AWS SDK for Ruby

開発者ガイド

Copyright © 2018 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon's trademarks and trade dress may not be used in connection with any product or service that is not Amazon's, in any manner that is likely to cause confusion among customers, or in any manner that disparages or discredits Amazon. All other trademarks not owned by Amazon are the property of their respective owners, who may or may not be affiliated with, connected to, or sponsored by Amazon.

Table of Contents

AWS SDK for Ruby Developer Guide

AWS SDK for Ruby へようこそ。

AWS SDK for Ruby は、Amazon Simple Storage Service、Amazon Elastic Compute Cloud、および Amazon DynamoDB を含むほとんどすべての AWS のサービスに Ruby クラスを提供して、コーディングの複雑さを排除します。AWS SDK for Ruby でサポートされているサービスの完全な一覧については、AWS SDK for Ruby Readme ファイルの「サポートされるサービス」セクションを参照してください。

本ガイドについて

AWS SDK for Ruby Developer Guide は、AWS のサービスを使用する Ruby アプリケーションを作成するため、AWS SDK for Ruby のインストール方法、セットアップ方法、および使用方法に関する情報を提供しています。

このガイドには以下のセクションがあります。

その他のドキュメントとリソース

AWS SDK for Ruby の開発者向けのリソースについて、さらに詳しくは以下を参照してください。

- AWS SDK for Ruby API リファレンス
- 開発者ブログ
- 開発者フォーラム (フォーラムにアクセスするには AWS アカウントが必要)
- Gitter チャネル
- https://twitter.com/awsforruby@awsforruby の Twitter
- GitHub で
  - リリース (ソース、gem、ドキュメントが含まれます)
  - 送信元
  - 変更ログ

- 移行ガイド
- 問題点
- 機能のリクエスト
- アップグレード関する注意

AWS クラウドへのデプロイ

AWS Elastic Beanstalk、AWS OpsWorks、AWS CodeDeploy などの AWS のサービスを使用して、アプリケーションを AWS クラウドにデプロイできます。Elastic Beanstalk で Ruby アプリケーションをデプロイするには、Elastic Beanstalk Developer Guideの「Ruby の使用」を参照してください。AWS OpsWorks で Ruby on Rails アプリケーションをデプロイするには、「AWS OpsWorks に Ruby on Rails アプリケーションをデプロイする」を参照してください。AWS デプロイサービスの概要については、AWS のデプロイオプションの概要を参照してください。

AWS SDK for Ruby の開始方法

AWS SDK for Ruby を初めて利用する場合は、ここから始める必要があります。このセクションでは、Amazon S3 にアクセスする Ruby アプリケーションを作成するための SDK のインストール、セットアップ、使用の方法について説明しています。

トピック

AWS SDK for Ruby を使用するためのクイックスタートガイド

このセクションでは、Amazon S3 のバケットのリストを表示するシンプルな Ruby アプリケーションを作成するために AWS SDK for Ruby を使用する方法を示します。

- SDK をインストールしていない場合は、AWS SDK for Ruby のインストール (p. 4)を参照してください。
- SDK を設定していない場合は、AWS SDK for Ruby の設定 (p. 5)を参照してください。

コードを記述する

次の例では、最大で 50 個のバケットの名前をリスト表示します。コードをコピーし、`buckets.rb` という名前で保存します。リソースのオブジェクトが `us-west-2` リージョンで作成されますが、Amazon S3 はリージョンに関係なく、アクセスできるバケットを返すことに注意してください。

```
require 'aws-sdk'

s3 = Aws::S3::Resource.new(region: 'us-west-2')

s3.buckets.limit(50).each do |b|
  puts "#{b.name}"
end
```

コードを実行します。

`buckets.rb` を実行するため、以下のコマンドを入力します。

```
ruby buckets.rb
```

Windows ユーザーの場合

Windows の SSL 証明書を使用して Ruby コードを実行すると、次のようなエラーが表示されます。

```
C:\Ruby>ruby buckets.rb
C:/Ruby200-x64/lib/ruby/2.0.0/net/http.rb:921:in `connect': SSL_connect returned=1
 errno=0 state=SSLv3 read server certificate B: certificate verify failed
 (Seahorse::Client::NetworkingError)
         from C:/Ruby200-x64/lib/ruby/2.0.0/net/http.rb:921:in `block in connect'

         from C:/Ruby200-x64/lib/ruby/2.0.0/timeout.rb:66:in `timeout'
         from C:/Ruby200-x64/lib/ruby/2.0.0/net/http.rb:921:in `connect'
         from C:/Ruby200-x64/lib/ruby/2.0.0/net/http.rb:862:in `do_start'
         from C:/Ruby200-x64/lib/ruby/2.0.0/net/http.rb:857:in `start'
...
```

この問題を修正するには、Ruby のソースファイルの最初の AWS 呼び出しの前のどこかに次の行を追加します。

```
Aws.use_bundled_cert!
```

AWS SDK for Ruby のインストール

このセクションには、前提条件とインストール手順が含まれています。

前提条件

AWS SDK for Ruby をインストールする前に、AWS アカウントと Ruby バージョン 1.9 以降が必要になります。

AWS アカウントをお持ちでない場合は、次に説明する手順にしたがってアカウントを作成してください。

AWS にサインアップするには

1. http://aws.amazon.com/ を開き、[AWS アカウントの作成] を選択します。
2. オンラインの手順に従います。

AWS SDK for Ruby のインストール

プロジェクトが Bundler を使用している場合、プロジェクトに AWS SDK for Ruby を追加するために `Gemfile` に次の行を追加します。

```
gem 'aws-sdk'
```

Bundler を使用していない場合、最も簡単な SDK のインストール方法は RubyGems を使用することです。SDK の最新バージョンをインストールするには、次のコマンドを使用します。

```
gem install aws-sdk
```

前のコマンドが、Unix ベースのシステム上で失敗した場合、次のコマンドに示すように、`sudo` を使用して SDK をインストールします。

```
sudo gem install aws-sdk
```

AWS SDK for Ruby の設定

AWS SDK for Ruby の設定方法を学習します。SDK を使用するには、AWS 認証情報を設定するか、または AWS STS アクセストークンを作成し、使用する AWS リージョンを設定します。

AWS 認証情報の設定

AWS SDK for Ruby を使用して、AWS サービスを呼び出す前に、AWS のサービスおよびリソースへのアクセスを確認するために SDK が使用する、AWS アクセス認証情報を設定する必要があります。

AWS SDK for Ruby は、次の手順で認証情報を検索します。

1. クライアントオブジェクト内で認証情報を設定する (p. 6)
2. Aws.config を使用して認証情報を設定する (p. 6)
3. 環境変数を使用して認証情報を設定する (p. 5)
4. 共有認証情報の設定 (p. 5)
5. IAM を使用して認証情報を設定する (p. 6)

以下のセクションでは、認証情報を設定できるさまざまな方法を最も柔軟な方法から説明します。AWS 認証情報と認証情報管理の推奨される方法の詳細については、Amazon Web Services General Referenceの「AWS セキュリティの認証情報」を参照してください。

共有認証情報の設定

ローカルシステム上の AWS 認証情報プロファイルのファイルで共有認証情報を設定します。

Linux または OS X のような、Unix ベースのシステムでは、このファイルは次の場所にあります。

```
~/.aws/credentials
```

Windows では、このファイルは次の場所にあります。

```
%HOMEPATH%\.aws\credentials
```

`default` が、これらの認証情報に付与されたデフォルトの設定プロファイルで、`your_access_key_id` がアクセスキーの値で、`your_secret_access_key` がシークレットアクセスキーの値である場合、このファイルは次の形式である必要があります。

```
[default]
aws_access_key_id = your_access_key_id
aws_secret_access_key = your_secret_access_key
```

環境変数を使用して認証情報を設定する

`AWS_ACCESS_KEY_ID` および `AWS_SECRET_ACCESS_KEY` 環境変数を設定する

`export` コマンドを使用して、Linux または OS X などの Unix ベースのシステム上でこれらの変数を設定します。次の例では、アクセスキーの値を `your_access_key_id` に設定し、シークレットアクセスキーの値を `your_secret_access_key` に設定します。

```
export AWS_ACCESS_KEY_ID=your_access_key_id
export AWS_SECRET_ACCESS_KEY=your_secret_access_key
```

Windows 上でこれらの変数を設定するには、次の例に示すように、`set` コマンドを使用します。

```
set AWS_ACCESS_KEY_ID=your_access_key_id
set AWS_SECRET_ACCESS_KEY=your_secret_access_key
```

Aws.config を使用して認証情報を設定する

`Aws.config` ハッシュで、値を更新することで、コードの認証情報を設定します。

次の例では、アクセスキーの値を `your_access_key_id`に設定し、シークレットアクセスキーの値を `your_secret_access_key` に設定します。その後作成するどのクライアントやリソースも、これらの認証情報を使用します。

```
Aws.config.update({
   credentials: Aws::Credentials.new('your_access_key_id', 'your_secret_access_key')
})
```

クライアントオブジェクト内で認証情報を設定する

AWS クライアントを作成する際、指定してコードの認証情報を設定します。

次の例では、アクセスキー `your_access_key_id` およびシークレットアクセスキー `your_secret_access_key` を使用して Amazon S3 クライアントを作成します。

```
s3 = Aws::S3::Client.new(
  access_key_id: 'your_access_key_id',
  secret_access_key: 'your_secret_access_key'
)
```

IAM を使用して認証情報を設定する

Amazon Elastic Compute Cloud インスタンスの場合、IAM ロールを指定し、そのロールへのアクセスを Amazon EC2 インスタンスに提供します。詳細については、Amazon EC2 User Guide for Linux Instancesの「Amazon EC2 の IAM ロール」または Amazon EC2 User Guide for Windows Instancesの「Amazon EC2 の IAM ロール」を参照してください。

AWS STS アクセストークンの作成

Aws::AssumeRoleCredentials メソッドを使用して、AWS STS アクセストークンを作成します。

次の例では、`linked::account::arn` が引き受けるロールの Amazon リソースネーム (ARN) で、`session-name` が引き受けたロールセッションの識別子である場合、アクセストークンを使用して Amazon S3 クライアントオブジェクトを作成します。

```
role_credentials = Aws::AssumeRoleCredentials.new(
  client: Aws::STS::Client.new,
  role_arn: "linked::account::arn",
  role_session_name: "session-name"
)

s3 = Aws::S3::Client.new(credentials: role_credentials)
```

リージョンの設定

ほとんどの AWS サービスを使用する際には リージョン を設定する必要があります。AWS リージョンは AWS の認証情報の設定と似た方法で設定できます。AWS SDK for Ruby は、次の手順でリージョンを検索します。

- クライアントまたはリソースオブジェクト内でのリージョンの設定 (p. 7)
- Aws.config を使用してリージョンを設定する (p. 7)
- 環境変数を使用してリージョンを設定する (p. 7)

