/service/s3"
    "encoding/json"
    "fmt"
    "os"
    "path/filepath"
)
```

Add the main function and parse the arguments to get the bucket name.

```
func main() {
    if len(os.Args) != 2 {
        exitErrorf("bucket name required\nUsage: %s bucket_name",
            filepath.Base(os.Args[0]))
    }
    bucket := os.Args[1]
```

Initialize a session that the SDK will use to load configuration, credentials, and region information from the shared credentials file, `~/.aws/credentials`, and create a new S3 service client.

```
    sess, err := session.NewSession(&aws.Config{
        Region: aws.String("us-west-2")},
    )

    svc := s3.New(sess)
```

Create a policy using the map interface, filling in the bucket as the resource.

```
    readOnlyAnonUserPolicy := map[string]interface{}{
        "Version": "2012-10-17",
        "Statement": []map[string]interface{}{
            {
                "Sid":       "AddPerm",
                "Effect":    "Allow",
                "Principal": "*",
                "Action": []string{
                    "s3:GetObject",
                },
                "Resource": []string{
                    fmt.Sprintf("arn:aws:s3:::%s/*", bucket),
```

```
                },
            },
        },
    }
```

Use Go's JSON package to marshal the policy into a JSON value so that it can be sent to S3.

```
    policy, err := json.Marshal(readOnlyAnonUserPolicy)
```

Call the S3 client's PutBucketPolicy to PUT the policy for the bucket and print the results.

```
    _, err = svc.PutBucketPolicy(&s3.PutBucketPolicyInput{
        Bucket: aws.String(bucket),
        Policy: aws.String(string(policy)),
    })

    fmt.Printf("Successfully set bucket %q's policy\n", bucket)
```

The `exitError` function is used to deal with printing any errors.

```
func exitErrorf(msg string, args ...interface{}) {
    fmt.Fprintf(os.Stderr, msg+"\n", args...)
    os.Exit(1)
}
```

See the complete example on GitHub.

Working with Amazon S3 Bucket ACLs

The following examples use AWS SDK for Go functions to:

- Get the access control lists (ACLs) on a bucket
- Get the ACLs on a bucket item
- Add a new user to the ACLs on a bucket
- Add a new user to the ACLs on a bucket item

You can download complete versions of these example files from the aws-doc-sdk-examples repository on GitHub.

Scenario

In these examples, a series of Go routines are used to manage ACLs on your Amazon S3 buckets. The routines use the AWS SDK for Go to perform Amazon S3 bucket operations using the following methods of the Amazon S3 client class:

- GetBucketAcl
- GetObjectAcl
- PutBucketAcl
- PutObjectAcl

Prerequisites

- You have set up, and configured the AWS SDK for Go.
- You are familiar with Amazon S3 bucket ACLs. To learn more, see Managing Access with ACLs in the Amazon S3 Developer Guide.

Get a Bucket ACL

The GetBucketAcl function gets the ACLs on a bucket.

The following example gets the ACLs on a bucket with the name specified as a command line argument.

Create the file *s3_get_bucket_acl.go*. Add the following statements to import the Go and AWS SDK for Go packages used in the example.

```
import (
    "github.com/aws/aws-sdk-go/aws/session"
    "github.com/aws/aws-sdk-go/service/s3"

    "fmt"
    "os"
)
```

Create a function to display errors and exit.

```
func exitErrorf(msg string, args ...interface{}) {
    fmt.Fprintf(os.Stderr, msg+"\n", args...)
```

```
    os.Exit(1)
}
```

This example requires one input parameter, the name of the bucket. If the name is not supplied, call the error function and exit.

```
if len(os.Args) != 2 {
    exitErrorf("Bucket name required\nUsage: go run", os.Args[0], "BUCKET")
}

bucket := os.Args[1]
```

Initialize the session that the SDK uses to load credentials from the shared credentials file `~/.aws/credentials`, the region from the shared configuration file `~/.aws/config`, and create a new Amazon S3 service client.

```
sess := session.Must(session.NewSessionWithOptions(session.Options{
    SharedConfigState: session.SharedConfigEnable,
}))

// Create S3 service client
svc := s3.New(sess)
```

Call GetBucketAcl, passing in the name of the bucket. If an error occurs, call `exitErrorf`. If no error occurs, loop through the results and print out the name, type, and permission for the grantees.

```
result, err := svc.GetBucketAcl(&s3.GetBucketAclInput{Bucket: &bucket})
if err != nil {
    exitErrorf(err.Error())
}

fmt.Println("Owner:", *result.Owner.DisplayName)
fmt.Println("")
fmt.Println("Grants")

for _, g := range result.Grants {
    // If we add a canned ACL, the name is nil
    if g.Grantee.DisplayName == nil {
        fmt.Println("  Grantee:    EVERYONE")
    } else {
```

```
        fmt.Println("  Grantee:   ", *g.Grantee.DisplayName)
    }

    fmt.Println("  Type:      ", *g.Grantee.Type)
    fmt.Println("  Permission:", *g.Permission)
    fmt.Println("")
}
```

See the complete example on GitHub.

Set a Bucket ACL

The PutBucketAcl function sets the ACLs on a bucket.

The following example gives a user access by email address to a bucket with the bucket name and email address specified as command line arguments. The user can also supply a permission argument. However, if it isn'o't supplied, the user is given READ access to the bucket.

Create the file *s3_put_bucket_acl.go*. Add the following statements to import the Go and AWS SDK for Go packages used in the example.

```
import (
    "github.com/aws/aws-sdk-go/aws/session"
    "github.com/aws/aws-sdk-go/service/s3"

    "fmt"
    "os"
)
```

Create a function to display errors and exit.

```
func exitErrorf(msg string, args ...interface{}) {
    fmt.Fprintf(os.Stderr, msg+"\n", args...)
    os.Exit(1)
}
```

Get the two required input parameters. If the optional permission parameter is supplied, make sure it is one of the allowed values. If not, print an error message and quit.

```
if len(os.Args) < 3 {
    exitErrorf("Bucket name and email address required; permission optional (READ if
 omitted)\nUsage: go run", os.Args[0], "BUCKET EMAIL [PERMISSION]")
```

```
}

bucket := os.Args[1]
address := os.Args[2]

permission := "READ"

if len(os.Args) == 4 {
    permission = os.Args[3]

    if !(permission == "FULL_CONTROL" || permission == "WRITE" || permission ==
 "WRITE_ACP" || permission == "READ" || permission == "READ_ACP") {
        fmt.Println("Illegal permission value. It must be one of:")
        fmt.Println("FULL_CONTROL, WRITE, WRITE_ACP, READ, or READ_ACP")
        os.Exit(1)

    }
}
```

Initialize the session that the SDK uses to load credentials from the shared credentials file `~/.aws/credentials`, the region from the shared configuration file `~/.aws/config`, and create a new Amazon S3 service client.

```
sess := session.Must(session.NewSessionWithOptions(session.Options{
    SharedConfigState: session.SharedConfigEnable,
}))

// Create S3 service client
svc := s3.New(sess)
```

Get the existing ACLs and append the new user to the list. If there is an error while retrieving the list, print an error message and quit.

```
result, err := svc.GetBucketAcl(&s3.GetBucketAclInput{Bucket: &bucket})
if err != nil {
    exitErrorf(err.Error())
}

owner := *result.Owner.DisplayName
ownerId := *result.Owner.ID

// Existing grants
```

```
grants := result.Grants

// Create new grantee to add to grants
var newGrantee = s3.Grantee{EmailAddress: &address, Type: &userType}
var newGrant = s3.Grant{Grantee: &newGrantee, Permission: &permission}

// Add them to the grants
grants = append(grants, &newGrant)
```

Build the parameter list for the call based on the existing ACLs and the new user information.

```
params := &s3.PutBucketAclInput{
    Bucket: &bucket,
    AccessControlPolicy: &s3.AccessControlPolicy{
        Grants: grants,
        Owner: &s3.Owner{
            DisplayName: &owner,
            ID:          &ownerId,
        },
    },
}
```

Call PutBucketAcl, passing in the parameter list. If an error occurs, display a message and quit. Otherwise, display a message indicating success.

```
_, err = svc.PutBucketAcl(params)
if err != nil {
    exitErrorf(err.Error())
}

fmt.Println("Congratulations. You gave user with email address", address, permission,
 "permission to bucket", bucket)
```

See the complete example on GitHub.

Making a Bucket Public using a Canned ACL

As in the previous example, the PutBucketAcl function sets the ACLs on a bucket.

The following example gives everyone read-only access to the items in the bucket with the bucket name specified as a command line argument.

Create the file *s3_make_bucket_public.go*. Add the following statements to import the Go and AWS SDK for Go packages used in the example.

```
import (
    "github.com/aws/aws-sdk-go/aws/session"
    "github.com/aws/aws-sdk-go/service/s3"

    "fmt"
    "os"
)
```

Create a function to display errors and exit.

```
func exitErrorf(msg string, args ...interface{}) {
    fmt.Fprintf(os.Stderr, msg+"\n", args...)
    os.Exit(1)
}
```

Get the required input parameter and create the **acl**.

```
if len(os.Args) < 2 {
    exitErrorf("Bucket name required.\nUsage: go run", os.Args[0], "BUCKET")
}

bucket := os.Args[1]
acl := "public-read"
```

Initialize the session that the SDK uses to load credentials from the shared credentials file `~/.aws/credentials` the region from the shared configuration file `~/.aws/config`, and create a new Amazon S3 service client.

```
sess := session.Must(session.NewSessionWithOptions(session.Options{
    SharedConfigState: session.SharedConfigEnable,
}))
svc := s3.New(sess)
```

Create the input for and call PutBucketAcl. If an error occurs, display a message and quit. Otherwise, display a message indicating success.

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

```
    Bucket: &bucket,
    ACL: &acl,
}

// Set bucket ACL
_, err := svc.PutBucketAcl(params)
if err != nil {
    exitErrorf(err.Error())
}

fmt.Println("Bucket " + bucket + " is now public")
```

See the complete example on GitHub.

Get a Bucket Object ACL

The PutObjectAcl function sets the ACLs on a bucket item.

The following example gets the ACLs for a bucket item with the bucket and item name specified as command line arguments.

Create the file *s3_get_bucket_object_acl.go*. Add the following statements to import the Go and AWS SDK for Go packages used in the example.

```
import (
    "github.com/aws/aws-sdk-go/aws/session"
    "github.com/aws/aws-sdk-go/service/s3"

    "fmt"
    "os"
)
```

Create a function to display errors and exit.

```
func exitErrorf(msg string, args ...interface{}) {
    fmt.Fprintf(os.Stderr, msg+"\n", args...)
    os.Exit(1)
}
```

This example requires two input parameters, the names of the bucket and object. If either name is not supplied, call the error function and exit.

```
if len(os.Args) != 3 {
    exitErrorf("Bucket and object names required\nUsage: go run", os.Args[0], "BUCKET
 OBJECT")
}

bucket := os.Args[1]
key := os.Args[2]
```

Initialize the session that the SDK uses to load credentials from the shared credentials file `~/.aws/credentials`, the region from the shared configuration file `~/.aws/config`, and create a new Amazon S3 service client.

```
sess := session.Must(session.NewSessionWithOptions(session.Options{
    SharedConfigState: session.SharedConfigEnable,
}))

// Create S3 service client
svc := s3.New(sess)
```

Call GetObjectAcl, passing in the names of the bucket and object. If an error occurs, call `exitErrorf`. If no error occurs, loop through the results and print out the name, type, and permission for the grantees.

```
result, err := svc.GetObjectAcl(&s3.GetObjectAclInput{Bucket: &bucket, Key: &key})
if err != nil {
    exitErrorf(err.Error())
}

fmt.Println("Owner:", *result.Owner.DisplayName)
fmt.Println("")
fmt.Println("Grants")

for _, g := range result.Grants {
    fmt.Println("  Grantee:   ", *g.Grantee.DisplayName)
    fmt.Println("  Type:      ", *g.Grantee.Type)
    fmt.Println("  Permission:", *g.Permission)
    fmt.Println("")
}
```

See the complete example on GitHub.

Set a Bucket Object ACL

The PutObjectAcl function sets the ACLs on a bucket item.

The following example gives a user access by email address to a bucket item, with the bucket and item names and email address specified as command line arguments. The user can also supply a permission argument. However, if it isn't supplied, the user is given READ access to the bucket.

Create the file *s3_put_bucket_object_acl.go*. Add the following statements to import the Go and AWS SDK for Go packages used in the example.

```
import (
    "github.com/aws/aws-sdk-go/aws/session"
    "github.com/aws/aws-sdk-go/service/s3"

    "fmt"
    "os"
)
```

Create a function to display errors and exit.

```
func exitErrorf(msg string, args ...interface{}) {
    fmt.Fprintf(os.Stderr, msg+"\n", args...)
    os.Exit(1)
}
```

Get the three required input parameters. If the optional permission parameter is supplied, make sure it is one of the allowed values. If not, print an error message and quit.

```
if len(os.Args) < 4 {
    exitErrorf("Bucket name, object name, and email address required; permission
 optional (READ if omitted)\nUsage: go run", os.Args[0], "BUCKET OBJECT EMAIL
 [PERMISSION]")
}

bucket := os.Args[1]
key := os.Args[2]
address := os.Args[3]

permission := "READ"
```

```
if len(os.Args) == 5 {
    permission = os.Args[4]

    if !(permission == "FULL_CONTROL" || permission == "WRITE" || permission ==
 "WRITE_ACP" || permission == "READ" || permission == "READ_ACP") {
        fmt.Println("Illegal permission value. It must be one of:")
        fmt.Println("FULL_CONTROL, WRITE, WRITE_ACP, READ, or READ_ACP")
        os.Exit(1)
    }
}
```

Initialize the session that the SDK uses to load credentials from the shared credentials file `~/.aws/credentials`, and create a new Amazon S3 service client.

```
sess := session.Must(session.NewSessionWithOptions(session.Options{
    SharedConfigState: session.SharedConfigEnable,
}))

// Create S3 service client
svc := s3.New(sess)
```

Build the parameter list for the call based on the existing ACLs and the new user information.

```
result, err := svc.GetObjectAcl(&s3.GetObjectAclInput{Bucket: &bucket, Key: &key})
if err != nil {
    exitErrorf(err.Error())
}

owner := *result.Owner.DisplayName
ownerId := *result.Owner.ID

// Existing grants
grants := result.Grants

// Create new grantee to add to grants
userType := "AmazonCustomerByEmail"
var newGrantee = s3.Grantee{EmailAddress: &address, Type: &userType}
var newGrant = s3.Grant{Grantee: &newGrantee, Permission: &permission}

// Add them to the grants
grants = append(grants, &newGrant)

params := &s3.PutObjectAclInput{
```

```
    Bucket: &bucket,
    Key:    &key,
    AccessControlPolicy: &s3.AccessControlPolicy{
        Grants: grants,
        Owner: &s3.Owner{
            DisplayName: &owner,
            ID:          &ownerId,
        },
    },
}
```

Call PutObjectAcl, passing in the parameter list. If an error occurs, display a message and quit. Otherwise, display a message indicating success.