このセクションのこれ以降の部分では、リージョンに設定する方法を最も柔軟な方法から説明します。

環境変数を使用してリージョンを設定する

`AWS_REGION` 環境変数を設定して、リージョンを設定します。

`export` コマンドを使用して、Linux または OS X などの Unix ベースのシステムでこの変数を設定します。次の例では、リージョンを `us-west-2` に設定します。

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

Windows でこれらの変数を設定するには、`set` コマンドを使用します。次の例では、リージョンを `us-west-2` に設定します。

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

Aws.config を使用してリージョンを設定する

`region` 値を `Aws.config` ハッシュに追加してリージョンを設定します。次の例は `Aws.config` ハッシュを更新して `us-west-1` リージョンを使用します。

```
Aws.config.update({region: 'us-west-1'})
```

その後作成するどのクライアントやリソースもこのリージョンにバインドされます。

クライアントまたはリソースオブジェクト内でのリージョンの設定

AWS クライアントまたはリソースを作成するときにリージョンを設定します。次の例では、`us-west-1` リージョンに Amazon S3 リソースオブジェクトを作成します。

```
s3 = Aws::S3::Resource.new(region: 'us-west-1')
```

標準外のエンドポイントを設定する

選択したリージョンで標準外エンドポイントを使用する必要がある場合は、`endpoint` エントリを `Aws.config` に追加するか、サービスクライアントまたはリソースオブジェクトを作成するときに `endpoint:` を設定します。次の例では、`other_endpoint` エンドポイントに Amazon S3 リソースオブジェクトを作成します。

```
s3 = Aws::S3::Resource.new(endpoint: other_endpoint)
```

AWS SDK for Ruby REPL の使用

開発者は `aws-sdk-core` gem の一部である `aws.rb`、インタラクティブなコマンドライン read-evaluate-print loop (REPL) コンソールツールを使用できます。

`aws.rb` は、インタラクティブな Ruby シェル (`irb`) を使用しますが、より強力な REPL 環境を提供する `pry` をインストールすることをお勧めします。

次のコマンドを使用して `pry` をインストールします。

```
gem install pry
```

`aws.rb` を使用するには、次の 2 つのコマンドラインの 1 つを使用して、コンソールウィンドウに呼び出します。

```
aws.rb
aws.rb -v
```

2 番目のコマンドラインは、AWS SDK for Ruby と AWS 間の通信に関する情報を提供する、包括的な HTTP ワイヤログのある REPL を呼び出します。ただし、このコマンドラインにより、オーバーヘッドが追加されるため、コードの実行が遅くなる可能性があるので注意して使用してください。

REPL は、各サービスクラスのヘルパーオブジェクトを定義します。サービスモジュール名を小文字変換して、ヘルパーオブジェクトの名前を取得します。たとえば、Amazon S3 および Amazon EC2 のヘルパーオブジェクトの名前は、それぞれ `s3` と `ec2` です。

Ruby on Rails で SDK を使用する

Ruby on Rails は、Ruby でウェブサイトを簡単に作成できるように Ruby のためのウェブ開発フレームワークを提供します。

AWS SDK for Ruby は、Rails との統合を容易にする gem を提供します。AWS Elastic Beanstalk、AWS OpsWorks、または AWS CodeDeploy を使用して、Rails アプリケーションを AWS クラウドにデプロイして実行できます。

AWS SDK for Ruby の Rails との統合

AWS は、AWS SDK for Ruby と Rails との統合をサポートする Gemfile `aws-sdk-rails` を提供しています。https://github.com/aws/aws-sdk-rails で GitHub リポジトリを表示できます。

次の例に示すように、アプリケーションの Gemfile に gem を追加します。

```
gem 'aws-sdk-rails'
```

gem には AWS SDK for Ruby が含まれるため、gem を追加するだけで Rails アプリケーションに AWS サポートを追加できます。

Amazon SES の ActionMailer のサポート

Rails プロジェクトの `config/environments` ファイル内の `aws-sdk-rails` gem を使用する場合 (たとえば、`config/environments/production.rb`)、次の例に示すように、`ActionMailer` クラスのバックエンドとして Amazon Simple Email Service (Amazon SES) を使用できます。

```
config.action_mailer.delivery_method = :aws_sdk
```

ActionMailer の詳細については、Ruby on Rails のウェブサイトにある アクションメーラーの基本を参照してください。

ログ記録

aws-sdk-rails gem は Rails.logger を使用するように SDK ロガーを設定します。

gem は、:info ログレベルを使用するように SDK ログメッセージも設定します。Aws.config ハッシュにある :log_level を設定することでログレベルを変更できます。次の例では、ログレベルを :debug に設定します。

```
Aws.config.update({log_level: :debug})
```

AWS SDK for Ruby バージョン 1 からバージョン 2 への移行

このトピックの目的は、AWS SDK for Ruby のバージョン 1 からバージョン 2 に移行できるようにすることです。

サイドバイサイドの使用

バージョン 1 の AWS SDK for Ruby をバージョン 2 に置き換える必要はありません。同じアプリケーションで併用できます。詳細については、このブログの投稿を参照してください。

簡単な例を示します。

```
require 'aws-sdk-v1' # version 1
require 'aws-sdk'    # version 2

s3 = AWS::S3::Client.new # version 1
s3 = Aws::S3::Client.new # version 2
```

動作している既存のバージョン 1 のコードを、バージョン 2 の SDK を使用するように書き換える必要はありません。有効な移行戦略は、バージョン 2 SDK に対して新しいコードを書くことです。

一般的な違い

バージョン 2 では、いくつかの重要なメソッドがバージョン 1 とは異なります。

- ルート名前空間の違い。Aws と AWS です。これにより、サイドバイ大度の使用が可能になります。
- Aws.config がメソッドではなくバニラ Ruby ハッシュになりました。
- 厳密なコンストラクタオプション - バージョン 1 SDK でクライアントまたはリソースオブジェクトを構築する場合、不明なコンストラクタは無視されます。バージョン 2 では、不明なコンストラクタオプションは ArgumentError をトリガーします。(例:

```
# version 1
AWS::S3::Client.new(http_reed_timeout: 10)
# oops, typo'd option is ignored
```

```
# version 2
Aws::S3::Client.new(http_reed_timeout: 10)
# => raises ArgumentError
```

クライアントの違い

バージョン 1 とバージョン 2 の間で、クライアントクラスにはわずかな外部相違点があります。多くのサービスクライアントでは、クライアントの構築後、インターフェイスに互換性があります。重要な相違の一部は以下のとおりです。

- `Aws::S3::Client` - バージョン 1 の Amazon S3 クライアントのクラスはハンドコードでした。バージョン 2 では、サービスモデルから生成されます。メソッドが入力に命名する点がバージョン 2 での大きな相違点です。
- `Aws::EC2::Client`- バージョン 2 では出力リストに複数の名前を使用します。バージョン 1 ではサフィックス `_set` を使用します。(例:

```
# version 1
resp = AWS::EC2::Client.new.describe_security_groups
resp.security_group_set
#=> [...]

# version 2
resp = Aws::EC2::Client.new.describe_security_groups
resp.security_groups
#=> [...]
```

- `Aws::SWF::Client` - バージョン 2 では構造化レスポンスを使用します。バージョン 1 ではバニラ Ruby ハッシュを使用します。
- サービスクラスの名前変更 - バージョン 2 では、複数のサービスに異なる名前を使用します。
  - `AWS::SimpleWorkflow` は `Aws::SWF` になりました
  - `AWS::ELB` は `Aws::ElasticLoadBalancing` になりました
  - `AWS::SimpleEmailService` は `Aws::SES` になりました
- クライアント設定オプション - バージョン 1 の設定オプションの一部がバージョン 2 で名前変更されました。その他は削除または変更されました。以下にプライマリの変更を挙げます。
  - `:use_ssl` は削除されました。バージョン 2 ではどこでも SSL を使用します。SSL を無効にするには、`http://` を使用する `:endpoint` を設定する必要があります。
  - `:ssl_ca_file` => `:ssl_ca_bundle`
  - `:ssl_ca_path` => `:ssl_ca_directory`
  - 「`:ssl_ca_store`」が追加されました。
  - `:endpoint` はホスト名ではなく完全修飾 HTTP または HTTPS uri を使用する必要があります。
  - 各サービスの `:*_port` オプションが削除され、`:endpoint` に置き換えられました。
  - `:user_agent_prefix` は `:user_agent_suffix` になりました

リソースの違い

バージョン 1 とバージョン 2 では、リソースインターフェイスに大きな違いがあります。バージョン 1 では完全にハンドコードでしたが、バージョン 2 ではリソースインターフェイスはモデルから生成されます。バージョン 2 リソースインターフェイスの方が、はるかに整合性がとれています。システム上の相違点の例を次に示します。

- リソースクラスの分離 - バージョン 2 では、サービス名はモジュールであり、クラスではありません。このモジュールがリソースインターフェイスです。

```
# version 1
s3 = AWS::S3.new

# version 2
s3 = Aws::S3::Resource.new
```

- リソースの参照 - バージョン 2 SDK では、リソースゲッタが集合と個別で 2 つの異なるメソッドに分離されています。

```
# version 1
s3.buckets['bucket-name'].objects['key'].delete

# version 2
s3.bucket('bucket-name').object('key').delete
```

- バッチオペレーション - バージョン 1 では、すべてのバッチオペレーションはハンドコードユーティリティでした。バージョン 2 では、多くのバッチオペレーションは API を使用した自動生成バッチオペレーションです。バージョン 2 のバッチインターフェイスは、バージョン 1 と大きく異なります。

Hello World のチュートリアル

このチュートリアルでは、AWS SDK for Ruby を使用して、共通の Amazon S3 オペレーションを実行するコマンドラインプログラムを作成する方法を示します。

プログラムの AWS SDK for Ruby を使用する

Ruby のソースファイルの上部に `require` ステートメントを追加することにより、AWS SDK for Ruby が提供するクラスとメソッドを使用できます。

```
require 'aws-sdk'
```

Amazon S3 リソースの作成

該当するリージョンに Aws::S3::Resource オブジェクトを作成します。次の例では、`us-west-2` リージョンに Amazon S3 リソースオブジェクトを作成します。Amazon S3 リソースは、リージョンに特有ではないので、リージョンは重要ではありません。

```
s3 = Aws::S3::Resource.new(region: 'us-west-2')
```

バケットの作成

Amazon S3 に保存するには、保存するバケットが必要です。

Aws::S3::Bucket オブジェクトを作成する。次の例では、`my-bucket` の名前で、バケット `my_bucket` を作成します。

```
my_bucket = s3.bucket('my-bucket')
my_bucket.create
```

バケットにファイルを追加する

`#upload_file` メソッドを使用して、バケットにファイルを追加します。次の例は、`my_file` という名前のファイルを `my-bucket` という名前のバケットに追加します。

```
name = File.basename 'my_file'
obj = s3.bucket('my-bucket').object(name)
obj.upload_file('my_file')
```

バケットの内容の一覧表示

バケットの内容を一覧表示するには、Aws::S3::Bucket:Objects メソッドを使用します。次の例は、`my-bucket` バケットに対して最大 50 のバケット項目を一覧表示します。

```
my_bucket.objects.limit(50).each do |obj|
  puts "  #{obj.key} => #{obj.etag}"
end
```

プログラム全体

以下は hello-s3.rb 全体のプログラムです。

```
require 'aws-sdk'

NO_SUCH_BUCKET = "The bucket '%s' does not exist!"

USAGE = <<DOC

Usage: hello-s3 bucket_name [operation] [file_name]

Where:
  bucket_name (required) is the name of the bucket

  operation   is the operation to perform on the bucket:
              create  - creates a new bucket
              upload  - uploads a file to the bucket
              list    - (default) lists up to 50 bucket items

  file_name   is the name of the file to upload,
              required when operation is 'upload'

DOC

# Set the name of the bucket on which the operations are performed
# This argument is required
bucket_name = nil

if ARGV.length > 0
  bucket_name = ARGV[0]
else
  puts USAGE
  exit 1
end

# The operation to perform on the bucket
operation = 'list' # default
operation = ARGV[1] if (ARGV.length > 1)

# The file name to use with 'upload'
file = nil
file = ARGV[2] if (ARGV.length > 2)

# Get an Amazon S3 resource
s3 = Aws::S3::Resource.new(region: 'us-west-2')

# Get the bucket by name
bucket = s3.bucket(bucket_name)

case operation
when 'create'
  # Create a bucket if it doesn't already exist
  if bucket.exists?
    puts "The bucket '%s' already exists!" % bucket_name
  else
    bucket.create
```

```
    puts "Created new S3 bucket: %s" % bucket_name
  end

when 'upload'
  if file == nil
    puts "You must enter the name of the file to upload to S3!"
    exit
  end

  if bucket.exists?
    name = File.basename file

    # Check if file is already in bucket
    if bucket.object(name).exists?
      puts "#{name} already exists in the bucket"
    else
      obj = s3.bucket(bucket_name).object(name)
      obj.upload_file(file)
      puts "Uploaded '%s' to S3!" % name
    end
  else
    NO_SUCH_BUCKET % bucket_name
  end

when 'list'
  if bucket.exists?
    # Enumerate the bucket contents and object etags
    puts "Contents of '%s':" % bucket_name
    puts '  Name => GUID'

    bucket.objects.limit(50).each do |obj|
      puts "  #{obj.key} => #{obj.etag}"
    end
  else
    NO_SUCH_BUCKET % bucket_name
  end

else
  puts "Unknown operation: '%s'!" % operation
  puts USAGE
end
```

プログラムの実行

バケットの内容を一覧表示するには、bucket-name がリストするバケット名である次のコマンドのいずれかを使用します。これはデフォルトオペレーションであるため、list を含む必要はありません。

```
ruby hello-s3.rb bucket-name list
ruby hello-s3.rb bucket-name
```

バケットを作成するには、bucket-name が作成するバケットの名前の、次のコマンドを使用します。

```
ruby hello-s3.rb bucket-name create
```

Amazon S3 にすでに bucket-name という名前のバケットがある場合は、サービスはエラーメッセージを発行し、別のコピーは作成されません。

バケットを作成すると、バケットにオブジェクトをアップロードできます。次のコマンドはバケットに your_file.txt を追加します。

```
ruby hello-s3.rb bucket-name upload your_file.txt
```

次のステップ

最初の AWS SDK for Ruby アプリケーションが完了したので、先ほど作成したコードを拡張するための推奨事項をいくつか示します。

- Aws::S3::Resource クラスからの `buckets` コレクションを使用して、バケットのリストを取得します。
- バケット クラスからの `#get` メソッドを使用して、バケットからオブジェクトをダウンロードします。
- 「バケットにファイルを追加する (p. 12)」のコードを使用して、バケットにすでに存在する項目を確認してから、バケット項目を更新します。

AWS SDK for Ruby によるプログラミング

このセクションでは、SDK の高度な機能の一部を使用する方法を含め、AWS SDK for Ruby を使用したソフトウェアの開発に関する情報を提供します。

トピック

デバッグのヒント: クライアントからワイヤトレース情報を取得

`http_wire_trace` オプションを設定して作成するとき、AWS クライアントからワイヤトレース情報を取得できます。この情報により、クライアントの変更、サービスの問題、ユーザーエラーの区別ができます。次の例では、ワイヤトレーシングを有効にして Amazon S3 クライアントを作成します。

```
s3 = Aws::S3::Client.new(http_wire_trace: true)
```

次のコードと引数 `bucket_name` が付与された場合、出力はその名前のバケットが存在するかどうかを示すメッセージを表示します。

```
require 'aws-sdk'

s3 = Aws::S3::Resource.new(client: Aws::S3::Client.new(http_wire_trace: true))

if s3.bucket(ARGV[0]).exists?
  puts "Bucket #{ARGV[0]} exists"
else
  puts "Bucket #{ARGV[0]} does not exist"
end
```

バケットが存在する場合、ACCESS_KEY がアクセス キーの値である場合、出力は次のようになります。(読みやすくするために `HEAD` 行に返却が追加されました。)

```
opening connection to bucket_name.s3-us-west-1.amazonaws.com:443...
opened
starting SSL for bucket_name.s3-us-west-1.amazonaws.com:443...
SSL established
<- "HEAD / HTTP/1.1\r\n
   Content-Type: \r\n
   Accept-Encoding: \r\n
   User-Agent: aws-sdk-ruby2/2.2.7 ruby/2.1.7 x64-mingw32\r\n
```

```
   X-Amz-Date: 20160121T191751Z\r\n
   Host: bucket_name.s3-us-west-1.amazonaws.com\r\n
   X-Amz-Content-Sha256: e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855\r
\n
   Authorization: AWS4-HMAC-SHA256 Credential=ACCESS_KEY/20160121/us-west-1/s3/
aws4_request,
                  SignedHeaders=host;x-amz-content-sha256;x-amz-date,

 Signature=2ca8301c5e829700940d3cc3bca2a3e8d79d177f2c046c34a1a285770db63820\r\n
   Content-Length: 0\r\n
   Accept: */*\r\n
   \r\n"
-> "HTTP/1.1 301 Moved Permanently\r\n"
-> "x-amz-bucket-region: us-west-2\r\n"
-> "x-amz-request-id: F3C75F33EF0792C4\r\n"
-> "x-amz-id-2: N6BzRLx8b68NmF50g1IxLzT
+E4uWPuAIRe7Pl4XKl5STT4tfNO7gBsO8qrrAnG4CbVpU0iIRXmk=\r\n"
-> "Content-Type: application/xml\r\n"
-> "Transfer-Encoding: chunked\r\n"
-> "Date: Thu, 21 Jan 2016 19:17:54 GMT\r\n"
-> "Server: AmazonS3\r\n"
-> "\r\n"
Conn keep-alive
Bucket bucket_name exists
```

クライアントレスポンス、およびエラーをスタブする

AWS SDK for Ruby アプリケーションでクライアントレスポンスとクライアントエラーをスタブする方法を学習します。

クライアントレスポンスをスタブする

レスポンスをスタブすると、AWS SDK for Ruby は、ネットワークトラフィックを無効にし、クライアントはスタブされた (または疑似) データを返します。スタブされたデータを指定しない場合、クライアントは次のように返します。

- リストは空の配列として
- マップは空のハッシュとして
- 数値はゼロで
- 日付は now で

次の例では、Amazon S3 バケットのリストのスタブされた名前を返します。

```
require 'aws-sdk'

s3 = Aws::S3::Client.new(stub_responses: true)

bucket_data = s3.stub_data(:list_buckets, :buckets => [{name:'aws-sdk'}, {name:'aws-
sdk2'}])
s3.stub_responses(:list_buckets, bucket_data)
bucket_names = s3.list_buckets.buckets.map(&:name)

# List each bucket by name
bucket_names.each do |name|
  puts name
```

```
end
```

このコードを実行すると、以下が表示されます。

```
aws-sdk
aws-sdk2
```

> **Note**
>
> スタブされたデータを指定すると、デフォルト値は、残りのインスタンス属性に適用されなくなります。これは、前の例で、残りのインスタンス属性の `creation_date` は、`now` ではなく、`nil` であることを意味します。

AWS SDK for Ruby はスタブされたデータを検証します。間違った種類のデータを渡すと、`ArgumentError` 例外を生成します。たとえば、`bucket_data` への前の割り当ての代わりに、以下を使用したとします。

```
bucket_data = s3.stub_data(:list_buckets, buckets:['aws-sdk', 'aws-sdk2'])
```

AWS SDK for Ruby は、2 つの `ArgumentError` 例外を生成します。

```
expected params[:buckets][0] to be a hash
expected params[:buckets][1] to be a hash
```

クライアントエラーをスタブする

AWS SDK for Ruby が特定のメソッドに対して生成したエラーをスタブすることができます。次の例では `Caught Timeout::Error error calling head_bucket on aws-sdk` が表示されます。

```
require 'aws-sdk'

s3 = Aws::S3::Client.new(stub_responses: true)
s3.stub_responses(:head_bucket, Timeout::Error)

begin
  s3.head_bucket({bucket: 'aws-sdk'})
rescue Exception => ex
  puts "Caught #{ex.class} error calling 'head_bucket' on 'aws-sdk'"
end
```

レスポンスデータのページング

一部の AWS 呼び出しはページ分割レスポンスを提供して、各レスポンスに返されるデータの量を制限します。1 ページのデータは、最大 1,000 個の項目を表します。

ページ分割レスポンスは Enumerable です

ページ分割レスポンスデータを処理する最も簡単な方法は、次の例のように、レスポンスオブジェクト内で組み込み列挙子を使用することです。

```
s3 = Aws::S3::Client.new

s3.list_objects(bucket:'aws-sdk').each do |response|
```

```
  puts response.contents.map(&:key)
end
```

これにより API コールごとに 1 つの応答オブジェクトが得られ、名前を持つバケットのオブジェクトを列挙されます。SDK はリクエストを完了するために追加のページを取得します。

ページ分割レスポンスを手動で処理

ページングをユーザー自身が処理するには、レスポンスの `next_page?` メソッドを使用して、取得するページがまだあることを確認するか、`last_page?` メソッドを使用して、取得するページがこれ以上ないことを確認します。

ページがまだある場合は、次の例のように、`next_page` を使用して、(`?` がないことを確認します) 次のページの結果を取得します。

```
s3 = Aws::S3::Client.new

# Get the first page of data
response = s3.list_objects(bucket:'aws-sdk')

# Get additional pages
while response.next_page? do
  response = response.next_page
  # Use the response data here...
end
```