```
_, err = svc.PutObjectAcl(params)
if err != nil {
    exitErrorf(err.Error())
}

fmt.Println("Congratulations. You gave user with email address", address, permission,
 "permission to bucket", bucket, "object", key)
```

See the complete example on GitHub.

Encrypting Amazon S3 Bucket Items

Amazon S3 supports encrypting Amazon S3 bucket objects on both the client and the server. To encrypt objects on the client, you perform the encryption yourself, either using keys that you create or keys that AWS Key Management Service (AWS KMS) manages for you.

To encrypt objects on the server, you have more options.

- You can have Amazon S3 automatically encrypt objects as you upload them to a bucket. Once you configure a bucket with this option, every object that you upload–from that point on–is encrypted.
- You can encrypt objects yourself before you upload them to a bucket. The disadvantage with this approach is that you can still upload objects that are not encrypted. We don't show this approach in the documentation.
- You can have Amazon S3 reject objects that you attempt to upload to a bucket without requesting Amazon S3 encrypt the items.

The following examples describe some of these options.

Learn about encryption in Amazon S3 at Protecting Data Using Encryption.

Topics

- Setting Default Server-Side Encryption for an Amazon S3 Bucket
- Requiring Encryption on the Server to Upload Amazon S3 Bucket Objects
- Encrypting an Amazon S3 Bucket Object on the Server Using AWS KMS

Setting Default Server-Side Encryption for an Amazon S3 Bucket

The following example uses the PutBucketEncryption method to enable KMS server-side encryption on any objects added to `amzn-s3-demo-bucket`.

The only exception is if the user configures their request to explicitly use server-side encryption. In that case, the specified encryption takes precedence.

Choose Copy to save the code locally.

Create the file *set_default_encryption.go*.

Import the required packages.

```
import (
    "github.com/aws/aws-sdk-go/aws"
    "github.com/aws/aws-sdk-go/aws/session"
    "github.com/aws/aws-sdk-go/service/s3"

    "fmt"
    "os"
)
```

Get the KMS key from the command line, where `key` is a KMS key ID as created in the Creating a CMK in AWS Key Management Service example, and set the bucket name.

```
if len(os.Args) != 2 {
    fmt.Println("You must supply a key")
    os.Exit(1)
}

key := os.Args[1]
```

```
bucket := "amzn-s3-demo-bucket"
```

Create a session and Amazon S3 client.

```
sess := session.Must(session.NewSessionWithOptions(session.Options{
    SharedConfigState: session.SharedConfigEnable,
}))

svc := s3.New(sess)
```

Create the input for and call `PutBucketEncryption`, and display a success message.

```
defEnc := &s3.ServerSideEncryptionByDefault{KMSMasterKeyID: aws.String(key),
 SSEAlgorithm: aws.String(s3.ServerSideEncryptionAwsKms)}
rule := &s3.ServerSideEncryptionRule{ApplyServerSideEncryptionByDefault: defEnc}
rules := []*s3.ServerSideEncryptionRule{rule}
serverConfig := &s3.ServerSideEncryptionConfiguration{Rules: rules}
input := &s3.PutBucketEncryptionInput{Bucket: aws.String(bucket),
 ServerSideEncryptionConfiguration: serverConfig}
_, err := svc.PutBucketEncryption(input)
fmt.Println("Bucket " + bucket + " now has KMS encryption by default")
```

See the complete example on GitHub.

Requiring Encryption on the Server to Upload Amazon S3 Bucket Objects

The following example uses the PutBucketPolicy method to require that objects uploaded to an Amazon S3 bucket have Amazon S3 encrypt the object with an AWS KMS key. Attempts to upload an object without specifying that Amazon S3 encrypt the object with an AWS KMS key raise an `Aws::S3::Errors::AccessDenied` exception.

Avoid using this configuration option if you use default server-side encryption as described in Setting Default Server-Side Encryption for an Amazon S3 Bucket as they could conflict and result in unexpected results.

Choose Copy to save the code locally.

Create the file *require_server_encryption.go*.

Import the required packages.

```
import (
```

```
    "github.com/aws/aws-sdk-go/aws"
    "github.com/aws/aws-sdk-go/aws/session"
    "github.com/aws/aws-sdk-go/service/s3"

    "fmt"
    "os"
    "encoding/json"
)
```

Set the name of the bucket, create a session, and create an Amazon S3 client.

```
bucket := "amzn-s3-demo-bucket"
sess := session.Must(session.NewSessionWithOptions(session.Options{
    SharedConfigState: session.SharedConfigEnable,
}))

svc := s3.New(sess)
```

Create an Amazon S3 policy that requires server-side KMS encryption on objects uploaded to the bucket.

```
PolicyDoc := map[string]interface{}{
    "Version": "2012-10-17",
    "Statement": []map[string]interface{}{
        {
            "Sid": "DenyIncorrectEncryptionHeader",
            "Effect": "Deny",
            "Principal": "*",
            "Action": "s3:PutObject",
            "Resource": "arn:aws:s3:::" + bucket + "/*",
            "Condition": map[string]interface{}{
                "StringNotEquals": map[string]interface{}{
                    "s3:x-amz-server-side-encryption": "aws:kms",
                },
            },
        },
        {
            "Sid": "DenyUnEncryptedObjectUploads",
            "Effect": "Deny",
            "Principal": "*",
            "Action": "s3:PutObject",
            "Resource": "arn:aws:s3:::" + bucket + "/*",
```

```
            "Condition": map[string]interface{}{
                "Null": map[string]interface{}{
                    "s3:x-amz-server-side-encryption": "true",
                },
            },
        },
    },
}
```

Convert the policy into JSON, create the input for and call `PutBucketPolicy`, apply the policy to the bucket, and print a success message.

```
policy, err := json.Marshal(PolicyDoc)
input := &s3.PutBucketPolicyInput{
    Bucket: aws.String(bucket),
    Policy: aws.String(string(policy)),
}
_, err = svc.PutBucketPolicy(input)
fmt.Println("Set policy for " + bucket)
```

See the complete example on GitHub.

Encrypting an Amazon S3 Bucket Object on the Server Using AWS KMS

The following example uses the PutObject method to add the object `myItem` to the bucket `amzn-s3-demo-bucket` with server-side encryption set to AWS KMS.

Note that this differs from Setting Default Server-Side Encryption for an Amazon S3 Bucket, is in that case, the objects are encrypted without you having to explicitly perform the operation.

Choose Copy to save the code locally.

Create the file *encrypt_object_on_server.go*.

Add the required packages.

```
import (
    "github.com/aws/aws-sdk-go/aws"
    "github.com/aws/aws-sdk-go/aws/session"
    "github.com/aws/aws-sdk-go/service/s3"

    "fmt"
```

```
    "os"
    "strings"
)
```

Get the KMS key from the command line, where `key` is a KMS key ID as created in the Creating a CMK in AWS Key Management Service example, and set the bucket and object names.

```
if len(os.Args) != 2 {
    fmt.Println("You must supply a key")
    os.Exit(1)
}

key := os.Args[1]
bucket := "amzn-s3-demo-bucket"
object := "myItem"
```

Create a session and Amazon S3 client.

```
sess := session.Must(session.NewSessionWithOptions(session.Options{
    SharedConfigState: session.SharedConfigEnable,
}))

svc := s3.New(sess)
```

Create input for and call `put_object`. Notice that the `server_side_encryption` property is set to `aws:kms`, indicating that Amazon S3 encrypts the object using AWS KMS, and display a success message to the user.

```
input := &s3.PutObjectInput{
    Body:                 strings.NewReader(object),
    Bucket:               aws.String(bucket),
    Key:                  aws.String(object),
    ServerSideEncryption: aws.String("aws:kms"),
    SSEKMSKeyId:          aws.String(key),
}

_, err := svc.PutObject(input)
fmt.Println("Added object " + object + " to bucket " + bucket + " with AWS KMS
 encryption")
```

See the complete example on GitHub.

Amazon SES Examples Using the AWS SDK for Go

Amazon Simple Email Service (Amazon SES) is an email platform that provides an easy, cost-effective way for you to send and receive email using your own email addresses and domains. You can use the following examples to access Amazon SES using the AWS SDK for Go. For more information about Amazon SES, see the Amazon SES documentation.

Topics

- Listing Valid Amazon SES Email Addresses
- Verifying an Email Address in Amazon SES
- Sending a Message to an Email Address in Amazon SES
- Deleting an Email Address in Amazon SES
- Getting Amazon SES Statistics

Listing Valid Amazon SES Email Addresses

The following example demonstrates how to use the AWS SDK for Go to list the valid Amazon SES email addresses.

```
package main

import (
    "fmt"
    "os"

    "github.com/aws/aws-sdk-go/aws"
    "github.com/aws/aws-sdk-go/aws/session"
    "github.com/aws/aws-sdk-go/service/ses"
)

func main() {
    // Initialize a session in us-west-2 that the SDK will use to load
    // credentials from the shared credentials file ~/.aws/credentials.
    sess, err := session.NewSession(&aws.Config{
        Region: aws.String("us-west-2")},
    )

    // Create SES service client
    svc := ses.New(sess)
```

```
    result, err := svc.ListIdentities(&ses.ListIdentitiesInput{IdentityType:
 aws.String("EmailAddress")})

    if err != nil {
        fmt.Println(err)
        os.Exit(1)
    }

    for _, email := range result.Identities {
        var e = []*string{email}

        verified, err :=
 svc.GetIdentityVerificationAttributes(&ses.GetIdentityVerificationAttributesInput{Identities:
 e})

        if err != nil {
            fmt.Println(err)
            os.Exit(1)
        }

        for _, va := range verified.VerificationAttributes {
            if *va.VerificationStatus == "Success" {
                fmt.Println(*email)
            }
        }
    }
}
```

See the complete example on GitHub.

Verifying an Email Address in Amazon SES

The following example demonstrates how to use the AWS SDK for Go to verify an Amazon SES email address.

```
package main

import (
    "fmt"

    //go get -u github.com/aws/aws-sdk-go
    "github.com/aws/aws-sdk-go/aws"
```

```
    "github.com/aws/aws-sdk-go/aws/session"
    "github.com/aws/aws-sdk-go/service/ses"
    "github.com/aws/aws-sdk-go/aws/awserr"
)

const (
    // Replace sender@example.com with your "From" address.
    // This address must be verified with Amazon SES.
    Sender = "sender@example.com"

    // Replace recipient@example.com with a "To" address. If your account
    // is still in the sandbox, this address must be verified.
    Recipient = "recipient@example.com"
)

func main() {
    // Create a new session in the us-west-2 region.
    // Replace us-west-2 with the AWS Region you're using for Amazon SES.
    sess, err := session.NewSession(&aws.Config{
        Region:aws.String("us-west-2")},
    )

    // Create an SES session.
    svc := ses.New(sess)

    // Attempt to send the email.
    _, err = svc.VerifyEmailAddress(&ses.VerifyEmailAddressInput{EmailAddress:
 aws.String(Recipient)})

    // Display error messages if they occur.
    if err != nil {
        if aerr, ok := err.(awserr.Error); ok {
            switch aerr.Code() {
            case ses.ErrCodeMessageRejected:
                fmt.Println(ses.ErrCodeMessageRejected, aerr.Error())
            case ses.ErrCodeMailFromDomainNotVerifiedException:
                fmt.Println(ses.ErrCodeMailFromDomainNotVerifiedException,
 aerr.Error())
            case ses.ErrCodeConfigurationSetDoesNotExistException:
                fmt.Println(ses.ErrCodeConfigurationSetDoesNotExistException,
 aerr.Error())
            default:
                fmt.Println(aerr.Error())
            }
```

```
        } else {
            // Print the error, cast err to awserr.Error to get the Code and
            // Message from an error.
            fmt.Println(err.Error())
        }

        return
    }

    fmt.Println("Verification sent to address: " + Recipient)
}
```

See the complete example on GitHub.

Sending a Message to an Email Address in Amazon SES

The following example demonstrates how to use the AWS SDK for Go to send a message to an Amazon SES email address.

```
package main

import (
    "fmt"

    //go get -u github.com/aws/aws-sdk-go
    "github.com/aws/aws-sdk-go/aws"
    "github.com/aws/aws-sdk-go/aws/session"
    "github.com/aws/aws-sdk-go/service/ses"
    "github.com/aws/aws-sdk-go/aws/awserr"
)

const (
    // Replace sender@example.com with your "From" address.
    // This address must be verified with Amazon SES.
    Sender = "sender@example.com"

    // Replace recipient@example.com with a "To" address. If your account
    // is still in the sandbox, this address must be verified.
    Recipient = "recipient@example.com"

    // Specify a configuration set. To use a configuration
    // set, comment the next line and line 92.
    //ConfigurationSet = "ConfigSet"
```

```
    // The subject line for the email.
    Subject = "Amazon SES Test (AWS SDK for Go)"

    // The HTML body for the email.
    HtmlBody =  "<h1>Amazon SES Test Email (AWS SDK for Go)</h1><p>This email was sent
 with " +
                "<a href='https://aws.amazon.com/ses/'>Amazon SES</a> using the " +
                "<a href='https://aws.amazon.com/sdk-for-go/'>AWS SDK for Go</a>.</p>"

    //The email body for recipients with non-HTML email clients.
    TextBody = "This email was sent with Amazon SES using the AWS SDK for Go."

    // The character encoding for the email.
    CharSet = "UTF-8"
)

func main() {
    // Create a new session in the us-west-2 region.
    // Replace us-west-2 with the AWS Region you're using for Amazon SES.
    sess, err := session.NewSession(&aws.Config{
        Region:aws.String("us-west-2")},
    )

    // Create an SES session.
    svc := ses.New(sess)

    // Assemble the email.
    input := &ses.SendEmailInput{
        Destination: &ses.Destination{
            CcAddresses: []*string{
            },
            ToAddresses: []*string{
                aws.String(Recipient),
            },
        },
        Message: &ses.Message{
            Body: &ses.Body{
                Html: &ses.Content{
                    Charset: aws.String(CharSet),
                    Data:    aws.String(HtmlBody),
                },
                Text: &ses.Content{
                    Charset: aws.String(CharSet),
```

```
                    Data:    aws.String(TextBody),
                },
            },
            Subject: &ses.Content{
                Charset: aws.String(CharSet),
                Data:    aws.String(Subject),
            },
        },
        Source: aws.String(Sender),
            // Uncomment to use a configuration set
            //ConfigurationSetName: aws.String(ConfigurationSet),
    }

    // Attempt to send the email.
    result, err := svc.SendEmail(input)

    // Display error messages if they occur.
    if err != nil {
        if aerr, ok := err.(awserr.Error); ok {
            switch aerr.Code() {
            case ses.ErrCodeMessageRejected:
                fmt.Println(ses.ErrCodeMessageRejected, aerr.Error())
            case ses.ErrCodeMailFromDomainNotVerifiedException:
                fmt.Println(ses.ErrCodeMailFromDomainNotVerifiedException,
 aerr.Error())
            case ses.ErrCodeConfigurationSetDoesNotExistException:
                fmt.Println(ses.ErrCodeConfigurationSetDoesNotExistException,
 aerr.Error())
            default:
                fmt.Println(aerr.Error())
            }
        } else {
            // Print the error, cast err to awserr.Error to get the Code and
            // Message from an error.
            fmt.Println(err.Error())
        }

        return
    }

    fmt.Println("Email Sent to address: " + Recipient)
    fmt.Println(result)
}
```

See the complete example on GitHub.

Deleting an Email Address in Amazon SES

The following example demonstrates how to use the AWS SDK for Go to delete an Amazon SES email address.