> **Note**
>
> `next_page` メソッドを呼び出して、取得するページがない場合は、SDK は Aws::PageableResponse::LastPageError 例外を生成します。

ページ分割データクラス

AWS SDK for Ruby のページ分割データは Aws::PageableResponse クラスによって処理され、Seahorse::Client::Response に含まれて、ページ分割データへのアクセスを提供します。

ウェーターを使用する

ウェーターは、クライアントに発生する特定の状態をポーリングするユーティリティメソッドです。ウェーターはサービスクライアントに対して定義されたポーリング間隔で何度か試行した後に失敗する場合があります。ウェーターを使用する方法の例については、「シンプルなプライマリキーを使用してシンプルなテーブルを作成する (p. 30)」を参照してください。

ウェーターの呼び出し

ウェーターを呼び出すには、クライアントサービスで `#wait_until` を呼び出します。次の例では、ウェーターは、続行する前にインスタンス `i-12345678` が実行するまで待機します。

```
ec2 = Aws::EC2::Client.new

begin
  ec2.wait_until(:instance_running, instance_ids:['i-12345678'])
  puts "instance running"
rescue Aws::Waiters::Errors::WaiterFailed => error
  puts "failed waiting for instance running: #{error.message}"
```

```
end
```

最初のパラメータがウェーターの名前で、サービスクライアントに対して固有であり、どのオペレーションを待機しているかを示します。2 番目のパラメータは、ウエーターに呼び出されたクライアントメソッドに渡されるパラメータのハッシュで、ウェーター名に応じて異なります。

待機できるオペレーションの一覧および各オペレーションに必要なクライアントメソッドについては、「#wait_until」および、使用しているクライアント用の「#waiter_names」フィールドドキュメントを参照してください。

待機の失敗

ウェーターは、次の例外のいずれかで失敗する場合があります。

Aws::Waiters::Errors::FailureStateError

待機中にエラー状態が発生しました。

Aws::Waiters::Errors::NoSuchWaiterError

指定されたウェーター名は使用しているクライアントに対して定義されていません。

Aws::Waiters::Errors::TooManyAttemptsError

試行数がウェーターの max_attempts 値を超えています。

Aws::Waiters::Errors::UnexpectedError

予期しないエラーが待機中に発生しました。

Aws::Waiters::Errors::WaiterFailed

待機中に待機状態の 1 つが超過、または別のエラーが発生しました。

これらすべてのエラー - NoSuchWaiterError を除く - は、WaiterFailed に基づいています。ウェーターのエラーを見つけるには、次の例に示すように、WaiterFailed を使用します。

```
rescue Aws::Waiters::Errors::WaiterFailed => error
  puts "failed waiting for instance running: #{error.message}"
end
```

ウェーターの設定

各ウェーターにはデフォルトのポーリング間隔とプログラムへコントロールを返す前の試行回数の上限があります。これらの値を設定するには、max_attempts および delay: パラメータを #wait_until 呼び出しで使用します。次の例は、最大 25 秒待機、5 秒間隔でポーリングします。

```
# Poll for ~25 seconds
client.wait_until(...) do |w|
  w.max_attempts = 5
  w.delay = 5
end
```

待機の失敗を無効にするには、これらのパラメータのいずれかの値を nil に設定します。

ウェーターの拡張

ウェーターの動作を変更するには、各ポーリング試行の前および待機の前にトリガーされたコールバックを登録できます。

次の例は、各試行の待機時間を 2 倍にすることにより、ウェーターのエクスポネンシャルバックオフを実装します。

```
ec2 = Aws::EC2::Client.new

ec2.wait_until(:instance_running, instance_ids:['i-12345678']) do |w|
  w.interval = 0 # disable normal sleep
  w.before_wait do |n, resp|
    sleep(n ** 2)
  end
end
```

次の例は試行最大数を無効にし、代わりに、エラーが発生する前に 1 時間 (3600 秒) 待機します。

```
started_at = Time.now
client.wait_until(...) do |w|
  # Disable max attempts
  w.max_attempts = nil

  # Poll for 1 hour, instead of a number of attempts
  before_wait do |attempts, response|
    throw :failure if Time.now - started_at > 3600
  end
end
```

クライアントタイムアウト期間を指定

次の例では、`us-west-2` リージョンの Amazon S3 クライアントを作成し、各クライアントオペレーションで 2 つの再試行間に 5 秒待機するように指定します。デフォルトでは、SDK は再試行の間に 15 秒あけて、最大 3 回の再試行を試み、合計最大 4 回の試行を実行します。したがって、オペレーションはタイムアウトするのに 60 秒かかることがあります。例では、タイムアウトに 15 秒かかる場合があります。

```
s3 = Aws::S3::Client.new(region: 'us-west-2', retry_limit: 2, http_open_timeout: 5)
```

AWS SDK for Ruby コード例

このセクションでは、AWS SDK for Ruby を使用して、AWS のサービスにアクセスするために使用できる例を提供します。

トピック

Amazon CloudWatch の例

AWS SDK for Ruby を使用して Amazon CloudWatch (CloudWatch) にアクセスするには、次の例を使用できます。CloudWatch の詳細については、CloudWatch 開発者ガイドを参照してください。

例

トピック

すべてのアラームに関する情報の取得

次の例では、CloudWatch アラームについての情報を示します。

```
require 'aws-sdk'

client = Aws::CloudWatch::Client.new(region: 'us-west-2')

# use  client.describe_alarms({alarm_names: ['Name1', 'Name2']})
# to get information about alarms Name1 and Name2
resp = client.describe_alarms
```

```
resp.metric_alarms.each do |alarm|
  puts 'Name:           ' + alarm.alarm_name
  puts 'State:          ' + alarm.state_value
  puts '  reason:       ' + alarm.state_reason
  puts 'Metric:         ' + alarm.metric_name
  puts 'Namespace:      ' + alarm.namespace
  puts 'Statistic:      ' + alarm.statistic
  puts 'Dimensions (' + alarm.dimensions.length.to_s + '):'

  alarm.dimensions.each do |d|
    puts '  Name:         ' + d.name
    puts '  Value:        ' + d.value
  end

  puts 'Period:         ' + alarm.period.to_s
  puts 'Unit:           ' + alarm.unit.to_s
  puts 'Eval periods:   ' + alarm.evaluation_periods.to_s
  puts 'Threshold:      ' + alarm.threshold.to_s
  puts 'Comp operator:  ' + alarm.comparison_operator
  puts
end
```

アラームを作成する

次の例では、Amazon S3 バケット my-bucket に 24 時間あたり 50 以上の項目が存在すると、ARN `ARN` で Amazon SNS トピックを使用してメッセージを送信する CloudWatch アラーム my-alarm を作成します。

```
require 'aws-sdk'

# Placeholder for put_metric_alarm args
args = {}
args[:alarm_name] = 'my-alarm'
args[:alarm_description] = 'Triggers alarm when S3 bucket my-bucket has more than 50 items'
args[:alarm_actions] = 'ARN'
args[:namespace] = 'AWS/S3'
args[:metric_name] = 'NumberOfObjects'

dim1 = {}
dim1[:name] = 'BucketName'
dim1[:value] = 'my-bucket'

dim2 = {}
dim2[:name] = 'StorageType'
dim2[:value] = 'AllStorageTypes'

dimensions = []

dimensions << dim1
dimensions << dim2

args[:dimensions] = dimensions

args[:statistic] = 'Maximum'

# NumberOfObjects REQUIRES this value
args[:period] = 86400

# NumberOfObjects REQUIRES this value
args[:unit] = nil

args[:evaluation_periods] = 1
args[:threshold] = 50
args[:comparison_operator] = 'GreaterThanThreshold'
```

```
cw = Aws::CloudWatch::Client.new(region: 'us-west-2')

cw.put_metric_alarm(args)
```

アラームアクションの有効化と無効化

Amazon CloudWatch アラームで、指定した期間中、1 つのメトリクスを監視します。CloudWatch アラームは、指定された複数の期間にわたるしきい値とメトリクスの値の関係性に基づき、1 つ以上のアクションを実行します。詳細については、「Amazon CloudWatch アラームの作成」を参照してください。

この例では、AWS SDK for Ruby を CloudWatch で使用して、以下のことを行います。

1. Aws::CloudWatch::Client#put_metric_alarm を使用して CloudWatch アラームのアクションを有効にする。
2. Aws::CloudWatch::Client#disable_alarm_actions を使用してアラームのすべてのアクションを無効にする。

この例の完全なコードは GitHub で入手できます。

前提条件

コード例を実行する前に、以下で説明されているように、AWS SDK for Ruby をインストールし、設定する必要があります。

- AWS SDK for Ruby のインストール (p. 4)
- AWS SDK for Ruby の設定 (p. 5)

また、arn:aws:sns:REGION-ID:ACCOUNT-ID:TOPIC-NAME を、有効な Amazon SNS トピックの ARN で置き換える必要があります。

例

```
require 'aws-sdk'

# Uncomment for Windows.
# Aws.use_bundled_cert!

cw = Aws::CloudWatch::Client.new(region: 'us-east-1')

# Enable an action for an Amazon CloudWatch alarm.
# If the alarm does not exist, create it.
# If the alarm exists, update its settings.
alarm_name = "TooManyObjectsInBucket"

cw.put_metric_alarm({
  alarm_name: alarm_name,
  alarm_description: "Alarm whenever an average of more than one object exists in the
 specified Amazon S3 bucket for more than one day.",
  actions_enabled: true, # Run actions if the alarm's state changes.
  metric_name: "NumberOfObjects",
  alarm_actions: [ "arn:aws:sns:REGION-ID:ACCOUNT-ID:TOPIC-NAME" ], # Notify this Amazon
 SNS topic only if the alarm's state changes to ALARM.
  namespace: "AWS/S3",
  statistic: "Average",
  dimensions: [
    {
```

```
      name: "BucketName",
      value: "my-bucket"
    },
    {
      name: "StorageType",
      value: "AllStorageTypes"
    }
  ],
  period: 86400, # Daily (24 hours * 60 minutes * 60 seconds = 86400 seconds).
  unit: "Count",
  evaluation_periods: 1, # More than one day.
  threshold: 1, # One object.
  comparison_operator: "GreaterThanThreshold" # More than one object.
})

# Disable all actions for the alarm.
cw.disable_alarm_actions({
  alarm_names: [ alarm_name ]
})
```

カスタムメトリクスに関する情報の取得

CloudWatch アラームは、指定した期間中、1 つのメトリクスを監視します。CloudWatch アラームは、指定された複数の期間にわたるしきい値とメトリクスの値の関係性に基づき、1 つ以上のアクションを実行します。詳細については、「Amazon CloudWatch アラームの作成」を参照してください。