```
package main

import (
    "fmt"
    "os"

    //go get -u github.com/aws/aws-sdk-go
    "github.com/aws/aws-sdk-go/aws"
    "github.com/aws/aws-sdk-go/aws/session"
    "github.com/aws/aws-sdk-go/service/ses"
)

const (
    // Replace sender@example.com with your "From" address
    Sender = "sender@example.com"

    // Replace recipient@example.com with a "To" address
    Recipient = "recipient@example.com"
)

func main() {
    // Create a new session in the us-west-2 region
    // Replace us-west-2 with the AWS Region you're using for Amazon SES
    sess, err := session.NewSession(&aws.Config{
        Region:aws.String("us-west-2")},
    )

    if err != nil {
        fmt.Println("Got error creating SES session:")
        fmt.Println(err.Error())
        os.Exit(1)
    }

    // Create an SES session
    svc := ses.New(sess)
```

```
    // Remove email address
    _, delErr :=
 svc.DeleteVerifiedEmailAddress(&ses.DeleteVerifiedEmailAddressInput{EmailAddress:
 aws.String(Recipient)})

    // Display error message if it occurs
    if delErr != nil {
        fmt.Println("Got error attempting to remove email address: " + Recipient)
        fmt.Println(delErr.Error())
      os.Exit(1)
    }

    // Display success message
    fmt.Println("Removed email address: " + Recipient)
}
```

See the complete example on GitHub.

Getting Amazon SES Statistics

The following example demonstrates how to use the AWS SDK for Go to get statistics about Amazon SES. Use this information to avoid damaging your reputation when emails are bounced or rejected.

```
package main

import (
    //go get -u github.com/aws/aws-sdk-go
    "github.com/aws/aws-sdk-go/aws"
    "github.com/aws/aws-sdk-go/aws/session"
    "github.com/aws/aws-sdk-go/service/ses"

    "fmt"
)

func main() {
    // Initialize a session that the SDK uses to load
    // credentials from the shared credentials file ~/.aws/credentials
    // and configuration from the shared configuration file ~/.aws/config.
    sess := session.Must(session.NewSessionWithOptions(session.Options{
        SharedConfigState: session.SharedConfigEnable,
    }))
```

```
    // Create an SES session.
    svc := ses.New(sess)

    // Attempt to send the email.
    result, err := svc.GetSendStatistics(nil)

    // Display any error message
    if err != nil {
        fmt.Println(err.Error())
        return
    }

    dps := result.SendDataPoints

    fmt.Println("Got", len(dps), "datapoints")
    fmt.Println("")

    for _, dp := range dps {
        fmt.Println("Timestamp: ", dp.Timestamp)
        fmt.Println("Attempts:  ", aws.Int64Value(dp.DeliveryAttempts))
        fmt.Println("Bounces:   ", aws.Int64Value(dp.Bounces))
        fmt.Println("Complaints:", aws.Int64Value(dp.Complaints))
        fmt.Println("Rejects:   ", aws.Int64Value(dp.Rejects))
        fmt.Println("")
    }
}
```

See the complete example on GitHub.

Amazon SNS Examples Using the AWS SDK for Go

Amazon Simple Notification Service (Amazon SNS) is a web service that enables applications, end users, and devices to instantly send and receive notifications from the cloud. You can use the following examples to access Amazon SNS using the AWS SDK for Go. For more information about Amazon SNS, see the Amazon SNS documentation.

Topics

- Listing Your Amazon SNS Topics
- Creating an Amazon SNS Topic
- List Your Amazon SNS Subscriptions
- Subscribe to an Amazon SNS Topic

- Sending a Message to All Amazon SNS Topic Subscribers

Listing Your Amazon SNS Topics

The following example lists the ARNs of your Amazon SNS topics in your default region.

```
package main

import (
    "github.com/aws/aws-sdk-go/aws/session"
    "github.com/aws/aws-sdk-go/service/sns"

    "fmt"
    "os"
)

func main() {
    // Initialize a session that the SDK will use to load
    // credentials from the shared credentials file. (~/.aws/credentials).
    sess := session.Must(session.NewSessionWithOptions(session.Options{
        SharedConfigState: session.SharedConfigEnable,
    }))

    svc := sns.New(sess)

    result, err := svc.ListTopics(nil)
    if err != nil {
        fmt.Println(err.Error())
        os.Exit(1)
    }

    for _, t := range result.Topics {
        fmt.Println(*t.TopicArn)
    }
}
```

See the complete example on GitHub.

Creating an Amazon SNS Topic

The following example creates a topic with the name from the command line, in your default region, and displays the resulting topic ARN.

```
package main

import (
    "github.com/aws/aws-sdk-go/aws"
    "github.com/aws/aws-sdk-go/aws/session"
    "github.com/aws/aws-sdk-go/service/sns"

    "fmt"
    "os"
)

func main() {
    if len(os.Args) < 2 {
        fmt.Println("You must supply a topic name")
        fmt.Println("Usage: go run SnsCreateTopic.go TOPIC")
        os.Exit(1)
    }

    // Initialize a session that the SDK will use to load
    // credentials from the shared credentials file. (~/.aws/credentials).
    sess := session.Must(session.NewSessionWithOptions(session.Options{
        SharedConfigState: session.SharedConfigEnable,
    }))

    svc := sns.New(sess)

    result, err := svc.CreateTopic(&sns.CreateTopicInput{
        Name: aws.String(os.Args[1]),
    })
    if err != nil {
        fmt.Println(err.Error())
        os.Exit(1)
    }

    fmt.Println(*result.TopicArn)
}
```

See the complete example on GitHub.

List Your Amazon SNS Subscriptions

The following example lists the ARNs for your Amazon SNS topic subscriptions and the associated topic in your default region.

```
package main

import (
    "github.com/aws/aws-sdk-go/aws"
    "github.com/aws/aws-sdk-go/aws/session"
    "github.com/aws/aws-sdk-go/service/sns"

    "flag"
    "fmt"
    "os"
)

func main() {
    emailPtr := flag.String("e", "", "The email address of the user subscribing to the
 topic")
    topicPtr := flag.String("t", "", "The ARN of the topic to which the user
 subscribes")

    flag.Parse()

    if *emailPtr == "" || *topicPtr == "" {
        fmt.Println("You must supply an email address and topic ARN")
        fmt.Println("Usage: go run SnsSubscribe.go -e EMAIL -t TOPIC-ARN")
        os.Exit(1)
    }

    // Initialize a session that the SDK will use to load
    // credentials from the shared credentials file. (~/.aws/credentials).
    sess := session.Must(session.NewSessionWithOptions(session.Options{
        SharedConfigState: session.SharedConfigEnable,
    }))

    svc := sns.New(sess)

    result, err := svc.Subscribe(&sns.SubscribeInput{
        Endpoint:              emailPtr,
        Protocol:              aws.String("email"),
        ReturnSubscriptionArn: aws.Bool(true), // Return the ARN, even if user has yet
 to confirm
        TopicArn:              topicPtr,
    })
    if err != nil {
        fmt.Println(err.Error())
```

```
        os.Exit(1)
    }

    fmt.Println(*result.SubscriptionArn)
}
```

See the complete example on GitHub.

Subscribe to an Amazon SNS Topic

The following example creates a subscription to the topic with the supplied ARN for the user with the supplied email address in your default region, and displays the resulting ARN.

```
package main

import (
    "github.com/aws/aws-sdk-go/aws"
    "github.com/aws/aws-sdk-go/aws/session"
    "github.com/aws/aws-sdk-go/service/sns"

    "flag"
    "fmt"
    "os"
)

func main() {
    emailPtr := flag.String("e", "", "The email address of the user subscribing to the
 topic")
    topicPtr := flag.String("t", "", "The ARN of the topic to which the user
 subscribes")

    flag.Parse()

    if *emailPtr == "" || *topicPtr == "" {
        fmt.Println("You must supply an email address and topic ARN")
        fmt.Println("Usage: go run SnsSubscribe.go -e EMAIL -t TOPIC-ARN")
        os.Exit(1)
    }

    // Initialize a session that the SDK will use to load
    // credentials from the shared credentials file. (~/.aws/credentials).
    sess := session.Must(session.NewSessionWithOptions(session.Options{
        SharedConfigState: session.SharedConfigEnable,
```

```
    }))

    svc := sns.New(sess)

    result, err := svc.Subscribe(&sns.SubscribeInput{
        Endpoint:              emailPtr,
        Protocol:              aws.String("email"),
        ReturnSubscriptionArn: aws.Bool(true), // Return the ARN, even if user has yet
 to confirm
        TopicArn:              topicPtr,
    })
    if err != nil {
        fmt.Println(err.Error())
        os.Exit(1)
    }

    fmt.Println(*result.SubscriptionArn)
}
```

See the complete example on GitHub.

Sending a Message to All Amazon SNS Topic Subscribers

The following example sends the message supplied on the command line to all subscribers to the Amazon SNS topic with the ARN specified on the command line.

```
package main

import (
    "github.com/aws/aws-sdk-go/aws/session"
    "github.com/aws/aws-sdk-go/service/sns"

    "flag"
    "fmt"
    "os"
)

func main() {
    msgPtr := flag.String("m", "", "The message to send to the subscribed users of the
 topic")
    topicPtr := flag.String("t", "", "The ARN of the topic to which the user
 subscribes")
```

```
    flag.Parse()

    if *msgPtr == "" || *topicPtr == "" {
        fmt.Println("You must supply a message and topic ARN")
        fmt.Println("Usage: go run SnsPublish.go -m MESSAGE -t TOPIC-ARN")
        os.Exit(1)
    }

    // Initialize a session that the SDK will use to load
    // credentials from the shared credentials file. (~/.aws/credentials).
    sess := session.Must(session.NewSessionWithOptions(session.Options{
        SharedConfigState: session.SharedConfigEnable,
    }))

    svc := sns.New(sess)

    result, err := svc.Publish(&sns.PublishInput{
        Message:  msgPtr,
        TopicArn: topicPtr,
    })
    if err != nil {
        fmt.Println(err.Error())
        os.Exit(1)
    }

    fmt.Println(*result.MessageId)
}
```

See the complete example on GitHub.

Amazon SQS Examples Using the AWS SDK for Go

Amazon Simple Queue Service (Amazon SQS) is a fully managed message queuing service that makes it easy to decouple and scale microservices, distributed systems, and serverless applications. The AWS SDK for Go examples can integrate Amazon SQS into your applications. The examples assume you have already set up and configured the SDK (that is, you've imported all required packages and set your credentials and region). For more information, see Getting Started with the AWS SDK for Go and Configuring the AWS SDK for Go.

You can download complete versions of these example files from the aws-doc-sdk-examples repository on GitHub.

Topics

- Using Amazon SQS Queues
- Sending and Receiving Messages in Amazon SQS
- Managing Visibility Timeout in Amazon SQS Queues
- Enabling Long Polling in Amazon SQS Queues
- Using Dead Letter Queues in Amazon SQS

Using Amazon SQS Queues

These AWS SDK for Go examples show you how to:

- List Amazon SQS queues
- Create Amazon SQS queues
- Get Amazon SQS queue URLs
- Delete Amazon SQS queues

You can download complete versions of these example files from the aws-doc-sdk-examples repository on GitHub.

Scenario

These examples demonstrate how to work with Amazon SQS queues.

The code uses these methods of the Amazon SQS client class:

- CreateQueue
- ListQueues
- GetQueueUrl
- DeleteQueue

Prerequisites

- You have set up and configured the AWS SDK for Go.
- You are familiar with using Amazon SQS. To learn more, see How Queues Work in the Amazon SQS Developer Guide.

List Queues

Create a new Go file named `ListQueues.go`. You must import the relevant Go and AWS SDK for Go packages by adding the following lines.

```
import (
    "fmt"

    "github.com/aws/aws-sdk-go/aws/session"
    "github.com/aws/aws-sdk-go/service/sqs"
)
```

Initialize a session that the SDK uses to load credentials from the shared credentials file, `~/.aws/credentials` and the default region from `~/.aws/config`.

```
sess := session.Must(session.NewSessionWithOptions(session.Options{
    SharedConfigState: session.SharedConfigEnable,
}))
```

Create a service client and call `ListQueues`, passing in `nil` to return all queues.

```
svc := sqs.New(sess)

result, err := svc.ListQueues(nil)
```

Loop through the queue URLs to print them.

```
for i, url := range result.QueueUrls {
    fmt.Printf("%d: %s\n", i, *url)
}
```

See the complete example on GitHub.

Create a Queue

Create a new Go file named `CreateQueue.go`. You must import the relevant Go and AWS SDK for Go packages by adding the following lines.

```
import (
    "flag"
```

```
    "fmt"

    "github.com/aws/aws-sdk-go/aws"
    "github.com/aws/aws-sdk-go/aws/session"
    "github.com/aws/aws-sdk-go/service/sqs"
)
```

Get the queue name from the command line.

```
queue := flag.String("q", "", "The name of the queue")
flag.Parse()

if *queue == "" {
    fmt.Println("You must supply a queue name (-q QUEUE")
    return
}
```

Initialize a session that the SDK uses to load credentials from the shared credentials file, `~/.aws/credentials` and the default region from `~/.aws/config`.

```
sess := session.Must(session.NewSessionWithOptions(session.Options{
    SharedConfigState: session.SharedConfigEnable,
}))
```

Create a service client and call `CreateQueue`, passing in the queue name and queue attributes.

```
svc := sqs.New(sess)

result, err := svc.CreateQueue(&sqs.CreateQueueInput{
    QueueName: queue,
    Attributes: map[string]*string{
        "DelaySeconds":           aws.String("60"),
        "MessageRetentionPeriod": aws.String("86400"),
    },
})
```

Display the queue URL.

```
fmt.Println("URL: " + *result.QueueUrl)
```

See the complete example on GitHub.

Get the URL of a Queue

Create a new Go file named `GetQueueUrl.go`. You must import the relevant Go and AWS SDK for Go packages by adding the following lines.

```
import (
    "flag"
    "fmt"

    "github.com/aws/aws-sdk-go/aws/session"
    "github.com/aws/aws-sdk-go/service/sqs"
)
```

Get the name of the queue from the command line.

```
queue := flag.String("q", "", "The name of the queue")
flag.Parse()

if *queue == "" {
    fmt.Println("You must supply a queue name (-q QUEUE")
    return
}
```

Initialize a session that the SDK uses to load credentials from the shared credentials file, `~/.aws/credentials` and the default region from `~/.aws/config`.

```
sess := session.Must(session.NewSessionWithOptions(session.Options{
    SharedConfigState: session.SharedConfigEnable,
}))
```

Create a service client and call `GetQueueUrl`, passing in the queue name.

```
svc := sqs.New(sess)

result, err := svc.GetQueueUrl(&sqs.GetQueueUrlInput{
    QueueName: queue,
})
```

Display the URL of the queue.

```
fmt.Println("URL: " + *result.QueueUrl)
```

See the complete example on GitHub.

Delete a Queue

Create a new Go file named `DeleteQueue.go`. You must import the relevant Go and AWS SDK for Go packages by adding the following lines.

```
import (
    "flag"
    "fmt"

    "github.com/aws/aws-sdk-go/aws/session"
    "github.com/aws/aws-sdk-go/service/sqs"
)
```

Get the name of the queue from the command line.

```
queue := flag.String("q", "", "The name of the queue")
flag.Parse()

if *queue == "" {
    fmt.Println("You must supply a queue name (-q QUEUE")
    return
}
```

Initialize a session that the SDK uses to load credentials from the shared credentials file, `~/.aws/credentials` and the default region from `~/.aws/config`.

```
sess := session.Must(session.NewSessionWithOptions(session.Options{
    SharedConfigState: session.SharedConfigEnable,
}))
```

Create a service client and all `DeleteQueue` passing in the queue name.

```
svc := sqs.New(sess)

result, err := svc.GetQueueUrl(&sqs.GetQueueUrlInput{
    QueueName: queueName,
})
```

See the complete example on GitHub.

Sending and Receiving Messages in Amazon SQS

These AWS SDK for Go examples show you how to:

- Send a message to an Amazon SQS queue
- Receive a message from an Amazon SQS queue
- Delete a message from an Amazon SQS queue

You can download complete versions of these example files from the aws-doc-sdk-examples repository on GitHub.

The Scenario

These examples demonstrate sending, receiving, and deleting messages from an Amazon SQS queue.

The code uses these methods of the Amazon SQS client class:

- SendMessage
- ReceiveMessage
- DeleteMessage
- GetQueueUrl

Prerequisites

- You have set up and configured the AWS SDK for Go.
- You are familiar with the details of Amazon SQS messages. To learn more, see Sending a Message to an Amazon SQS Queue and Receiving and Deleting a Message from an Amazon SQS Queue in the Amazon SQS Developer Guide.

Send a Message to a Queue

Create a new Go file named `SendMessage.go`.

You must import the relevant Go and AWS SDK for Go packages by adding the following lines.

```
import (
```

```
    "flag"
    "fmt"

    "github.com/aws/aws-sdk-go/aws"
    "github.com/aws/aws-sdk-go/aws/session"
    "github.com/aws/aws-sdk-go/service/sqs"
)
```

Get the name of the queue from the command line.

```
queue := flag.String("q", "", "The name of the queue")
flag.Parse()

if *queue == "" {
    fmt.Println("You must supply the name of a queue (-q QUEUE)")
    return
}
```

Initialize a session that the SDK uses to load credentials from the shared credentials file, `~/.aws/credentials` and the default region from `~/.aws/config`.

```
sess := session.Must(session.NewSessionWithOptions(session.Options{
    SharedConfigState: session.SharedConfigEnable,
}))
```

Create a service client and call *SendMessage*. The input represents information about a fiction best seller for a particular week and defines title, author, and weeks on the list values.

```
svc := sqs.New(sess)

_, err := svc.SendMessage(&sqs.SendMessageInput{
    DelaySeconds: aws.Int64(10),
    MessageAttributes: map[string]*sqs.MessageAttributeValue{
        "Title": &sqs.MessageAttributeValue{
            DataType:    aws.String("String"),
            StringValue: aws.String("The Whistler"),
        },
        "Author": &sqs.MessageAttributeValue{
            DataType:    aws.String("String"),
            StringValue: aws.String("John Grisham"),
        },
        "WeeksOn": &sqs.MessageAttributeValue{
```

```
            DataType:    aws.String("Number"),
            StringValue: aws.String("6"),
        },
    },
    MessageBody: aws.String("Information about current NY Times fiction bestseller for
 week of 12/11/2016."),
    QueueUrl:    queueURL,
})
```

See the complete example on GitHub.

Receiving a Message from a Queue

Create a new Go file named `ReceiveMessage.go`.

You must import the relevant Go and AWS SDK for Go packages by adding the following lines.

```
import (
    "flag"
    "fmt"

    "github.com/aws/aws-sdk-go/aws"
    "github.com/aws/aws-sdk-go/aws/session"
    "github.com/aws/aws-sdk-go/service/sqs"
)
```

Get the name of the queue and how long the message is hidden from other consumers from the command line.

```
queue := flag.String("q", "", "The name of the queue")
timeout := flag.Int64("t", 5, "How long, in seconds, that the message is hidden from
 others")
flag.Parse()

if *queue == "" {
    fmt.Println("You must supply the name of a queue (-q QUEUE)")
    return
}

if *timeout < 0 {
    *timeout = 0
}
```

```
if *timeout > 12*60*60 {
    *timeout = 12 * 60 * 60
}
```

Initialize a session that the SDK uses to load credentials from the shared credentials file, `~/.aws/credentials` and the default region from `~/.aws/config`.

```
sess := session.Must(session.NewSessionWithOptions(session.Options{
    SharedConfigState: session.SharedConfigEnable,
}))
```

Create a service client and call the *GetQueueUrl* function to get the URL of the queue.

```
svc := sqs.New(sess)

urlResult, err := svc.GetQueueUrl(&sqs.GetQueueUrlInput{
    QueueName: queue,
})
```

The URL of the queue is in the *QueueUrl* property of the returned object.

```
queueURL := urlResult.QueueUrl
```

Call the *ReceiveMessage* function to get the messages in the queue.

```
msgResult, err := svc.ReceiveMessage(&sqs.ReceiveMessageInput{
    AttributeNames: []*string{
        aws.String(sqs.MessageSystemAttributeNameSentTimestamp),
    },
    MessageAttributeNames: []*string{
        aws.String(sqs.QueueAttributeNameAll),
    },
    QueueUrl:            queueURL,
    MaxNumberOfMessages: aws.Int64(1),
    VisibilityTimeout:   timeout,
})
```

Print the message handle of the first message in the queue (you need the handle to delete the message).

```
fmt.Println("Message Handle: " + *msgResult.Messages[0].ReceiptHandle)
```

See the complete example on GitHub.

Delete a Message from a Queue

Create a new Go file named `DeleteMessage.go`.

You must import the relevant Go and AWS SDK for Go packages by adding the following lines.

```
import (
    "flag"
    "fmt"

    "github.com/aws/aws-sdk-go/aws/session"
    "github.com/aws/aws-sdk-go/service/sqs"
)
```

Get the name of the queue and the handle for the message from the command line.

```
queue := flag.String("q", "", "The name of the queue")
messageHandle := flag.String("m", "", "The receipt handle of the message")
flag.Parse()

if *queue == "" || *messageHandle == "" {
    fmt.Println("You must supply a queue name (-q QUEUE) and message receipt handle (-m
 MESSAGE-HANDLE)")
    return
}
```

Initialize a session that the SDK uses to load credentials from the shared credentials file, `~/.aws/credentials` and the default region from `~/.aws/config`.

```
sess := session.Must(session.NewSessionWithOptions(session.Options{
    SharedConfigState: session.SharedConfigEnable,
}))
```

Create a service client and call the *DeleteMessage* function, passing in the name of the queue and the message handle.

```
svc := sqs.New(sess)
```

```
_, err := svc.DeleteMessage(&sqs.DeleteMessageInput{
    QueueUrl:      queueURL,
    ReceiptHandle: messageHandle,
})
```

See the complete example on GitHub.

Managing Visibility Timeout in Amazon SQS Queues

This AWS SDK for Go example shows you how to:

- Change visibility timeout with Amazon SQS queues

You can download complete versions of these example files from the aws-doc-sdk-examples repository on GitHub.

Scenario

This example manages visibility timeout with Amazon SQS queues. Visibility is the duration, in seconds, while messages are in the queue, but not available to other consumers. It uses these methods of the Amazon SQS client class:

- ChangeMessageVisibility
- GetQueueUrl

Prerequisites

- You have set up and configured the AWS SDK for Go.
- You are familiar with using Amazon SQS visibility timeout. To learn more, see Visibility Timeout in the Amazon SQS Developer Guide.

Change the Visibility Timeout

Create a new Go file named `ChangeMsgVisibility.go`.

You must import the relevant Go and AWS SDK for Go packages by adding the following lines.

```
import (
```

```
    "flag"
    "fmt"

    "github.com/aws/aws-sdk-go/aws/session"
    "github.com/aws/aws-sdk-go/service/sqs"
)
```

Get the queue name, receipt handle of the message, and visibility duration from the command line. Ensure the visibility is from 0 (zero) seconds to 12 hours.

```
queue := flag.String("q", "", "The name of the queue")
handle := flag.String("h", "", "The receipt handle of the message")
visibility := flag.Int64("v", 30, "The duration, in seconds, that the message is not
 visible to other consumers")
flag.Parse()

if *queue == "" || *handle == "" {
    fmt.Println("You must supply a queue name (-q QUEUE) and message receipt handle (-h
 HANDLE)")
    return
}

if *visibility < 0 {
    *visibility = 0
}

if *visibility > 12*60*60 {
    *visibility = 12 * 60 * 60
}
```

Initialize a session that the SDK will use to load credentials from the shared credentials file, `~/.aws/credentials` and the default region from `~/.aws/config`.

```
sess := session.Must(session.NewSessionWithOptions(session.Options{
    SharedConfigState: session.SharedConfigEnable,
}))
```

Create a service client and call `GetQueueUrl` to retrieve the URL of the queue.

```
svc := sqs.New(sess)

result, err := svc.GetQueueUrl(&sqs.GetQueueUrlInput{
```

```
    QueueName: queue,
})
```

The URL of the queue is in the `QueueUrl` property of the returned object.

```
queueURL := urlResult.QueueUrl
```

Call `ChangeMessageVisibility` to change the visibility of the messages in the queue.

```
_, err := svc.ChangeMessageVisibility(&sqs.ChangeMessageVisibilityInput{
    ReceiptHandle:     handle,
    QueueUrl:          queueURL,
    VisibilityTimeout: visibility,
})
```

See the complete example on GitHub.

Enabling Long Polling in Amazon SQS Queues

These AWS SDK for Go examples show you how to:

- Enable long polling when you create an Amazon SQS queue
- Enable long polling on an existing Amazon SQS queue
- Enable long polling when a message is received

You can download complete versions of these example files from the aws-doc-sdk-examples repository on GitHub.

Scenario

Long polling reduces the number of empty responses by allowing Amazon SQS to wait a specified time for a message to become available in the queue before sending a response. Also, long polling eliminates false empty responses by querying all of the servers instead of a sampling of servers. To enable long polling, you must specify a non-zero wait time for received messages. You can do this by setting the `ReceiveMessageWaitTimeSeconds` parameter of a queue or by setting the `WaitTimeSeconds` parameter on a message when it is received.

The code uses these methods of the Amazon SQS client class:

- CreateQueue
- SetQueueAttributes
- ReceiveMessage

Prerequisites

- You have set up and configured the AWS SDK for Go.
- You are familiar with Amazon SQS polling. To learn more, see Long Polling in the Amazon SQS Developer Guide.

Enable Long Polling When Creating a Queue

This example creates a queue with long polling enabled. If the queue already exists, no error is returned.

Create a new Go file named `CreateLPQueue.go`. You must import the relevant Go and AWS SDK for Go packages by adding the following lines.

```
import (
    "flag"
    "fmt"
    "strconv"

    "github.com/aws/aws-sdk-go/aws"
    "github.com/aws/aws-sdk-go/aws/session"
    "github.com/aws/aws-sdk-go/service/sqs"
)
```

Get the queue name and wait time from the command line. Ensure that the wait time is between 0 (zero) and 20 seconds.

```
queue := flag.String("q", "", "The name of the queue")
waitTime := flag.Int("w", 10, "How long, in seconds, to wait for long polling")
flag.Parse()

if *queue == "" {
    fmt.Println("You must supply a queue name (-q QUEUE")
    return
}
```

```
if *waitTime < 1 {
    *waitTime = 1
}

if *waitTime > 20 {
    *waitTime = 20
}
```

Initialize a session that the SDK will use to load credentials from the shared credentials file, `~/.aws/credentials` and the default AWS Region from `~/.aws/config`.

```
sess := session.Must(session.NewSessionWithOptions(session.Options{
    SharedConfigState: session.SharedConfigEnable,
}))
```

Create a service client and call `CreateQueue`, passing in the time to wait for messages.

```
svc := sqs.New(sess)

result, err := svc.CreateQueue(&sqs.CreateQueueInput{
    QueueName: queueName,
    Attributes: aws.StringMap(map[string]string{
        "ReceiveMessageWaitTimeSeconds": strconv.Itoa(*waitTime),
    }),
})
```

See the complete example on GitHub.

Enable Long Polling on an Existing Queue

Create a new Go file named `ConfigureLPQueue.go`.

You must import the relevant Go and AWS SDK for Go packages by adding the following lines.

```
import (
    "flag"
    "fmt"
    "strconv"

    "github.com/aws/aws-sdk-go/aws"
    "github.com/aws/aws-sdk-go/aws/session"
```

```
    "github.com/aws/aws-sdk-go/service/sqs"
)
```

Get the queue name and the optional timeout value from the command line. Ensure that the wait time is between 1 and 20 seconds.

```
queue := flag.String("q", "", "The name of the queue")
waitTime := flag.Int("w", 10, "The wait time, in seconds, for long polling")
flag.Parse()

if *queue == "" {
    fmt.Println("You must supply a queue name (-q QUEUE")
    return
}

if *waitTime < 1 {
    *waitTime = 1
}

if *waitTime > 20 {
    *waitTime = 20
}
```

Initialize a session that the SDK will use to load credentials from the shared credentials file, `~/.aws/credentials`, and a default AWS Region from `~/.aws/config`.

```
sess := session.Must(session.NewSessionWithOptions(session.Options{
    SharedConfigState: session.SharedConfigEnable,
}))
```

Get the URL of the queue.

```
svc := sqs.New(sess)

result, err := svc.GetQueueUrl(&sqs.GetQueueUrlInput{
    QueueName: queue,
})
```

The URL is in the QueueUrl property of the returned object.

```
queueURL := result.QueueUrl
```

Update the queue to enable long polling with a call to `SetQueueAttributes`, passing in the queue URL.