この例では、AWS SDK for Ruby を CloudWatch で使用して、以下のことを行います。

1. Aws::CloudWatch::Client#put_metric_data を使用してカスタムメトリクスを CloudWatch に送信する。
2. Aws::CloudWatch::Client#list_metrics-instance を使用してカスタムメトリクスに関する情報を取得する。

前提条件

コード例を実行する前に、以下で説明されているように、AWS SDK for Ruby をインストールし、設定する必要があります。

- AWS SDK for Ruby のインストール (p. 4)
- AWS SDK for Ruby の設定 (p. 5)

例

```
require 'aws-sdk'

# Uncomment for Windows.
# Aws.use_bundled_cert!

cw = Aws::CloudWatch::Client.new(region: 'us-east-1')

# Send custom metrics to Amazon CloudWatch.
# In this example, add metrics to the custom namespace "SITE/TRAFFIC":
# For the custom dimension named "SiteName", for the value named "example.com", add
# "UniqueVisitors" of 5885 and "UniqueVisits" of 8628.
# For the custom dimension named "PageURL", for the value named "my-page.html", add
# "PageViews" of 18057.
cw.put_metric_data({
  namespace: "SITE/TRAFFIC",
```

```
  metric_data: [
    {
      metric_name: "UniqueVisitors",
      dimensions: [
        {
          name: "SiteName",
          value: "example.com"
        }
      ],
      value: 5885.0,
      unit: "Count"
    },
    {
      metric_name: "UniqueVisits",
      dimensions: [
        {
          name: "SiteName",
          value: "example.com"
        }
      ],
      value: 8628.0,
      unit: "Count"
    },
    {
      metric_name: "PageViews",
      dimensions: [
        {
          name: "PageURL",
          value: "my-page.html"
        }
      ],
      value: 18057.0,
      unit: "Count"
    }
  ]
})

# Get information about custom metrics.
list_metrics_output = cw.list_metrics({
  namespace: "SITE/TRAFFIC"
})

list_metrics_output.metrics.each do |metric|
  puts metric.metric_name
  metric.dimensions.each do |dimension|
    puts "#{dimension.name} = #{dimension.value}"
  end
  puts "\n"
end
```

Amazon CloudWatch Events にイベントを送信する

CloudWatch イベント は、AWS リソースでの変更を示すシステムイベントのほぼリアルタイムのストリームを、AWS Lambda 関数またはその他のターゲットに配信します。詳細については、「Amazon CloudWatch イベントとは」を参照してください。この例では、AWS SDK for Ruby を CloudWatch イベント で使用して、以下のことを行います。

1. Aws::CloudWatchEvents::Client#put_rule を使用して CloudWatch イベント でルールを作成する。
2. Aws::CloudWatchEvents::Client#put_targets を使用してルールにターゲットを追加する。
3. イベントを CloudWatch イベント に送信し、ルールに一致するようにする。
4. Aws::CloudWatch::Client#get_metric_statistics と Aws::CloudWatchLogs::Client#describe_log_streams を使用して、Amazon CloudWatch Logs とメトリクスで結果を表示する。

前提条件

コード例を実行する前に、以下で説明されているように、AWS SDK for Ruby をインストールし、設定する必要があります。

- AWS SDK for Ruby のインストール (p. 4)
- AWS SDK for Ruby の設定 (p. 5)

また、次の操作も行う必要があります。

- lambda_function_arn に割り当てられたプレースホルダーの値を、実際の Lambda 関数の ARN に置き換える。
  1. ここで説明されているように Lambda 関数を作成する。
  2. 関数 LogEC2InstanceStateChange に名前を付ける。
  3. ロールに対して、[Choose an Existing Role] を選択する。既存のロールの場合は、lambda_basic_execution を選択する。
  4. 関数を作成した後、ARN をコピーし、コードに貼り付ける。
- cwe_service_role_arn に割り当てられたプレースホルダーの値を、適切な AWS IAM サービスロールの ARN で置き換える。
  1. IAM コンソールで、ロールを作成し、CloudWatch イベント へのフルアクセス権を付与するポリシーをアタッチする。
  2. ロールに信頼関係 events.amazonaws.com があることを確認する。ポリシーとロールの例については、GitHub のサンプルコード内のコメントを参照してください。
  3. ロールの作成が完了したら、ポリシーをアタッチし、信頼関係を確立する。次に、ロールの ARN をコピーしてコードに貼り付ける。
- instance_id に割り当てられたプレースホルダーの値を、実際の Amazon EC2 インスタンス ID と置き換える。

例

```
require 'aws-sdk'

# Uncomment for Windows.
# Aws.use_bundled_cert!

cwe = Aws::CloudWatchEvents::Client.new(region: 'us-east-1')

# Replace this value with the ARN of the AWS Lambda function you created earlier.
lambda_function_arn = "arn:aws:lambda:REGION-ID:ACCOUNT-
ID:function:LogEC2InstanceStateChange"

# Replace this value with the ARN of the AWS IAM service role you created earlier.
cwe_service_role_arn = "arn:aws:iam::ACCOUNT-ID:role/SERVICE-ROLE-NAME"

# Create a rule in Amazon CloudWatch Events.
rule_name = "my-ec2-rule"

# The rule will use this pattern to route the event to the target.
# This pattern is used whenever an Amazon EC2 instance begins running.
event_pattern = {
  "source" => [
    "aws.ec2"
  ],
  "detail-type" => [
    "EC2 Instance State-change Notification"
```

```
  ],
  "detail" => {
    "state" => [
      "running"
    ]
  }
}.to_json

cwe.put_rule({
  name: rule_name,
  event_pattern: event_pattern,
  state: "ENABLED",
  role_arn: cwe_service_role_arn
})

# Add a target to the rule.
cwe.put_targets({
  rule: rule_name,
  targets: [
    {
      id: "my-rule-target",
      arn: lambda_function_arn
    }
  ]
})

# To test the rule, stop and then restart an existing Amazon EC2 instance.
# For example:
ec2 = Aws::EC2::Client.new(region: 'us-east-1')

# Replace this with an actual instance ID.
instance_id = "i-INSTANCE-ID"

puts "Attempting to stop the instance. This may take a few minutes..."

ec2.stop_instances({
  instance_ids: [ instance_id ]
})

# Make sure the instance is stopped before attempting to restart it.
ec2.wait_until(:instance_stopped, instance_ids: [ instance_id ])

puts "Attempt to restart the instance. This may take a few minutes..."

ec2.start_instances({
  instance_ids: [ instance_id ]
})

# Make sure the instance is running before continuing on.
ec2.wait_until(:instance_running, instance_ids: [ instance_id ])

# See if and when the rule was triggered.
cw = Aws::CloudWatch::Client.new(region: 'us-east-1')

invocations = cw.get_metric_statistics({
  namespace: "AWS/Events",
  metric_name: "Invocations",
  dimensions: [
    {
      name: "RuleName",
      value: rule_name,
    },
  ],
  start_time: Time.now - 600, # Look back over the past 10 minutes to see if the rule was
 triggered (10 minutes * 60 seconds = 600 seconds).
  end_time: Time.now,
```

```
  period: 60, # Look back every 60 seconds over those past 10 minutes to see how many times
 the rule may have been triggered.
  statistics: [ "Sum" ],
  unit: "Count"
})

if invocations.datapoints.count > 0
  puts "Rule invocations:"
  invocations.datapoints.each do |datapoint|
    puts "  #{datapoint.sum} invocation(s) at #{datapoint.timestamp}"
  end
else
  puts "No rule invocations."
end

# View the latest related log in Amazon CloudWatch Logs.
cwl = Aws::CloudWatchLogs::Client.new(region: 'us-east-1')

describe_log_streams_response = cwl.describe_log_streams({
  log_group_name: "/aws/lambda/LogEC2InstanceStateChange",
  order_by: "LastEventTime",
  descending: true
})

get_log_events_response = cwl.get_log_events({
  log_group_name: "/aws/lambda/LogEC2InstanceStateChange",
  log_stream_name: describe_log_streams_response.log_streams[0].log_stream_name # Get the
 latest log stream only.
})

puts "\nLog messages:\n\n"

get_log_events_response.events.each do |event|
  puts event.message
end
```

Amazon DynamoDB の例

AWS SDK for Ruby を使用して Amazon DynamoDB サービスにアクセスするには、次の例を使用できます。DynamoDB に関する詳細については、Amazon DynamoDB ドキュメントを参照してください。具体的には、「Ruby および DynamoDB」を参照して、以下の方法を学習します。

- テーブルを作成し、JSON 形式のサンプルデータをロードします。
- テーブルで、作成、読み込み、更新、削除のオペレーションを実行します。
- 簡単なクエリを実行します。

このトピックでは、インタラクティブウェブインターフェイスを含む DynamoDB のダウンロード可能バージョンへのリンクも提供しており、DynamoDB をオフラインで試すことができます。

例

トピック

- テーブルの更新 (p. 32)
- インデックスの作成 (p. 32)

全テーブルに関する情報の取得

次の例では、us-west-2 リージョンの各テーブル内の項目の名前と数が一覧表示されます。

```
require 'aws-sdk'

dynamoDB = Aws::DynamoDB::Resource.new(region: 'us-west-2')

dynamoDB.tables.each do |t|
  puts "Name:    #{t.name}"
  puts "#Items:  #{t.item_count}"
end
```

1 つのプライマリキーを使用して単純なテーブルを作成する

次の例では、us-west-2 リージョンの ID、FirstName、LastName の 3 つの属性を使用して、テーブル Users を作成します。

wait_until 呼び出しは、DynamoDB によって作成されるまでは、テーブルの使用をブロックします。デフォルトでは、DynamoDB クライアントの wait_until メソッドは、テーブルが作成されたかどうかを調べるために 20 秒ごとに (最大 500 秒) 確認します。

```
require 'aws-sdk'

attribute_defs = [
  { attribute_name: 'ID',        attribute_type: 'N' },
  { attribute_name: 'FirstName', attribute_type: 'S' },
  { attribute_name: 'LastName',  attribute_type: 'S' }
]

key_schema = [
  { attribute_name: 'ID', key_type: 'HASH' }
]

index_schema = [
  { attribute_name: 'FirstName', key_type: 'HASH'  },
  { attribute_name: 'LastName',  key_type: 'RANGE' }
]

global_indexes = [{
  index_name:             'LastNameFirstNameIndex',
  key_schema:             index_schema,
  projection:             { projection_type: 'ALL' },
  provisioned_throughput: { read_capacity_units: 5, write_capacity_units: 10 }
}]

request = {
  attribute_definitions:    attribute_defs,
  table_name:               'Users',
  key_schema:               key_schema,
  global_secondary_indexes: global_indexes,
  provisioned_throughput:   { read_capacity_units: 5, write_capacity_units: 10 }
}
```

```
dynamodb_client = Aws::DynamoDB::Client.new(region: 'us-west-2')

dynamodb_client.create_table(request)
dynamodb_client.wait_until(:table_exists, table_name: 'Users')
```