```
_, err := svc.SetQueueAttributes(&sqs.SetQueueAttributesInput{
    QueueUrl: queueURL,
    Attributes: aws.StringMap(map[string]string{
        "ReceiveMessageWaitTimeSeconds": strconv.Itoa(aws.IntValue(waitTime)),
    }),
})
```

See the complete example on GitHub.

Enable Long Polling on Message Receipt

Create a new Go file named `ReceiveLPMessage.go`.

You must import the relevant Go and AWS SDK for Go packages by adding the following lines.

```
import (
    "flag"
    "fmt"

    "github.com/aws/aws-sdk-go/aws"
    "github.com/aws/aws-sdk-go/aws/session"
    "github.com/aws/aws-sdk-go/service/sqs"
)
```

Get the queue name and optional visibility and wait time values from the command line. Ensure that the visibility is between 0 (zero) seconds and 12 hours and that the wait time is between 0 and 20 seconds.

```
queue := flag.String("q", "", "The name of the queue")
waitTime := flag.Int64("w", 10, "How long the queue waits for messages")
flag.Parse()

if *queue == "" {
    fmt.Println("You must supply a queue name (-q QUEUE")
    return
}
```

```
if *waitTime < 0 {
    *waitTime = 0
}

if *waitTime > 20 {
    *waitTime = 20
}
```

Initialize a session that the SDK will use to load credentials from the shared credentials file, `~/.aws/credentials` and the default AWS Region from `~/.aws/config`.

```
sess := session.Must(session.NewSessionWithOptions(session.Options{
    SharedConfigState: session.SharedConfigEnable,
}))
```

Create a service client and call `GetQueueUrl` to get the URL of the queue.

```
svc := sqs.New(sess)

result, err := svc.GetQueueUrl(&sqs.GetQueueUrlInput{
    QueueName: queue,
})
```

Call `ReceiveMessage` to get the messages, using long polling, from the queue.

```
result, err := svc.ReceiveMessage(&sqs.ReceiveMessageInput{
    QueueUrl: queueURL,
    AttributeNames: aws.StringSlice([]string{
        "SentTimestamp",
    }),
    MaxNumberOfMessages: aws.Int64(1),
    MessageAttributeNames: aws.StringSlice([]string{
        "All",
    }),
    WaitTimeSeconds: waitTime,
})
```

Display the IDs of the mesages.

```
    fmt.Println("Message IDs:")
```

```
    for _, msg := range msgs {
        fmt.Println("    " + *msg.MessageId)
    }
```

See the complete example on GitHub.

Using Dead Letter Queues in Amazon SQS

This AWS SDK for Go example shows you how to configure source Amazon SQS queues that send messages to a dead letter queue.

You can download complete versions of these example files from the aws-doc-sdk-examples repository on GitHub.

Scenario

A dead letter queue is one that other (source) queues can target for messages that can't be processed successfully. You can set aside and isolate these messages in the dead letter queue to determine why their processing didn't succeed. You must individually configure each source queue that sends messages to a dead letter queue. Multiple queues can target a single dead letter queue.

The code uses this method of the Amazon SQS client class:

- SetQueueAttributes

Prerequisites

- You have set up and configured the AWS SDK for Go.
- You are familiar with Amazon SQS dead letter queues. To learn more, see Using Amazon SQS Dead Letter Queues in the Amazon SQS Developer Guide.

Configure Source Queues

After you create a queue to act as a dead letter queue, you must configure the other queues that route unprocessed messages to the dead letter queue. To do this, specify a redrive policy that identifies the queue to use as a dead letter queue and the maximum number of receives by individual messages before they are routed to the dead letter queue.

Create a new Go file with the name `DeadLetterQueue.go`.

You must import the relevant Go and AWS SDK for Go packages by adding the following lines.

```
import (
    "encoding/json"
    "flag"
    "fmt"
    "strings"

    "github.com/aws/aws-sdk-go/aws"
    "github.com/aws/aws-sdk-go/aws/session"
    "github.com/aws/aws-sdk-go/service/sqs"
)
```

Get the name of the queue and the dead letter queue from the commmand line.

```
    queue := flag.String("q", "", "The name of the queue")
    dlQueue := flag.String("d", "", "The name of the dead-letter queue")
    flag.Parse()

    if *queue == "" || *dlQueue == "" {
        fmt.Println("You must supply the names of the queue (-q QUEUE) and the dead-
letter queue (-d DLQUEUE)")
        return
    }
```

Initialize a session that the SDK will use to load credentials from the shared credentials file, `~/.aws/credentials` and the default AWS Region from `~/.aws/config`.

```
sess := session.Must(session.NewSessionWithOptions(session.Options{
    SharedConfigState: session.SharedConfigEnable,
}))
```

Create a service client and call `GetQueueUrl` to get the URL for the queue.

```
svc := sqs.New(sess)

result, err := svc.GetQueueUrl(&sqs.GetQueueUrlInput{
    QueueName: queue,
})
```

The URL of the queue is in the `QueueUrl` property of the returned object.

```
queueURL := result.QueueUrl
```

Similarly, get the URL of the dead letter queue.

Create the ARN of the dead-letter queue from the URL.

```
parts := strings.Split(*queueURL, "/")
subParts := strings.Split(parts[2], ".")

arn := "arn:aws:" + subParts[0] + ":" + subParts[1] + ":" + parts[3] + ":" + parts[4]
```

Define the redrive policy for the queue.

```
policy := map[string]string{
    "deadLetterTargetArn": *dlQueueARN,
    "maxReceiveCount":     "10",
}
```

Marshal the policy to use as input for the `SetQueueAttributes` call.

```
b, err := json.Marshal(policy)
```

Set the policy on the queue.

```
_, err = svc.SetQueueAttributes(&sqs.SetQueueAttributesInput{
    QueueUrl: queueURL,
    Attributes: map[string]*string{
        sqs.QueueAttributeNameRedrivePolicy: aws.String(string(b)),
    },
})
```

See the complete example on GitHub.

Amazon WorkDocs Examples

You can use the following examples to access Amazon WorkDocs (WorkDocs) using the AWS SDK for Go. For more information about WorkDocs, see the Amazon WorkDocs documentation.

You need your organization ID to use these examples. Get your organization ID from the AWS console using the following steps:

- Select the AWS Directory Service
- Select `Directories`

The organization ID is the `Directory ID` corresponding to your Amazon WorkDocs site.

Examples

Topics

- Listing Users
- Listing User Docs

Listing Users

The following example lists the names of all users, or lists additional details about a user if a user name is specified on the command line. Choose Copy to save the code locally, or see the link to the complete example at the end of this topic.

Import the following Go packages.

```
import (
    "os"

    "github.com/aws/aws-sdk-go/aws/session"
    "github.com/aws/aws-sdk-go/service/workdocs"

    "flag"
    "fmt"
)
```

Get an optional user name and organization ID from the command line. If the organization ID is missing, print an error message and quit.

```
orgPtr := flag.String("o", "", "The ID of your organization")
userPtr := flag.String("u", "", "User for whom info is retrieved")

flag.Parse()

if *orgPtr == "" {
    fmt.Println("You must supply the organization ID")
```

```
    flag.PrintDefaults()
    os.Exit(1)
}
```

Create input for the `DescribeUsers` method.

```
input := new(workdocs.DescribeUsersInput)
input.OrganizationId = orgPtr

// Show all users if we don't get a user name
if *userPtr == "" {
    fmt.Println("Getting info about all users")
} else {
    fmt.Println("Getting info about user " + *userPtr)
    input.Query = userPtr
}
```

Create a session and WorkDocs client.

```
sess := session.Must(session.NewSessionWithOptions(session.Options{
    SharedConfigState: session.SharedConfigEnable,
}))

svc := workdocs.New(sess)
```

Call `DescribeUsers` and display the information for the user or all users.

```
result, err := svc.DescribeUsers(input)
if err != nil {
    fmt.Println("Error getting user info", err)
    return
}

if *userPtr == "" {
    fmt.Println("Found", *result.TotalNumberOfUsers, "users")
    fmt.Println("")
}

for _, user := range result.Users {
    fmt.Println("Username:   " + *user.Username)

    if *userPtr != "" {
```

```
        fmt.Println("Firstname:  " + *user.GivenName)
        fmt.Println("Lastname:   " + *user.Surname)
        fmt.Println("Email:      " + *user.EmailAddress)
        fmt.Println("Root folder " + *user.RootFolderId)
    }

    fmt.Println("")
}
```

See the complete example on GitHub.

Listing User Docs

The following example lists the documents for the user whose name is specified on the command line. Choose Copy to save the code locally, or see the link to the complete example at the end of this topic.

Import the following Go packages.

```
import (
    "os"

    "github.com/aws/aws-sdk-go/aws/session"
    "github.com/aws/aws-sdk-go/service/workdocs"

    "flag"
    "fmt"
)
```

Get the user and organization ID from the command line. If either is missing, print an error message and quit.

```
userPtr := flag.String("u", "", "User for whom info is retrieved")
orgPtr := flag.String("o", "", "Your organization ID")

flag.Parse()

if *userPtr == "" || *orgPtr == "" {
    flag.PrintDefaults()
    os.Exit(1)
}
```

Create a session and WorkDocs client.

```
sess := session.Must(session.NewSessionWithOptions(session.Options{
    SharedConfigState: session.SharedConfigEnable,
}))

svc := workdocs.New(sess)
```

Get the root folder for the user.

```
input := new(workdocs.DescribeUsersInput)
input.OrganizationId = orgPtr
input.Query = userPtr

result, err := svc.DescribeUsers(input)
if err != nil {
    fmt.Println("Error getting user info", err)
    return
}

var folderID = ""

if *result.TotalNumberOfUsers == 1 {
    for _, user := range result.Users {
        folderID = *user.RootFolderId
    }
```

Run the `DescribeFolderContents` method and display the name, size, and last modified information for each document.

```
result, err :=
 svc.DescribeFolderContents(&workdocs.DescribeFolderContentsInput{FolderId: &folderID})

if err != nil {
    fmt.Println("Error getting docs for user", err)
    return
}

fmt.Println(*userPtr + " docs:")
fmt.Println("")

for _, doc := range result.Documents {
```

```
    fmt.Println(*doc.LatestVersionMetadata.Name)
    fmt.Println("  Size:         ", *doc.LatestVersionMetadata.Size, "(bytes)")
    fmt.Println("  Last modified:", *doc.LatestVersionMetadata.ModifiedTimestamp)
    fmt.Println("")
}
```

See the complete example on GitHub.

Using the AWS SDK for Go Utilities

The AWS SDK for Go includes the following utilities to help you more easily use AWS services. Find the SDK utilities in their related AWS service package.

Amazon CloudFront URL Signer

The Amazon CloudFront URL signer simplifies the process of creating signed URLs. A signed URL includes information, such as an expiration date and time, that enables you to control access to your content. Signed URLs are useful when you want to distribute content through the internet, but want to restrict access to certain users (for example, to users who have paid a fee).

To sign a URL, create a `URLSigner` instance with your CloudFront key pair ID and the associated private key. Then call the `Sign` or `SignWithPolicy` method and include the URL to sign. For more information about Amazon CloudFront key pairs, see Creating CloudFront Key Pairs for Your Trusted Signers in the Amazon CloudFront Developer Guide.

The following example creates a signed URL that's valid for one hour after it is created.

```
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
}
```

For more information about the signing utility, see the sign package in the AWS SDK for Go API Reference.

Amazon DynamoDB Attributes Converter

The attributes converter simplifies converting Amazon DynamoDB attribute values to and from concrete Go types. Conversions make it easy to work with attribute values in Go and to write values to Amazon DynamoDB tables. For example, you can create records in Go and then use the converter when you want to write those records as attribute values to a DynamoDB table.

The following example converts a structure to an Amazon DynamoDB`AttributeValues` map and then puts the data to the `exampleTable`.

```
type Record struct {
    MyField string
    Letters []string
    A2Num   map[string]int
}
r := Record{
    MyField: "dynamodbattribute.ConvertToX example",
    Letters: []string{"a", "b", "c", "d"},
    A2Num:   map[string]int{"a": 1, "b": 2, "c": 3},
}

//...

svc := dynamodb.New(session.New(&aws.Config{Region: aws.String("us-west-2")}))
item, err := dynamodbattribute.ConvertToMap(r)
if err != nil {
    fmt.Println("Failed to convert", err)
    return
}
result, err := svc.PutItem(&dynamodb.PutItemInput{
    Item:      item,
    TableName: aws.String("exampleTable"),
})
fmt.Println("Item put to dynamodb", result, err)
```

For more information about the converter utility, see the dynamodbattribute package in the AWS SDK for Go API Reference.

Amazon Elastic Compute Cloud Metadata

`EC2Metadata` is a client that interacts with the Amazon EC2 metadata service. The client can help you easily retrieve information about instances on which your applications run, such as its region or local IP address. Typically, you must create and submit HTTP requests to retrieve instance metadata. Instead, create an `EC2Metadata` service client.

```
c := ec2metadata.New(session.New())
```

Then use the service client to retrieve information from a metadata category like `local-ipv4` (the private IP address of the instance).

```
localip, err := c.GetMetadata("local-ipv4")
```

```
if err != nil {
    log.Printf("Unable to retrieve the private IP address from the EC2 instance: %s\n",
 err)
    return
}
```

For a list of all metadata categories, see Instance Metadata Categories in the Amazon EC2 User Guide.

Retrieving an Instance's Region

There's no instance metadata category that returns only the region of an instance. Instead, use the included `Region` method to easily return an instance's region.

```
region, err := ec2metadata.New(session.New()).Region()
if err != nil {
    log.Printf("Unable to retrieve the region from the EC2 instance %v\n", err)
}
```

For more information about the EC2 metadata utility, see the ec2metadata package in the AWS SDK for Go API Reference.

Amazon S3 Transfer Managers

The Amazon Simple Storage Service upload and download managers can break up large objects so they can be transferred in multiple parts, in parallel. This makes it easy to resume interrupted transfers.

Upload Manager

The Amazon Simple Storage Service upload manager determines if a file can be split into smaller parts and uploaded in parallel. You can customize the number of parallel uploads and the size of the uploaded parts.

Example: Uploading a File

The following example uses the Amazon S3 `Uploader` to upload a file. Using `Uploader` is similar to the `s3.PutObject()` operation.