テーブルに項目を追加する

次の例では us-west-2 リージョン内の Users テーブルに 123456 の値の ID、John の値の FirstName、Doe の値の LastName を使用した項目を追加します。

```
require 'aws-sdk'

dynamoDB = Aws::DynamoDB::Resource.new(region: 'us-west-2')

table = dynamoDB.table('Users')

table.put_item({
  item:
    {
      "ID" => 123456,
      "FirstName" => 'Snoop',
      "LastName" => 'Doug'
}})
```

テーブル内の項目に関する情報の取得

次の例では、us-west-2 リージョンの Users テーブルから最大 50 項目を一覧表示します。

```
require 'aws-sdk'

dynamoDB = Aws::DynamoDB::Resource.new(region: 'us-west-2')

table = dynamoDB.table('Users')

scan_output = table.scan({
  limit: 50,
  select: "ALL_ATTRIBUTES"
})

scan_output.items.each do |item|
  keys = item.keys

  keys.each do |k|
    puts "#{k}: #{item[k]}"
  end
end
```

テーブル内の特定の項目に関する情報の取得

次の例は us-west-2 リージョン内の Users テーブルの 123456 の ID を使用して、項目の最初と最後の名前を表示します。

```
require 'aws-sdk'

dynamoDB = Aws::DynamoDB::Resource.new(region: 'us-west-2')

table = dynamoDB.table('Users')
```

```
resp = table.get_item({
  key: { 'ID' => 123456 }
})

first_name = resp.item['FirstName']
last_name = resp.item['LastName']

puts "First name: #{first_name}"
puts "Last name:  #{last_name}"
```

テーブルの更新

次の例では、新しいフィールド airmiles が含まれるように us-west-2 リージョン内の Users テーブルで、すべての項目を更新し、値を 10000 に設定します。

```
require 'aws-sdk'

dynamoDB = Aws::DynamoDB::Resource.new(region: 'us-west-2')

table = dynamoDB.table('Users')

# Get the IDs of all of the users
resp = table.scan({ select: "ALL_ATTRIBUTES" })

resp.items.each do |item|
  id = item['ID']

  request = {
    key: { 'ID' => id },
    update_expression: 'set airmiles=:pVal',
    expression_attribute_values: { ':pVal' => '10000' }
  }

  # Update the item in the table:
  table.update_item(request)
end
```

インデックスの作成

次の例は us-west-2 リージョン内の Users テーブルに、新しいインデックス air-mileage-index を追加します。インデックスのステータスが、ACTIVE になったら、airmiles の値に基づいて、テーブルの項目を検索できます。

```
require 'aws-sdk'

request = {
  attribute_definitions: [
    {
      attribute_name: 'airmiles',
      attribute_type: 'N',
    },
  ],
  table_name: 'Users',
  global_secondary_index_updates: [
    {
      create: {
        index_name: 'air-mileage-index',
        key_schema: [
          {
            attribute_name: 'airmiles',
```

```
            key_type: 'HASH',
          },
        ],
        projection: {
          projection_type: 'ALL',
        },
        provisioned_throughput: {
          read_capacity_units: 5,
          write_capacity_units: 10,
        },
      },
    },
  ],
}

dynamoDB = Aws::DynamoDB::Client.new(region: 'us-west-2')

dynamoDB.update_table(request)
```

Amazon EC2 例

AWS SDK for Ruby を使用して Amazon Elastic Compute Cloud (Amazon EC2) にアクセスするには、次の例を使用できます。Amazon EC2 に関する詳細は、Amazon EC2 ドキュメントを参照してください。

例

トピック

VPC を作成する

次の例では、CIDR ブロック `10.200.0.0/16` を持つ Virtual Private Cloud (VPC) `MyGroovyVPC` を作成し、VPC の ID を表示します。

例では 65,536 個のプライベート IP アドレスで仮想ネットワークを作成します。

```
require 'aws-sdk'

ec2 = Aws::EC2::Resource.new(region: 'us-west-2')

vpc = ec2.create_vpc({ cidr_block: '10.200.0.0/16' })

# So we get a public DNS
vpc.modify_attribute({
  enable_dns_support: { value: true }
})

vpc.modify_attribute({
  enable_dns_hostnames: { value: true }
})

# Name our VPC
vpc.create_tags({ tags: [{ key: 'Name', value: 'MyGroovyVPC' }]})

puts vpc.vpc_id
```

インターネットゲートウェイを作成して VPC にアタッチする

次の例では、インターネットゲートウェイ `MyGroovyIGW` を作成し、ID `VPC_ID` のある VPC にアタッチして、インターネットゲートウェイの ID を表示します。

```
require 'aws-sdk'

ec2 = Aws::EC2::Resource.new(region: 'us-west-2')

igw = ec2.create_internet_gateway

igw.create_tags({ tags: [{ key: 'Name', value: 'MyGroovyIGW' }]})
igw.attach_to_vpc(vpc_id: VPC_ID)

puts igw.id
```

パブリックサブネットを作成する

次の例では、`us-west-2` リージョンとアベイラビリティーゾーン `us-west-2a` でパブリックサブネット `MyGroovySubnet` を作成します。例では、CIDR ブロック `10.200.10.0/24` を使用する ID `VPC_ID` を持つ VPC をパブリックサブネットにアタッチして、サブネットの ID を表示します。

この例で作成したパブリックサブネットには、VPC 内に 256 個のプライベート IP アドレスがあります。

```
require 'aws-sdk'

ec2 = Aws::EC2::Resource.new(region: 'us-west-2')

subnet = ec2.create_subnet({
  vpc_id: VPC_ID,
  cidr_block: '10.200.10.0/24',
  availability_zone: 'us-west-2a'
})

subnet.create_tags({ tags: [{ key: 'Name', value: 'MyGroovySubnet' }]})
```

```
puts subnet.id
```

ルートテーブルを作成しサブネットに関連付ける

次の例では、ID `VPC_ID` の VPC で `us-west-2` リージョンでルートテーブル `MyGroovyRouteTable` を作成します。ルートテーブルは CIDR ブロック `0.0.0.0/0` を持つルートと ID `IGW_ID` のゲートウェイを使用します。例では、ルートテーブルを ID `SUBNET_ID` を持つサブネットと関連付け、ルートテーブルの ID を表示します。

```
require 'aws-sdk'

ec2 = Aws::EC2::Resource.new(region: 'us-west-2')

table = ec2.create_route_table({
    vpc_id: VPC_ID
  })

table.create_tags({ tags: [{ key: 'Name', value: 'MyGroovyRouteTable' }]})

table.create_route({
  destination_cidr_block: '0.0.0.0/0',
  gateway_id: IGW_ID
})

table.associate_with_subnet({
  subnet_id: SUBNET_ID
})

puts table.id
```

Amazon EC2 の Elastic IP アドレスの使用

Elastic IP アドレスは、AWS アカウントに関連付けられた、動的なクラウドコンピューティングのために設計された静的 IP アドレスです。これは、インターネットから到達可能なパブリック IP アドレスです。インスタンスにパブリック IP アドレスがない場合、Elastic IP アドレスをインスタンスで使用して、インターネットと通信するようにできます。

Amazon EC2 の Elastic IP アドレスの詳細については、Amazon EC2 User Guide for Linux Instancesの「Elastic IP アドレス」または Amazon EC2 User Guide for Windows Instancesの「Elastic IP アドレス」を参照してください。

この例では、AWS SDK for Ruby を Amazon EC2 で使用して、以下のことを行います。

1. Aws::EC2::Client#allocate_address メソッドを使用して Elastic IP アドレスを割り当てる。
2. Aws::EC2::Client#associate_address メソッドを使用して、Amazon EC2 インスタンスにアドレスを関連付ける。
3. Aws::EC2::Client#describe_addresses メソッドを使用して、インスタンスに関連付けられているアドレスに関する情報を取得する。
4. Aws::EC2::Client#release_address メソッドを使用してアドレスを解放する。

この例の完全なコードは GitHub で入手できます。

前提条件

コード例を操作する前に、以下で説明されているように、AWS SDK for Ruby をインストールし、設定する必要があります。

- AWS SDK for Ruby のインストール (p. 4)
- AWS SDK for Ruby の設定 (p. 5)

また、EC2 インスタンスを起動し、インスタンス ID をメモしておく必要があります。

Note

次のコードを実行する前に、`INSTANCE-ID` 文字列を実際のインスタンス ID に置き換える必要があります。これは `i-0a123456b7c8defg9` のようになります。

スクリプト例

```
require 'aws-sdk'

ec2 = Aws::EC2::Client.new(region: 'us-east-1')

instance_id = "INSTANCE-ID" # For example, "i-0a123456b7c8defg9"

def display_addresses(ec2, instance_id)
  describe_addresses_result = ec2.describe_addresses({
    filters: [
      {
        name: "instance-id",
        values: [ instance_id ]
      },
    ]
  })
  if describe_addresses_result.addresses.count == 0
    puts "No addresses currently associated with the instance."
  else
    describe_addresses_result.addresses.each do |address|
      puts "=" * 10
      puts "Allocation ID: #{address.allocation_id}"
      puts "Association ID: #{address.association_id}"
      puts "Instance ID: #{address.instance_id}"
      puts "Public IP: #{address.public_ip}"
      puts "Private IP Address: #{address.private_ip_address}"
    end
  end
end

puts "Before allocating the address for the instance...."
display_addresses(ec2, instance_id)

puts "\nAllocating the address for the instance..."
allocate_address_result = ec2.allocate_address({
  domain: "vpc"
})

puts "\nAfter allocating the address for instance, but before associating the address with
 the instance..."
display_addresses(ec2, instance_id)

puts "\nAssociating the address with the instance..."
associate_address_result = ec2.associate_address({
  allocation_id: allocate_address_result.allocation_id,
  instance_id: instance_id,
})

puts "\nAfter associating the address with the instance, but before releasing the address
 from the instance..."
display_addresses(ec2, instance_id)
```

```
puts "\nReleasing the address from the instance..."
ec2.release_address({
  allocation_id: allocate_address_result.allocation_id,
})

puts "\nAfter releasing the address from the instance..."
display_addresses(ec2, instance_id)
```

セキュリティグループを作成する

次の例では、ID `VPC_ID` を持つ VPC の `us-west-2` リージョンのセキュリティグループ `MyGroovySecurityGroup` を作成します。例では、セキュリティグループは、すべてのアドレス (CIDR ブロック `0.0.0.0/0`) からポート 22 (SSH) 経由でのアクセスを許可され、「MyGroovyInstance のセキュリティグループ」と説明されています。次に、セキュリティグループの ID が表示されます。

```
require 'aws-sdk'

ec2 = Aws::EC2::Resource.new(region: 'us-west-2')

sg = ec2.create_security_group({
    group_name: 'MyGroovySecurityGroup',
    description: 'Security group for MyGroovyInstance',
    vpc_id: VPC_ID
  })

sg.authorize_egress({
  ip_permissions: [{
    ip_protocol: 'tcp',
    from_port: 22,
    to_port: 22,
    ip_ranges: [{
      cidr_ip: '0.0.0.0/0'
  }]
})

puts sg.id
```