```
mySession, _ := session.NewSession()
```

```
uploader := s3manager.NewUploader(mySession)
result, err := uploader.Upload(&s3manager.UploadInput{
    Bucket: &uploadBucket,
    Key:    &uploadFileKey,
    Body:   uploadFile,
})
```

Configuration Options

When you instantiate an `Uploader` instance, you can specify several configuration options (`UploadOptions`) to customize how objects are uploaded:

- `PartSize`– Specifies the buffer size, in bytes, of each part to upload. The minimum size per part is 5 MB.
- `Concurrency`– Specifies the number of parts to upload in parallel.
- `LeavePartsOnError`– Indicates whether to leave successfully uploaded parts in Amazon S3.

Tweak the `PartSize` and `Concurrency` configuration values to find the optimal configuration. For example, systems with high-bandwidth connections can send bigger parts and more uploads in parallel.

For more information about `Uploader` and its configurations, see the s3manager package in the AWS SDK for Go API Reference.

UploadInput Body Field (io.ReadSeeker vs. io.Reader)

The `Body` field of the `s3manager.UploadInput` struct is an `io.Reader` type. However, the field also satisfies the `io.ReadSeeker` interface.

For `io.ReadSeeker` types, the `Uploader` doesn't buffer the body contents before sending it to Amazon S3. `Uploader` calculates the expected number of parts before uploading the file to Amazon S3. If the current value of `PartSize` requires more than 10,000 parts to upload the file, `Uploader` increases the part size value so that fewer parts are required.

For `io.Reader` types, the bytes of the reader must buffer each part in memory before the part is uploaded. When you increase the `PartSize` or `Concurrency` value, the required memory (RAM) for the `Uploader` increases significantly. The required memory is approximately ``*PartSize*`` * ``*Concurrency*``. For example, if you specify 100 MB for `PartSize` and 10 for `Concurrency`, the required memory will be at least 1 GB.

Because an `io.Reader` type cannot determine its size before reading its bytes, `Uploader` cannot calculate how many parts must be uploaded. Consequently, `Uploader` can reach the Amazon S3 upload limit of 10,000 parts for large files if you set the `PartSize` too low. If you try to upload more than 10,000 parts, the upload stops and returns an error.

Handling Partial Uploads

If an upload to Amazon S3 fails, by default, `Uploader` uses the Amazon S3`AbortMultipartUpload` operation to remove the uploaded parts. This functionality ensures that failed uploads do not consume Amazon S3 storage.

You can set `LeavePartsOnError` to true so that the `Uploader` doesn't delete successfully uploaded parts. This is useful for resuming partially completed uploads. To operate on uploaded parts, you must get the `UploadID` of the failed upload. The following example demonstrates how to use the `s3manager.MultiUploadFailure` message to get the `UploadID`.

```
u := s3manager.NewUploader(session.New())
output, err := u.upload(input)
if err != nil {
    if multierr, ok := err.(s3manager.MultiUploadFailure); ok {
        // Process error and its associated uploadID
        fmt.Println("Error:", multierr.Code(), multierr.Message(), multierr.UploadID())
    } else {
        // Process error generically
        fmt.Println("Error:", err.Error())
    }
}
```

Example: Upload a Folder to Amazon S3

The following example uses the `path/filepath` package to recursively gather a list of files and upload them to the specified Amazon S3 bucket. The keys of the Amazon S3 objects are prefixed with the file's relative path.

```
package main

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

```
    "github.com/aws/aws-sdk-go/aws"
    "github.com/aws/aws-sdk-go/aws/session"
    "github.com/aws/aws-sdk-go/service/s3/s3manager"
)

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)
    }()

    // For each file found walking, upload it to S3
    uploader := s3manager.NewUploader(session.New())
    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(&s3manager.UploadInput{
```

```
            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
}
```

Example: Upload a File to Amazon S3 and Send its Location to Amazon SQS

The following example uploads a file to an Amazon S3 bucket and then sends a notification message of the file's location to an Amazon Simple Queue Service queue.

```
package main

import (
    "log"
    "os"

    "github.com/aws/aws-sdk-go/aws"
    "github.com/aws/aws-sdk-go/aws/session"
    "github.com/aws/aws-sdk-go/service/s3/s3manager"
    "github.com/aws/aws-sdk-go/service/sqs"
)

// Uploads a file to a specific bucket in S3 with the file name
// as the object's key. After it's uploaded, a message is sent
// to a queue.
```

```
func main() {
    if len(os.Args) != 4 {
        log.Fatalln("Usage:", os.Args[0], "<bucket> <queue> <file>")
    }

    file, err := os.Open(os.Args[3])
    if err != nil {
        log.Fatal("Open failed:", err)
    }
    defer file.Close()

    // Upload the file to S3 using the S3 Manager
    uploader := s3manager.NewUploader(session.New())
    uploadRes, err := uploader.Upload(&s3manager.UploadInput{
        Bucket: aws.String(os.Args[1]),
        Key:    aws.String(file.Name()),
        Body:   file,
    })
    if err != nil {
        log.Fatalln("Upload failed:", err)
    }

    // Get the URL of the queue that the message will be posted to
    svc := sqs.New(session.New())
    urlRes, err := svc.GetQueueUrl(&sqs.GetQueueUrlInput{
        QueueName: aws.String(os.Args[2]),
    })
    if err != nil {
        log.Fatalln("GetQueueURL failed:", err)
    }

    // Send the message to the queue
    _, err = svc.SendMessage(&sqs.SendMessageInput{
        MessageBody: &uploadRes.Location,
        QueueUrl:    urlRes.QueueUrl,
    })
    if err != nil {
        log.Fatalln("SendMessage failed:", err)
    }
}
```

Download Manager

The Amazon S3 download manager determines if a file can be split into smaller parts and downloaded in parallel. You can customize the number of parallel downloads and the size of the downloaded parts.

Example: Download a File

The following example uses the Amazon S3`Downloader` to download a file. Using `Downloader` is similar to the `s3.GetObject()` operation.

```
downloader := s3manager.NewDownloader(session.New())
numBytes, err := downloader.Download(downloadFile,
  &s3.GetObjectInput{
    Bucket: &downloadBucket,
    Key:    &downloadFileKey,
})
```

The `downloadFile` parameter is an `io.WriterAt` type. The `WriterAt` interface enables the `Downloader` to write multiple parts of the file in parallel.

Configuration Options

When you instantiate a `Downloader` instance, you can specify several configuration options (`DownloadOptions`) to customize how objects are downloaded:

- `PartSize`– Specifies the buffer size, in bytes, of each part to download. The minimum size per part is 5 MB.
- `Concurrency`– Specifies the number of parts to download in parallel.

Tweak the `PartSize` and `Concurrency` configuration values to find the optimal configuration. For example, systems with high-bandwidth connections can receive bigger parts and more downloads in parallel.

For more information about `Downloader` and its configurations, see the s3manager package in the AWS SDK for Go API Reference.