Amazon EC2 のセキュリティグループでの作業

Amazon EC2 セキュリティグループは、1 つ以上のインスタンスのトラフィックを制御する仮想ファイアウォールとして機能します。各セキュリティグループに対してルールを追加し、関連付けられたインスタンスに対するトラフィックを許可します。セキュリティグループルールはいつでも変更できます。新しいルールは、セキュリティグループに関連付けられているインスタンスすべてに自動的に適用されます。

Amazon EC2 セキュリティグループの詳細については、以下を参照してください。

- Linux インスタンス用の Amazon EC2 Amazon セキュリティグループ
- Windows インスタンス用の Amazon EC2 Amazon セキュリティグループ

この例では、AWS SDK for Ruby を Amazon EC2 で使用して、以下のことを行います。

1. セキュリティグループを作成する。
2. セキュリティグループにルールを追加する。
3. セキュリティグループに関する情報を取得する。
4. セキュリティグループを削除する。

以下のすべての例を含む完全なサンプルスクリプトは、GitHub で入手できます。

前提条件

以下のコードを使用する前に、AWS SDK for Ruby をインストールし、設定する必要があります。以下を参照してください。

- AWS SDK for Ruby のインストール (p. 4)
- AWS SDK for Ruby の設定 (p. 5)

また、VPC を作成し、VPC ID を書き留めておく必要があります。

SDK を設定する

まず、AWS SDK for Ruby が必要で、EC2 クライアントを作成します。次に、作成するセキュリティグループの名前を付けます。また、VPC の ID を指定する必要があります。これは VPC の作成後にコンソールで確認できます。必ず、「VPC-ID」を実際の VPC ID に置き換えてください。

```
require 'aws-sdk'

ec2 = Aws::EC2::Client.new(region: 'us-east-1')

security_group_name = "my-security-group"
vpc_id = "VPC-ID" # For example, "vpc-1234ab56".
security_group_created = false # Used later to determine whether it's okay to delete the
 security group.
```

後で、`security_group_created` 変数をスクリプトで使用して、セキュリティグループが作成されたかどうか (したがって、削除できるかどうか) 確認します。

セキュリティグループの作成

次に、すべてのアドレス (CIDR ブロック `0.0.0.0/0`) からポート 22 (SSH) および 80 (HTTP) 経由でアクセスを許可するセキュリティグループを作成します。

```
# Create a security group.
begin
  create_security_group_result = ec2.create_security_group({
    group_name: security_group_name,
    description: "An example description for my security group.",
    vpc_id: vpc_id
  })

  # Add rules to the security group.
  # For example, allow all inbound HTTP and SSH traffic.
  ec2.authorize_security_group_ingress({
    group_id: create_security_group_result.group_id,
    ip_permissions: [
      {
        ip_protocol: "tcp",
        from_port: 80,
        to_port: 80,
        ip_ranges: [
          {
            cidr_ip: "0.0.0.0/0",
          }
        ]
      },
      {
        ip_protocol: "tcp",
        from_port: 22,
```

```
        to_port: 22,
        ip_ranges: [
          {
            cidr_ip: "0.0.0.0/0",
          }
        ]
      }
    ]
  })

  security_group_created = true
rescue Aws::EC2::Errors::InvalidGroupDuplicate
  puts "A security group with the name '#{security_group_name}' already exists."
end
```

begin ブロックが例外なく実行される場合は、security_group_created を true に設定します。

セキュリティグループに関する情報の取得

セキュリティグループを作成したので、既存のセキュリティグループとその IP のアクセス許可についての情報を出力します。

```
def describe_ip_permission(ip_permission)
  puts "-" * 22
  puts "IP Protocol: #{ip_permission.ip_protocol}"
  puts "From Port: #{ip_permission.from_port.to_s}"
  puts "To Port: #{ip_permission.to_port.to_s}"
  if ip_permission.ip_ranges.count > 0
    puts "IP Ranges:"
    ip_permission.ip_ranges.each do |ip_range|
      puts "  #{ip_range.cidr_ip}"
    end
  end
  if ip_permission.ipv_6_ranges.count > 0
    puts "IPv6 Ranges:"
    ip_permission.ipv_6_ranges.each do |ipv_6_range|
      puts "  #{ipv_6_range.cidr_ipv_6}"
    end
  end
  if ip_permission.prefix_list_ids.count > 0
    puts "Prefix List IDs:"
    ip_permission.prefix_list_ids.each do |prefix_list_id|
      puts "  #{prefix_list_id.prefix_list_id}"
    end
  end
  if ip_permission.user_id_group_pairs.count > 0
    puts "User ID Group Pairs:"
    ip_permission.user_id_group_pairs.each do |user_id_group_pair|
      puts "  ." * 7
      puts "  Group ID: #{user_id_group_pair.group_id}"
      puts "  Group Name: #{user_id_group_pair.group_name}"
      puts "  Peering Status: #{user_id_group_pair.peering_status}"
      puts "  User ID: #{user_id_group_pair.user_id}"
      puts "  VPC ID: #{user_id_group_pair.vpc_id}"
      puts "  VPC Peering Connection ID: #{user_id_group_pair.vpc_peering_connection_id}"
    end
  end
end

describe_security_groups_result = ec2.describe_security_groups

describe_security_groups_result.security_groups.each do |security_group|
  puts "\n"
  puts "*" * (security_group.group_name.length + 12)
```

```
  puts "Group Name: #{security_group.group_name}"
  puts "Group ID: #{security_group.group_id}"
  puts "Description: #{security_group.description}"
  puts "VPC ID: #{security_group.vpc_id}"
  puts "Owner ID: #{security_group.owner_id}"
  if security_group.ip_permissions.count > 0
    puts "=" * 22
    puts "IP Permissions:"
    security_group.ip_permissions.each do |ip_permission|
      describe_ip_permission(ip_permission)
    end
  end
  if security_group.ip_permissions_egress.count > 0
    puts "=" * 22
    puts "IP Permissions Egress:"
    security_group.ip_permissions_egress.each do |ip_permission|
      describe_ip_permission(ip_permission)
    end
  end
  if security_group.tags.count > 0
    puts "=" * 22
    puts "Tags:"
    security_group.tags.each do |tag|
      puts "  #{tag.key} = #{tag.value}"
    end
  end
end
```

セキュリティグループの削除

スクリプトの最後で、セキュリティグループが正常に作成され、`security_group_created` フラグが `true` に設定されたことを前提として、セキュリティグループを削除します。

```
if security_group_created
  ec2.delete_security_group({ group_id: create_security_group_result.group_id })
end
```

キーペアでの作業

以下の例は、AWS SDK for Ruby を Amazon EC2 で使用する方法を示します。

- キーペアの作成.
- キーペアに関する情報を取得します。
- キーペアを削除します。

キーペアの詳細については、Amazon EC2 User Guide for Linux Instancesの「Amazon EC2 のキーペア」または Amazon EC2 User Guide for Windows Instancesの「Amazon EC2 キーペアと Windows インスタンス」を参照してください。

これらの例を実行するために使用できるコードについては、「完全な例 (p. 42)」を参照してください。

キーペアの作成

create_key_pair メソッドを呼び出して、作成するキーペアの名前を指定します。

```
  key_pair = ec2.create_key_pair({
    key_name: key_pair_name
```

```
})
```

このコードでは:

- `ec2` は、Aws::EC2::Client オブジェクトを表す変数です。
- `key_pair_name` は、キーペアの名前を表す文字列変数です。
- `key_pair` は、`create_key_pair` メソッドの呼び出しによって返される Aws::EC2::KeyPair オブジェクトを表す変数です。

詳細については、「完全な例 (p. 42)」を参照してください。

キーペアに関する情報の取得

単一のキーペアに関する情報を取得するには、次のような属性を使用します。

- key_name は、キーペアの名前を取得します。
- key_fingerprint は、DER でエンコードされたプライベートキーの SHA-1 ダイジェストを取得します。
- key_material は、暗号化されていない PEM でエンコードされた RSA プライベートキーを取得します。

```
puts "Created key pair '#{key_pair.key_name}'."
puts "\nSHA-1 digest of the DER encoded private key:"
puts "#{key_pair.key_fingerprint}"
puts "\nUnencrypted PEM encoded RSA private key:"
puts "#{key_pair.key_material}"
```

このコードでは、`key_pair` は、Aws::EC2::KeyPair オブジェクトを表す変数で、前の例の create_key_pair メソッドを呼び出すことで返されます。

複数のキーペアに関する情報を取得するには、describe_key_pairs メソッドを呼び出します。

```
key_pairs_result = ec2.describe_key_pairs()

if key_pairs_result.key_pairs.count > 0
  puts "\nKey pair names:"
  key_pairs_result.key_pairs.each do |key_pair|
    puts key_pair.key_name
  end
end
```

このコードでは:

- `ec2` は、Aws::EC2::Client オブジェクトを表す変数です。
- `key_pair_result` は、`describe_key_pairs` メソッドの呼び出しによって返される Aws::EC2::Types::DescribeKeyPairsResult オブジェクトを表す変数です。
- `Aws::EC2::Types::DescribeKeyPairsResult` オブジェクトの key_pairs メソッドを呼び出すと、キーペアを表す Aws::EC2::Types::KeyPairInfo オブジェクトの配列が返されます。

詳細については、「完全な例 (p. 42)」を参照してください。

キーペアの削除

delete_key_pair メソッドを呼び出して、削除するキーペアの名前を指定します。

```
ec2.delete_key_pair({
```

```
  key_name: key_pair_name
})
```

このコードでは:

- `ec2` は、Aws::EC2::Client オブジェクトを表す変数です。
- `key_pair_name` は、キーペアの名前を表す文字列変数です。

詳細については、「完全な例 (p. 42)」を参照してください。

完全な例

適応して実行できる次のコードは、前の例を 1 つの例にまとめています。

```
require 'aws-sdk'

ec2 = Aws::EC2::Client.new(region: 'us-east-1')

key_pair_name = "my-key-pair"

# Create a key pair.
begin
  key_pair = ec2.create_key_pair({
    key_name: key_pair_name
  })
  puts "Created key pair '#{key_pair.key_name}'."
  puts "\nSHA-1 digest of the DER encoded private key:"
  puts "#{key_pair.key_fingerprint}"
  puts "\nUnencrypted PEM encoded RSA private key:"
  puts "#{key_pair.key_material}"
rescue Aws::EC2::Errors::InvalidKeyPairDuplicate
  puts "A key pair named '#{key_pair_name}' already exists."
end

# Get information about Amazon EC2 key pairs.
key_pairs_result = ec2.describe_key_pairs()

if key_pairs_result.key_pairs.count > 0
  puts "\nKey pair names:"
  key_pairs_result.key_pairs.each do |key_pair|
    puts key_pair.key_name
  end
end

# Delete the key pair.
ec2.delete_key_pair({
  key_name: key_pair_name
})
```