Example: Download All Objects in a Bucket

The following example uses pagination to gather a list of objects from an Amazon S3 bucket. Then it downloads each object to a local file.

```
package main

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

    "github.com/aws/aws-sdk-go/aws/session"
    "github.com/aws/aws-sdk-go/service/s3"
    "github.com/aws/aws-sdk-go/service/s3/s3manager"
)

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

func main() {
    manager := s3manager.NewDownloader(session.New())
    d := downloader{bucket: Bucket, dir: LocalDirectory, Downloader: manager}

    client := s3.New(session.New())
    params := &s3.ListObjectsInput{Bucket: &Bucket, Prefix: &Prefix}
    client.ListObjectsPages(params, d.eachPage)
}

type downloader struct {
    *s3manager.Downloader
    bucket, dir string
}

func (d *downloader) eachPage(page *s3.ListObjectsOutput, more bool) bool {
    for _, obj := range page.Contents {
        d.downloadToFile(*obj.Key)
    }

    return true
```

```
}

func (d *downloader) downloadToFile(key string) {
    // Create the directories in the path
    file := filepath.Join(d.dir, key)
    if err := os.MkdirAll(filepath.Dir(file), 0775); err != nil {
        panic(err)
    }

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

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

Security for this AWS Product or Service

Cloud security at Amazon Web Services (AWS) is the highest priority. As an AWS customer, you benefit from a data center and network architecture that is built to meet the requirements of the most security-sensitive organizations. Security is a shared responsibility between AWS and you. The Shared Responsibility Model describes this as Security of the Cloud and Security in the Cloud.

Security of the Cloud – AWS is responsible for protecting the infrastructure that runs all of the services offered in the AWS Cloud and providing you with services that you can use securely. Our security responsibility is the highest priority at AWS, and the effectiveness of our security is regularly tested and verified by third-party auditors as part of the AWS Compliance Programs.

Security in the Cloud – Your responsibility is determined by the AWS service you are using, and other factors including the sensitivity of your data, your organization's requirements, and applicable laws and regulations.

This AWS product or service follows the shared responsibility model through the specific Amazon Web Services (AWS) services it supports. For AWS service security information, see the AWS service security documentation page and AWS services that are in scope of AWS compliance efforts by compliance program.

Topics

- Data Protection in this AWS Product or Service
- Identity and Access Management
- Compliance Validation for this AWS Product or Service
- Resilience for this AWS Product or Service
- Infrastructure Security for this AWS Product or Service
- Enforcing a minimum TLS version in the AWS SDK for Go
- Amazon S3 Encryption Client Migration

Data Protection in this AWS Product or Service

The AWS shared responsibility model applies to data protection in this AWS product or service. As described in this model, AWS is responsible for protecting the global infrastructure that runs all of the AWS Cloud. You are responsible for maintaining control over your content that is hosted on

this infrastructure. You are also responsible for the security configuration and management tasks for the AWS services that you use. For more information about data privacy, see the Data Privacy FAQ. For information about data protection in Europe, see the AWS Shared Responsibility Model and GDPR blog post on the *AWS Security Blog*.

For data protection purposes, we recommend that you protect AWS account credentials and set up individual users with AWS IAM Identity Center or AWS Identity and Access Management (IAM). That way, each user is given only the permissions necessary to fulfill their job duties. We also recommend that you secure your data in the following ways:

- Use multi-factor authentication (MFA) with each account.
- Use SSL/TLS to communicate with AWS resources. We require TLS 1.2 and recommend TLS 1.3.
- Set up API and user activity logging with AWS CloudTrail. For information about using CloudTrail trails to capture AWS activities, see Working with CloudTrail trails in the *AWS CloudTrail User Guide*.
- Use AWS encryption solutions, along with all default security controls within AWS services.
- Use advanced managed security services such as Amazon Macie, which assists in discovering and securing sensitive data that is stored in Amazon S3.
- If you require FIPS 140-3 validated cryptographic modules when accessing AWS through a command line interface or an API, use a FIPS endpoint. For more information about the available FIPS endpoints, see Federal Information Processing Standard (FIPS) 140-3.

We strongly recommend that you never put confidential or sensitive information, such as your customers' email addresses, into tags or free-form text fields such as a **Name** field. This includes when you work with this AWS product or service or other AWS services using the console, API, AWS CLI, or AWS SDKs. Any data that you enter into tags or free-form text fields used for names may be used for billing or diagnostic logs. If you provide a URL to an external server, we strongly recommend that you do not include credentials information in the URL to validate your request to that server.

Identity and Access Management

AWS Identity and Access Management (IAM) is an AWS service that helps an administrator securely control access to AWS resources. IAM administrators control who can be *authenticated* (signed in) and *authorized* (have permissions) to use AWS resources. IAM is an AWS service that you can use with no additional charge.

Topics

- Audience
- Authenticating with identities
- Managing access using policies
- How AWS services work with IAM
- Troubleshooting AWS identity and access

Audience

How you use AWS Identity and Access Management (IAM) differs, depending on the work that you do in AWS.

Service user – If you use AWS services to do your job, then your administrator provides you with the credentials and permissions that you need. As you use more AWS features to do your work, you might need additional permissions. Understanding how access is managed can help you request the right permissions from your administrator. If you cannot access a feature in AWS, see Troubleshooting AWS identity and access or the user guide of the AWS service you are using.

Service administrator – If you're in charge of AWS resources at your company, you probably have full access to AWS. It's your job to determine which AWS features and resources your service users should access. You must then submit requests to your IAM administrator to change the permissions of your service users. Review the information on this page to understand the basic concepts of IAM. To learn more about how your company can use IAM with AWS, see the user guide of the AWS service you are using.

IAM administrator – If you're an IAM administrator, you might want to learn details about how you can write policies to manage access to AWS. To view example AWS identity-based policies that you can use in IAM, see the user guide of the AWS service you are using.

Authenticating with identities

Authentication is how you sign in to AWS using your identity credentials. You must be *authenticated* (signed in to AWS) as the AWS account root user, as an IAM user, or by assuming an IAM role.

You can sign in to AWS as a federated identity by using credentials provided through an identity source. AWS IAM Identity Center (IAM Identity Center) users, your company's single sign-on

authentication, and your Google or Facebook credentials are examples of federated identities. When you sign in as a federated identity, your administrator previously set up identity federation using IAM roles. When you access AWS by using federation, you are indirectly assuming a role.

Depending on the type of user you are, you can sign in to the AWS Management Console or the AWS access portal. For more information about signing in to AWS, see How to sign in to your AWS account in the *AWS Sign-In User Guide*.

If you access AWS programmatically, AWS provides a software development kit (SDK) and a command line interface (CLI) to cryptographically sign your requests by using your credentials. If you don't use AWS tools, you must sign requests yourself. For more information about using the recommended method to sign requests yourself, see AWS Signature Version 4 for API requests in the *IAM User Guide*.

Regardless of the authentication method that you use, you might be required to provide additional security information. For example, AWS recommends that you use multi-factor authentication (MFA) to increase the security of your account. To learn more, see Multi-factor authentication in the *AWS IAM Identity Center User Guide* and AWS Multi-factor authentication in IAM in the *IAM User Guide*.

AWS account root user

When you create an AWS account, you begin with one sign-in identity that has complete access to all AWS services and resources in the account. This identity is called the AWS account *root user* and is accessed by signing in with the email address and password that you used to create the account. We strongly recommend that you don't use the root user for your everyday tasks. Safeguard your root user credentials and use them to perform the tasks that only the root user can perform. For the complete list of tasks that require you to sign in as the root user, see Tasks that require root user credentials in the *IAM User Guide*.

Federated identity

As a best practice, require human users, including users that require administrator access, to use federation with an identity provider to access AWS services by using temporary credentials.

A *federated identity* is a user from your enterprise user directory, a web identity provider, the AWS Directory Service, the Identity Center directory, or any user that accesses AWS services by using credentials provided through an identity source. When federated identities access AWS accounts, they assume roles, and the roles provide temporary credentials.

For centralized access management, we recommend that you use AWS IAM Identity Center. You can create users and groups in IAM Identity Center, or you can connect and synchronize to a set of users and groups in your own identity source for use across all your AWS accounts and applications. For information about IAM Identity Center, see What is IAM Identity Center? in the *AWS IAM Identity Center User Guide*.

IAM users and groups

An *IAM user* is an identity within your AWS account that has specific permissions for a single person or application. Where possible, we recommend relying on temporary credentials instead of creating IAM users who have long-term credentials such as passwords and access keys. However, if you have specific use cases that require long-term credentials with IAM users, we recommend that you rotate access keys. For more information, see Rotate access keys regularly for use cases that require long-term credentials in the *IAM User Guide*.

An *IAM group* is an identity that specifies a collection of IAM users. You can't sign in as a group. You can use groups to specify permissions for multiple users at a time. Groups make permissions easier to manage for large sets of users. For example, you could have a group named *IAMAdmins* and give that group permissions to administer IAM resources.

Users are different from roles. A user is uniquely associated with one person or application, but a role is intended to be assumable by anyone who needs it. Users have permanent long-term credentials, but roles provide temporary credentials. To learn more, see Use cases for IAM users in the *IAM User Guide*.

IAM roles

An *IAM role* is an identity within your AWS account that has specific permissions. It is similar to an IAM user, but is not associated with a specific person. To temporarily assume an IAM role in the AWS Management Console, you can switch from a user to an IAM role (console). You can assume a role by calling an AWS CLI or AWS API operation or by using a custom URL. For more information about methods for using roles, see Methods to assume a role in the *IAM User Guide*.

IAM roles with temporary credentials are useful in the following situations:

- **Federated user access** – To assign permissions to a federated identity, you create a role and define permissions for the role. When a federated identity authenticates, the identity is associated with the role and is granted the permissions that are defined by the role. For information about roles for federation, see Create a role for a third-party identity provider

(federation) in the *IAM User Guide*. If you use IAM Identity Center, you configure a permission set. To control what your identities can access after they authenticate, IAM Identity Center correlates the permission set to a role in IAM. For information about permissions sets, see Permission sets in the *AWS IAM Identity Center User Guide*.

- **Temporary IAM user permissions** – An IAM user or role can assume an IAM role to temporarily take on different permissions for a specific task.
- **Cross-account access** – You can use an IAM role to allow someone (a trusted principal) in a different account to access resources in your account. Roles are the primary way to grant cross-account access. However, with some AWS services, you can attach a policy directly to a resource (instead of using a role as a proxy). To learn the difference between roles and resource-based policies for cross-account access, see Cross account resource access in IAM in the *IAM User Guide*.
- **Cross-service access** – Some AWS services use features in other AWS services. For example, when you make a call in a service, it's common for that service to run applications in Amazon EC2 or store objects in Amazon S3. A service might do this using the calling principal's permissions, using a service role, or using a service-linked role.
  - **Forward access sessions (FAS)** – When you use an IAM user or role to perform actions in AWS, you are considered a principal. When you use some services, you might perform an action that then initiates another action in a different service. FAS uses the permissions of the principal calling an AWS service, combined with the requesting AWS service to make requests to downstream services. FAS requests are only made when a service receives a request that requires interactions with other AWS services or resources to complete. In this case, you must have permissions to perform both actions. For policy details when making FAS requests, see Forward access sessions.
  - **Service role** – A service role is an IAM role that a service assumes to perform actions on your behalf. An IAM administrator can create, modify, and delete a service role from within IAM. For more information, see Create a role to delegate permissions to an AWS service in the *IAM User Guide*.
  - **Service-linked role** – A service-linked role is a type of service role that is linked to an AWS service. The service can assume the role to perform an action on your behalf. Service-linked roles appear in your AWS account and are owned by the service. An IAM administrator can view, but not edit the permissions for service-linked roles.
- **Applications running on Amazon EC2** – You can use an IAM role to manage temporary credentials for applications that are running on an EC2 instance and making AWS CLI or AWS API requests. This is preferable to storing access keys within the EC2 instance. To assign an AWS role to an EC2 instance and make it available to all of its applications, you create an instance profile

that is attached to the instance. An instance profile contains the role and enables programs that are running on the EC2 instance to get temporary credentials. For more information, see Use an IAM role to grant permissions to applications running on Amazon EC2 instances in the *IAM User Guide*.

Managing access using policies

You control access in AWS by creating policies and attaching them to AWS identities or resources. A policy is an object in AWS that, when associated with an identity or resource, defines their permissions. AWS evaluates these policies when a principal (user, root user, or role session) makes a request. Permissions in the policies determine whether the request is allowed or denied. Most policies are stored in AWS as JSON documents. For more information about the structure and contents of JSON policy documents, see Overview of JSON policies in the *IAM User Guide*.

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

By default, users and roles have no permissions. To grant users permission to perform actions on the resources that they need, an IAM administrator can create IAM policies. The administrator can then add the IAM policies to roles, and users can assume the roles.

IAM policies define permissions for an action regardless of the method that you use to perform the operation. For example, suppose that you have a policy that allows the `iam:GetRole` action. A user with that policy can get role information from the AWS Management Console, the AWS CLI, or the AWS API.

Identity-based policies

Identity-based policies are JSON permissions policy documents that you can attach to an identity, such as an IAM user, group of users, or role. These policies control what actions users and roles can perform, on which resources, and under what conditions. To learn how to create an identity-based policy, see Define custom IAM permissions with customer managed policies in the *IAM User Guide*.

Identity-based policies can be further categorized as *inline policies* or *managed policies*. Inline policies are embedded directly into a single user, group, or role. Managed policies are standalone policies that you can attach to multiple users, groups, and roles in your AWS account. Managed policies include AWS managed policies and customer managed policies. To learn how to choose between a managed policy or an inline policy, see Choose between managed policies and inline policies in the *IAM User Guide*.

Resource-based policies

Resource-based policies are JSON policy documents that you attach to a resource. Examples of resource-based policies are IAM *role trust policies* and Amazon S3 *bucket policies*. In services that support resource-based policies, service administrators can use them to control access to a specific resource. For the resource where the policy is attached, the policy defines what actions a specified principal can perform on that resource and under what conditions. You must specify a principal in a resource-based policy. Principals can include accounts, users, roles, federated users, or AWS services.

Resource-based policies are inline policies that are located in that service. You can't use AWS managed policies from IAM in a resource-based policy.

Access control lists (ACLs)

Access control lists (ACLs) control which principals (account members, users, or roles) have permissions to access a resource. ACLs are similar to resource-based policies, although they do not use the JSON policy document format.

Amazon S3, AWS WAF, and Amazon VPC are examples of services that support ACLs. To learn more about ACLs, see Access control list (ACL) overview in the *Amazon Simple Storage Service Developer Guide*.

Other policy types

AWS supports additional, less-common policy types. These policy types can set the maximum permissions granted to you by the more common policy types.

- **Permissions boundaries** – A permissions boundary is an advanced feature in which you set the maximum permissions that an identity-based policy can grant to an IAM entity (IAM user or role). You can set a permissions boundary for an entity. The resulting permissions are the intersection of an entity's identity-based policies and its permissions boundaries. Resource-based policies that specify the user or role in the `Principal` field are not limited by the permissions boundary. An explicit deny in any of these policies overrides the allow. For more information about permissions boundaries, see Permissions boundaries for IAM entities in the *IAM User Guide*.
- **Service control policies (SCPs)** – SCPs are JSON policies that specify the maximum permissions for an organization or organizational unit (OU) in AWS Organizations. AWS Organizations is a service for grouping and centrally managing multiple AWS accounts that your business owns. If you enable all features in an organization, then you can apply service control policies (SCPs) to

any or all of your accounts. The SCP limits permissions for entities in member accounts, including each AWS account root user. For more information about Organizations and SCPs, see Service control policies in the *AWS Organizations User Guide*.

- **Resource control policies (RCPs)** – RCPs are JSON policies that you can use to set the maximum available permissions for resources in your accounts without updating the IAM policies attached to each resource that you own. The RCP limits permissions for resources in member accounts and can impact the effective permissions for identities, including the AWS account root user, regardless of whether they belong to your organization. For more information about Organizations and RCPs, including a list of AWS services that support RCPs, see Resource control policies (RCPs) in the *AWS Organizations User Guide*.
- **Session policies** – Session policies are advanced policies that you pass as a parameter when you programmatically create a temporary session for a role or federated user. The resulting session's permissions are the intersection of the user or role's identity-based policies and the session policies. Permissions can also come from a resource-based policy. An explicit deny in any of these policies overrides the allow. For more information, see Session policies in the *IAM User Guide*.

Multiple policy types

When multiple types of policies apply to a request, the resulting permissions are more complicated to understand. To learn how AWS determines whether to allow a request when multiple policy types are involved, see Policy evaluation logic in the *IAM User Guide*.

How AWS services work with IAM

To get a high-level view of how AWS services work with most IAM features, see AWS services that work with IAM in the *IAM User Guide*.

To learn how to use a specific AWS service with IAM, see the security section of the relevant service's User Guide.

Troubleshooting AWS identity and access

Use the following information to help you diagnose and fix common issues that you might encounter when working with AWS and IAM.

Topics

- I am not authorized to perform an action in AWS
- I am not authorized to perform iam:PassRole

- I want to allow people outside of my AWS account to access my AWS resources

I am not authorized to perform an action in AWS

If you receive an error that you're not authorized to perform an action, your policies must be updated to allow you to perform the action.

The following example error occurs when the `mateojackson` IAM user tries to use the console to view details about a fictional *`my-example-widget`* resource but doesn't have the fictional `awes:`*`GetWidget`* permissions.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 awes:GetWidget on resource: my-example-widget
```

In this case, the policy for the `mateojackson` user must be updated to allow access to the *`my-example-widget`* resource by using the `awes:`*`GetWidget`* action.

If you need help, contact your AWS administrator. Your administrator is the person who provided you with your sign-in credentials.

I am not authorized to perform iam:PassRole

If you receive an error that you're not authorized to perform the `iam:PassRole` action, your policies must be updated to allow you to pass a role to AWS.

Some AWS services allow you to pass an existing role to that service instead of creating a new service role or service-linked role. To do this, you must have permissions to pass the role to the service.

The following example error occurs when an IAM user named `marymajor` tries to use the console to perform an action in AWS. However, the action requires the service to have permissions that are granted by a service role. Mary does not have permissions to pass the role to the service.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

In this case, Mary's policies must be updated to allow her to perform the `iam:PassRole` action.

If you need help, contact your AWS administrator. Your administrator is the person who provided you with your sign-in credentials.

I want to allow people outside of my AWS account to access my AWS resources

You can create a role that users in other accounts or people outside of your organization can use to access your resources. You can specify who is trusted to assume the role. For services that support resource-based policies or access control lists (ACLs), you can use those policies to grant people access to your resources.

To learn more, consult the following:

- To learn whether AWS supports these features, see How AWS services work with IAM.
- To learn how to provide access to your resources across AWS accounts that you own, see Providing access to an IAM user in another AWS account that you own in the *IAM User Guide*.
- To learn how to provide access to your resources to third-party AWS accounts, see Providing access to AWS accounts owned by third parties in the *IAM User Guide*.
- To learn how to provide access through identity federation, see Providing access to externally authenticated users (identity federation) in the *IAM User Guide*.
- To learn the difference between using roles and resource-based policies for cross-account access, see Cross account resource access in IAM in the *IAM User Guide*.

Compliance Validation for this AWS Product or Service

To learn whether an AWS service is within the scope of specific compliance programs, see AWS services in Scope by Compliance Program and choose the compliance program that you are interested in. For general information, see AWS Compliance Programs.

You can download third-party audit reports using AWS Artifact. For more information, see Downloading Reports in AWS Artifact.

Your compliance responsibility when using AWS services is determined by the sensitivity of your data, your company's compliance objectives, and applicable laws and regulations. AWS provides the following resources to help with compliance:

- Security Compliance & Governance – These solution implementation guides discuss architectural considerations and provide steps for deploying security and compliance features.
- HIPAA Eligible Services Reference – Lists HIPAA eligible services. Not all AWS services are HIPAA eligible.

- AWS Compliance Resources – This collection of workbooks and guides might apply to your industry and location.
- AWS Customer Compliance Guides – Understand the shared responsibility model through the lens of compliance. The guides summarize the best practices for securing AWS services and map the guidance to security controls across multiple frameworks (including National Institute of Standards and Technology (NIST), Payment Card Industry Security Standards Council (PCI), and International Organization for Standardization (ISO)).
- Evaluating Resources with Rules in the *AWS Config Developer Guide* – The AWS Config service assesses how well your resource configurations comply with internal practices, industry guidelines, and regulations.
- AWS Security Hub – This AWS service provides a comprehensive view of your security state within AWS. Security Hub uses security controls to evaluate your AWS resources and to check your compliance against security industry standards and best practices. For a list of supported services and controls, see Security Hub controls reference.
- Amazon GuardDuty – This AWS service detects potential threats to your AWS accounts, workloads, containers, and data by monitoring your environment for suspicious and malicious activities. GuardDuty can help you address various compliance requirements, like PCI DSS, by meeting intrusion detection requirements mandated by certain compliance frameworks.
- AWS Audit Manager – This AWS service helps you continuously audit your AWS usage to simplify how you manage risk and compliance with regulations and industry standards.

This AWS product or service follows the shared responsibility model through the specific Amazon Web Services (AWS) services it supports. For AWS service security information, see the AWS service security documentation page and AWS services that are in scope of AWS compliance efforts by compliance program.

Resilience for this AWS Product or Service

The AWS global infrastructure is built around AWS Regions and Availability Zones.

AWS Regions provide multiple physically separated and isolated Availability Zones, which are connected with low-latency, high-throughput, and highly redundant networking.

With Availability Zones, you can design and operate applications and databases that automatically fail over between zones without interruption. Availability Zones are more highly available, fault tolerant, and scalable than traditional single or multiple data center infrastructures.

For more information about AWS Regions and Availability Zones, see AWS Global Infrastructure.

This AWS product or service follows the shared responsibility model through the specific Amazon Web Services (AWS) services it supports. For AWS service security information, see the AWS service security documentation page and AWS services that are in scope of AWS compliance efforts by compliance program.

Infrastructure Security for this AWS Product or Service

This AWS product or service uses managed services, and therefore is protected by the AWS global network security. For information about AWS security services and how AWS protects infrastructure, see AWS Cloud Security. To design your AWS environment using the best practices for infrastructure security, see Infrastructure Protection in *Security Pillar AWS Well-Architected Framework*.

You use AWS published API calls to access this AWS Product or Service through the network. Clients must support the following:

- Transport Layer Security (TLS). We require TLS 1.2 and recommend TLS 1.3.
- Cipher suites with perfect forward secrecy (PFS) such as DHE (Ephemeral Diffie-Hellman) or ECDHE (Elliptic Curve Ephemeral Diffie-Hellman). Most modern systems such as Java 7 and later support these modes.

Additionally, requests must be signed by using an access key ID and a secret access key that is associated with an IAM principal. Or you can use the AWS Security Token Service (AWS STS) to generate temporary security credentials to sign requests.

This AWS product or service follows the shared responsibility model through the specific Amazon Web Services (AWS) services it supports. For AWS service security information, see the AWS service security documentation page and AWS services that are in scope of AWS compliance efforts by compliance program.

Enforcing a minimum TLS version in the AWS SDK for Go

To add increased security when communicating with AWS services, you should configure your client to use TLS 1.2 or later.

Note

As of Go 1.18, the TLS configuration used by the `net/http#Client` defaults to TLS 1.2 as a minimum, and disables support for TLS 1.0 and TLS 1.1.

How do I set my TLS version?

You can set the TLS version to 1.2 using the following code.

1. Create a custom HTTP transport to require a minimum version of TLS 1.2

```
tr := &http.Transport{
    TLSClientConfig: &tls.Config{
        MinVersion: tls.VersionTLS12,
    },
}
```

2. Configure the transport.

```
// In Go versions earlier than 1.13
err := http2.ConfigureTransport(tr)
if err != nil {
    fmt.Println("Got an error configuring HTTP transport")
    fmt.Println(err)
    return
}

// In Go versions later than 1.13
tr.ForceAttemptHTTP2 = true
```

3. Create an HTTP client with the configured transport, and use that to create a session. REGION is the AWS Region, such as *us-west-2*.

```
client := http.Client{Transport: tr}

sess := session.Must(session.NewSession(&aws.Config{
    Region:     &REGION,
    HTTPClient: &client,
}))
```

4. Use the following function to confirm your TLS version.

```
func GetTLSVersion(tr *http.Transport) string {
    switch tr.TLSClientConfig.MinVersion {
    case tls.VersionTLS10:
        return "TLS 1.0"
    case tls.VersionTLS11:
        return "TLS 1.1"
    case tls.VersionTLS12:
        return "TLS 1.2"
    case tls.VersionTLS13:
        return "TLS 1.3"
    }

    return "Unknown"
}
```

5. Confirm your TLS version by calling *GetTLSVersion*.

```
if tr, ok := sess.Config.HTTPClient.Transport.(*http.Transport); ok {
    log.Printf("Client uses %v", GetTLSVersion(tr))
}
```

Amazon S3 Encryption Client Migration

This topic shows how to migrate your applications from Version 1 (V1) of the Amazon Simple Storage Service (Amazon S3) encryption client to Version 2 (V2) and ensure application availability throughout the migration process.

Migration Overview

This migration happens in two phases:

1. **Update existing clients to read new formats.** First, deploy an updated version of the AWS SDK for Go to your application. This allows existing V1 encryption clients to decrypt objects written by the new V2 clients. If your application uses multiple AWS SDKs, you must upgrade each SDK separately.

2. **Migrate encryption and decryption clients to V2.** Once all of your V1 encryption clients can read new formats, you can migrate your existing encryption and decryption clients to their respective V2 versions.

Update Existing Clients to Read New Formats

The V2 encryption client uses encryption algorithms that older versions of the client do not support. The first step in the migration is to update your V1 decryption clients to SDK version v1.34.0 or later. After completing this step, your application's V1 clients will be able to decrypt objects encrypted by V2 encryption clients.

Update Applications That Use Go Modules

If your applications use the Go module dependency system introduced in Go 1.11, you must take the following steps to update your application to the latest release of the AWS SDK for Go.

First, update your SDK module dependency.

```
$ go get github.com/aws/aws-sdk-go@latest
```

Next, validate your dependency has correctly updated to the required minimum version.

```
$ go list -m github.com/aws/aws-sdk-go
github.com/aws/aws-sdk-go v1.34.0
```

After you upgrade and validate your dependencies, deploy the application to your fleet. Once the rollout is complete, you can migrate your V1 encryption and decryption clients to V2.

Updating Applications That Use GOPATH

If your application uses `GOPATH` to manage its dependencies, you must take the following steps to update and verify that your application is using the minimum SDK release or later.

First, update your SDK `GOPATH` source code.

```
$ go get -u github.com/aws/aws-sdk-go
```

Next, ensure the SDK source path to the release (v1.34.0) or later (https://github.com/aws/aws-sdk-go/releases).

```
$ cd $GOPATH/src/github.com/aws/aws-sdk-go
$ git fetch
```

```
$ go checkout v1.34.0
```

After you upgrade your dependencies and verify, deploy the application to your fleet. Once the rollout is complete, you can migrate your V1 encryption and decryption clients to V2.

Migrate Encryption and Decryption Clients to V2

After updating your existing clients to read the new encryption formats, you can now proceed with safely updating your applications to the V2 encryption and decryption clients. The next series of steps will guide you through successfully migrating your code from V1 to V2.

Migrate Cipher Data Generators

Applications that use NewKMSKeyGenerator or NewKMSKeyGeneratorWithMatDesc functions for constructing a CipherDataGenerator need to migrate their usage to NewKMSContextKeyGenerator. This migration is required because support for the former `CipherDataGenerators` have been deprecated. Attempting to construct a V2 client using the old methods will result in an error during client construction.

Example: Migrate NewKMSKeyGenerator

Pre-migration

```
sess := session.Must(session.NewSession())
kmsClient := kms.New(sess)
cmkID := "1234abcd-12ab-34cd-56ef-1234567890ab"

cipherDataGenerator := s3crypto.NewKMSKeyGenerator(kmsClient, cmkID)
```

Post-migration

```
sess := session.Must(session.NewSession())
kmsClient := kms.New(sess)
cmkID := "1234abcd-12ab-34cd-56ef-1234567890ab"
var matDesc s3crypto.MaterialDescription

// changed NewKMSKeyGenerator to NewKMSContextKeyGenerator
cipherDataGenerator := s3crypto.NewKMSContextKeyGenerator(kmsClient, cmkID, matDesc)
```

Example: Migrate NewKMSKeyGeneratorWithMatDesc

Pre-migration

```
sess := session.Must(session.NewSession())
kmsClient := kms.New(sess)
cmkID := "1234abcd-12ab-34cd-56ef-1234567890ab"
matDesc := s3crypto.MaterialDescription{
    "custom-key": aws.String("custom value"),
}

cipherDataGenerator := s3crypto.NewKMSKeyGeneratorWithMatDesc(kmsClient, cmkID,
 matDesc)
```

Post-migration

```
sess := session.Must(session.NewSession())
kmsClient := kms.New(sess)
cmkID := "1234abcd-12ab-34cd-56ef-1234567890ab"
matDesc := s3crypto.MaterialDescription{
    "custom-key": aws.String("custom value"),
}

// changed NewKMSKeyGeneratorWithMatDesc to NewKMSContextKeyGenerator
cipherDataGenerator := s3crypto.NewKMSContextKeyGenerator(kmsClient, cmkID, matDesc)
```

Migrate Content Cipher Builders

Applications that use AESCBCContentCipherBuilder to construct an AES/CBC content cipher must migrate to AES/GCM using AESGCMContentCipherBuilderV2.

Applications that use AESGCMContentCipherBuilder to construct the AES/GCM content cipher must migrate to AESGCMContentCipherBuilderV2.

Attempting to use the deprecated content cipher builders with the V2 encryption client will result in a runtime error during the client construction.

> **Important**
>
> Due to limitations in the Go standard library, objects must be read into memory when performing encryption and decryption operations using AES/GCM. Caution must be taken to ensure that your application does not experience memory allocation failures.

Example: Migrate AESCBCContentCipherBuilder

Pre-migration

```
contentCipherBuilder := s3crypto.AESCBCContentCipherBuilder(cipherDataGenerator,
 s3crypto.AESCBCPadder)
```

Post-migration

```
contentCipherBuilder := s3crypto.AESGCMContentCipherBuilderV2(cipherDataGenerator)
```

Example: Migrate AESGCMContentCipherBuilder

Pre-migration

```
contentCipherBuilder := s3crypto.AESGCMContentCipherBuilder(cipherDataGenerator,
 s3crypto.AESCBCPadder)
```

Post-migration

```
contentCipherBuilder := s3crypto.AESGCMContentCipherBuilderV2(cipherDataGenerator)
```

Migrate Encryption Client Constructors

The V2 encryption client constructor adds an error interface type as an additional return parameter. An error can be returned during V2 client construction if the client is given a deprecated ContentCipherBuilder or CipherDataGenerator. Review the migration steps required to migrate these types.

Example

Pre-migration

```
encryptionClient := s3crypto.NewEncryptionClient(sess, contentCipherBuilder)
```

Post-migration

```
encryptionClient, err := s3crypto.NewEncryptionClientV2(sess, contentCipherBuilder)
if err != nil {
    panic(err)
```

```
}
```

Migrate Custom Encryption Client Configurations

Clients that utilize custom client configuration options will be required to update their function argument signatures to use EncryptionClientOptions for setting custom options such as an alternative SaveStrategy.

Pre-migration

```
// example setting an alternative SaveStrategy
encryptionClient := s3crypto.NewEncryptionClient(cipherDataGenerator,
 contentCipherBuilder, func(o *s3crypto.EncryptionClient) {
    // Set Instruction File Save Strategy
    o.SaveStrategy = s3crypto.S3SaveStrategy{Client: s3.New(sess)}
})
```

Post-migration

```
// example setting an alternative SaveStrategy
encryptionClient, err := s3crypto.NewEncryptionClientV2(sess, contentCipherBuilder,
 func(o *s3crypto.EncryptionClientOptions) {
    // Set Instruction File Save Strategy
    o.SaveStrategy = s3crypto.S3SaveStrategy{Client: s3.New(sess)}
})
if err != nil {
    panic(err)
}
```

Migrate Decryption Client Constructors

The V2 decryption client now requires that an application registers the content ciphers and key wrapping algorithms that it wants to decrypt. This is registration is done using the CryptoRegistry, and a series of registration helper functions are included to enable the V2 decryption client to decrypt objects written in either the V1 or V2 encryption formats.

Step 1: Instantiate a CryptoRegistry

```
registry := s3crypto.NewCryptoRegistry()
```

Step 2: Register required content decryption algorithms

To read content encrypted using V1 AESGCMContentCipherBuilder or V2 AESGCMContentCipherBuilderV2:

```
if err := s3crypto.RegisterAESGCMContentCipher(registry); err != nil {
   panic(err)
}
```

To read content encrypted using V1 AESCBCContentCipherBuilder:

```
padder := s3crypto.AESCBCPadder // Use the padder provided to
 AESCBCContentCipherBuilder

 if err := s3crypto.RegisterAESCBCContentCipher(registry, padder); err != nil {
     panic(err)
 }
```

To read custom content cipher implementations:

If your applications implements or uses a custom content cipher implementation, you may register that implementation using the CryptoRegistry's AddCEK method. If you require custom padders for your cipher, they can be registered using AddPadder.

```
if err := registry.AddCEK("CustomCEK", NewCustomCEK); err != nil {
    panic(err)
}

if err := registry.AddPadder("CustomPadder", NewCustomPadder); err != nil {
    panic(err)
}
```

Step 3: Register required key wrapping algorithms

To read keys created using the V2 NewKMSContextKeyGenerator:

Your application can opt to limit the CMK that is used when calling the KMS `Decrypt` API. Two registration functions allow the selection of the desired behavior. RegisterKMSContextWrapWithCMK and RegisterKMSContextWrapWithAnyCMK. Only one of these two methods should be used, and attempting to use both functions with a single registry will result in a runtime error.

```
// Use RegisterKMSContextWrapWithCMK to limit the KMS Decrypt to a single CMK
```

```
if err := s3crypto.RegisterKMSContextWrapWithCMK(registry, kms.New(sess), "key-id");
 err != nil {
    panic(err)
}

// Use RegisterKMSContextWrapWithAnyCMK to allow the KMS Decrypt call for any CMK
if err := s3crypto.RegisterKMSContextWrapWithAnyCMK(registry, kms.New(sess)); err !=
 nil {
    panic(err)
}
```

To read keys created using the V1 NewKMSKeyGenerator or NewKMSKeyGeneratorWithMatDesc:

Your application can opt to limit the CMK that is used when calling the KMS Decrypt API. Two registration functions allow the selection of the desired behavior. RegisterKMSWrapWithCMK and RegisterKMSWrapWithAnyCMK. Use only one of these methods. Attempting to register both functions into the registry will result in a runtime error.

```
// Use RegisterKMSWrapWithCMK to limit the KMS Decrypt Call to a single CMK
if err := s3crypto.RegisterKMSWrapWithCMK(registry, kms.New(sess), "key-id"); err !=
 nil {
    panic(err)
}

// Use RegisterKMSWrapWithAnyCMK to allow KMS Decrypt call for any CMK
if err := s3crypto.RegisterKMSWrapWithAnyCMK(registry, kms.New(sess)); err != nil {
    panic(err)
}
```

To read custom key wrapping algorithm implementations:

If your applications implements or uses a custom key wrapping implementation, you may register that implementation using the CryptoRegistry's AddWrap method.

```
if err := registry.AddWrap("CustomWrap", NewCustomWrap); err != nil {
    panic(err)
}
```

Step 4: Construct the client

After registering your applications required content decryption and key wrapping algorithms to the CryptoRegistry, you can now construct a V2 decryption client using NewDecryptionClientV2.

```
decryptionClient, err := s3crypto.NewDecryptionClientV2(sess, registry)
if err != nil {
    panic(err)
}
```

Migrating Custom Decryption Client Configurations

Clients that use custom client configuration options are required to update their functional argument signatures to use DecryptionClientOptions for setting custom options, such as an alternative LoadStrategy.

Example

Pre-migration

```
// example setting an alternative LoadStrategy
decryptionClient := s3crypto.NewDecryptionClient(sess, func(o
 *s3crypto.DecryptionClient) {
    // Set Instruction File Load Strategy
    o.LoadStrategy = s3crypto.S3LoadStrategy{Client: s3.New(sess)}
})
```

Post-migration

```
// example setting an alternative LoadStrategy
decryptionClient, err := s3crypto.NewDecryptionClientV2(sess, registry, func(o
 *s3crypto.DecryptionClientOptions) {
    // Set Instruction File Load Strategy
    o.LoadStrategy = s3crypto.S3LoadStrategy{Client: s3.New(sess)}
})
if err != nil {
    panic(err)
}
```

After you complete this migration, you can proceed to testing and deployment using your application's best practices. After deploying your application deployment, you will have successfully migrated it from the V1 to V2 Amazon S3 encryption clients.

Document History

This topic describes important changes to the AWS SDK for Go Developer Guide over the course of its history.

To view the list of changes to the AWS SDK for Go and its documentation, see the CHANGELOG.md file in the `aws/aws-sdk-go` repository in GitHub.

Last documentation update: Dec 29, 2021

Feb 7, 2022

Removed SDK Metrics material. This feature was deprecated on December 31, 2021.

Jul 10, 2017

Added the Document History topic.

June 14, 2017

Updated error handling example code.

April 14, 2017

Added S3 policy example.

April 11, 2017

Added S3 bucket ACL examples.

April 7, 2017

Added samples of getting/setting bucket and bucket object ACLs.

March 28, 2017

Updated basic Amazon S3 bucket examples.

February 27, 2017

Added an example of AssumeRoleTokenProvider.

November 2, 2016

Added SDK setters examples.

October 23, 2016

Changed deprecated session.New to session.NewSession.

September 29, 2016

Added a new topic for handling service errors from GitHub repository.

September 28, 2016

Added an example of Go extending SDK for context.Context.

August 31, 2016

Initial release of the AWS SDK for Go Developer Guide.