このコードを実行するには、以下を行う必要があります。

1. AWS SDK for Ruby をインストールします。詳細については、「AWS SDK for Ruby のインストール (p. 4)」を参照してください。
2. AWS のサービスおよびリソースへのアクセスを確認するために AWS SDK for Ruby が使用する、AWS アクセス認証情報を設定します。詳細については、「AWS SDK for Ruby の設定 (p. 5)」を参照してください。AWS 認証情報が、この例で説明されている AWS のアクションおよびリソースへのアクセス権を持つ AWS Identity and Access Management (IAM) のエンティティにマッピングされることを確認してください。この例では、認証情報が AWS 認証情報プロファイルのファイル、またはローカルシステムの `AWS_ACCESS_KEY_ID` および `AWS_SECRET_ACCESS_KEY` 環境変数で設定されていることを前提としています。

すべてのインスタンスに関する情報の取得

次の例では、us-west-2 リージョンのすべての Amazon EC2 インスタンスの ID および状態 (保留中、実行中、シャットダウン中、終了、停止、中止) を表示します。

```
require 'aws-sdk'

ec2 = Aws::EC2::Resource.new(region: 'us-west-2')

# To only get the first 10 instances:
# ec2.instances.limit(10).each do |i|
ec2.instances.each do |i|
  puts "ID:    #{i.id}"
  puts "State: #{i.state.name}"
end
```

特定のタグ値のあるすべてのインスタンスに関する情報の取得

次の例では、us-west-2 リージョンでタグ Group とタグ値 MyGroovyGroup のある Amazon EC2 インスタンスの ID および状態 (保留中、実行中、シャットダウン中、終了、停止、中止) を表示します。

> **Note**
>
> タグ名と値は大文字と小文字が区別されます。

```
require 'aws-sdk'

ec2 = Aws::EC2::Resource.new(region: 'us-west-2')

# Get all instances with tag key 'Group'
# and tag value 'MyGroovyGroup':
ec2.instances({filters: [{name: 'tag:Group', values: ['MyGroovyGroup']}]}).each do |i|
  puts 'ID:    ' + i.id
  puts 'State: ' + i.state.name
end
```

特定のインスタンスに関する情報の取得

次の例では、us-west-2 リージョンのインスタンス i-123abc の状態を示します。

```
require 'aws-sdk'

ec2 = Aws::EC2::Resource.new(region: 'us-west-2')

i = ec2.instance('i-123abc')

if i.exists?
  puts "State: #{i.state.name}"
end
```

インスタンスを作成する

次の例では、タグ Group と値 MyGroovyGroup を持つ Amazon EC2 インスタンス MyGroovyInstance を作成します。インスタンスは、マシンイメージ MACHINE_IMAGE で、ID ACCOUNT_ID のアカウント、ID SECURITY_GROUP_ID のセキュリティグループ、ID SUBNET_ID のサブネットで、アベイラビリ

ティーゾーン us-west-2a に作成されます。次に、インスタンスの ID とパブリック IP アドレスが表示されます。

> Note
>
> 空のスクリプト値には、Amazon EC2 インスタンスが起動するときに実行する手順を追加できます。

```
require 'aws-sdk'
require 'base64'

# User code that's executed when the instance starts
script = ''

encoded_script = Base64.encode64(script)

ec2 = Aws::EC2::Resource.new(region: 'us-west-2')

instance = ec2.create_instances({
  image_id: 'IMAGE_ID',
  min_count: 1,
  max_count: 1,
  key_name: 'MyGroovyKeyPair',
  security_group_ids: ['SECURITY_GROUP_ID'],
  user_data: encoded_script,
  instance_type: 't2.micro',
  placement: {
    availability_zone: 'us-west-2a'
  },
  subnet_id: 'SUBNET_ID',
  iam_instance_profile: {
    arn: 'arn:aws:iam::' + 'ACCOUNT_ID' + ':instance-profile/aws-opsworks-ec2-role'
  }
})

# Wait for the instance to be created, running, and passed status checks
ec2.client.wait_until(:instance_status_ok, {instance_ids: [instance[0].id]})

# Name the instance 'MyGroovyInstance' and give it the Group tag 'MyGroovyGroup'
instance.create_tags({ tags: [{ key: 'Name', value: 'MyGroovyInstance' }, { key: 'Group',
 value: 'MyGroovyGroup' }]})

puts instance.id
puts instance.public_ip_address
```

インスタンスの停止

次の例では、us-west-2 リージョンでインスタンス i-123abc を停止します。

```
require 'aws-sdk'

ec2 = Aws::EC2::Resource.new(region: 'us-west-2')

i = ec2.instance('i-123abc')

if i.exists?
  case i.state.code
  when 48  # terminated
    puts "#{id} is terminated, so you cannot stop it"
  when 64  # stopping
    puts "#{id} is stopping, so it will be stopped in a bit"
  when 89  # stopped
```

```
    puts "#{id} is already stopped"
  else
    i.stop
  end
end
```

インスタンスの作成

次の例では、us-west-2 リージョンでインスタンス i-123abc を起動します。

```
require 'aws-sdk'

ec2 = Aws::EC2::Resource.new(region: 'us-west-2')

i = ec2.instance('i-123abc')

if i.exists?
  case i.state.code
  when 0  # pending
    puts "#{id} is pending, so it will be running in a bit"
  when 16  # started
    puts "#{id} is already started"
  when 48  # terminated
    puts "#{id} is terminated, so you cannot start it"
  else
    i.start
  end
end
```

インスタンスの再起動

次の例では、us-west-2 リージョンでインスタンス i-123abc を再起動します。

```
require 'aws-sdk'

ec2 = Aws::EC2::Resource.new(region: 'us-west-2')

i = ec2.instance('i-123abc')

if i.exists?
  case i.state.code
  when 48  # terminated
    puts "#{id} is terminated, so you cannot reboot it"
  else
    i.reboot
  end
end
```

Amazon EC2 での Amazon EC2 インスタンスの管理

この例では、AWS SDK for Ruby を Amazon EC2 で使用して、以下のことを行います。

1. Aws::EC2::Client#stop_instances を使用して既存の Amazon EC2 インスタンスを停止する。
2. Aws::EC2::Client#start_instances を使用してインスタンスを再起動する。
3. Aws::EC2::Client#reboot_instances を使用してインスタンスを再起動する。
4. Aws::EC2::Client#monitor_instances を使用して、インスタンスの詳細モニタリングを有効にする。
5. Aws::EC2::Client#describe_instances を使用して、使用可能なインスタンスに関する情報を取得する。

前提条件

コード例を実行する前に、以下で説明されているように、AWS SDK for Ruby をインストールし、設定する必要があります。

- AWS SDK for Ruby のインストール (p. 4)
- AWS SDK for Ruby の設定 (p. 5)

また、コード内の INSTANCE-ID を、既存の EC2 インスタンスのインスタンス ID で置き換える必要があります。

例

```
require 'aws-sdk'

# Uncomment for Windows.
# Aws.use_bundled_cert!

def wait_for_instances(ec2, state, ids)
  begin
    ec2.wait_until(state, instance_ids: ids)
    puts "Success: #{state}."
  rescue Aws::Waiters::Errors::WaiterFailed => error
    puts "Failed: #{error.message}"
  end
end

ec2 = Aws::EC2::Client.new(region: 'us-east-1')

instance_id = "INSTANCE-ID" # For example, "i-0a123456b7c8defg9"

puts "Attempting to stop instance '#{instance_id}'. This may take a few minutes..."
ec2.stop_instances({ instance_ids: [instance_id] })
wait_for_instances(ec2, :instance_stopped, [instance_id])

puts "\nAttempting to restart instance '#{instance_id}'. This may take a few minutes..."
ec2.start_instances({ instance_ids: [instance_id] })
wait_for_instances(ec2, :instance_running, [instance_id])

puts "\nAttempting to reboot instance '#{instance_id}'. This may take a few minutes..."
ec2.reboot_instances({ instance_ids: [instance_id] })
wait_for_instances(ec2, :instance_status_ok, [instance_id])

# Enable detailed monitoring for the instance.
puts "\nAttempting to enable detailed monitoring for instance '#{instance_id}'..."

begin
  monitor_instances_result = ec2.monitor_instances({
    instance_ids: [instance_id]
  })
  puts "Detailed monitoring state for instance '#{instance_id}':
 #{monitor_instances_result.instance_monitorings[0].monitoring.state}"
rescue Aws::EC2::Errors::InvalidState
  puts "Instance '#{instance_id}' is not in a monitorable state. Continuing on..."
end

# Get information about available instances.
puts "\nAvailable instances:"

describe_instances_result = ec2.describe_instances

describe_instances_result.reservations.each do |reservation|
```

```
  if reservation.instances.count > 0
    reservation.instances.each do |instance|
      puts "=" * (instance.instance_id.length + 13)
      puts "Instance ID: #{instance.instance_id}"
      puts "State: #{instance.state.name}"
      puts "Image ID: #{instance.image_id}"
      puts "Instance Type: #{instance.instance_type}"
      puts "Architecure: #{instance.architecture}"
      puts "IAM Instance Profile: #{instance.iam_instance_profile}"
      puts "Key Name: #{instance.key_name}"
      puts "Launch Time: #{instance.launch_time}"
      puts "Detailed Monitoring State: #{instance.monitoring.state}"
      puts "Public IP Address: #{instance.public_ip_address}"
      puts "Public DNS Name: #{instance.public_dns_name}"
      puts "VPC ID: #{instance.vpc_id}"
      puts "Subnet ID: #{instance.subnet_id}"
      if instance.tags.count > 0
        puts "Tags:"
        instance.tags.each do |tag|
          puts "  #{tag.key} = #{tag.value}"
        end
      end
    end
  end

end
```

インスタンスを削除する

次の例では、`us-west-2` リージョンでインスタンス `i-123abc` を終了します。

```
require 'aws-sdk'

ec2 = Aws::EC2::Resource.new(region: 'us-west-2')

i = ec2.instance('i-123abc')

if i.exists?
  case i.state.code
  when 48  # terminated
    puts "#{id} is already terminated"
  else
    i.terminate
  end
end
```

リージョンおよびアベイラビリティーゾーンに関する情報の取得

以下の例は、AWS SDK for Ruby を Amazon EC2 と共に使用する方法を示します。

- 使用可能な Amazon EC2 リージョンおよびそのエンドポイントに関する情報を取得する。
- 使用可能な Amazon EC2 のアベイラビリティーゾーンに関する情報を取得する。

Amazon EC2 リージョンとアベイラビリティーゾーンの詳細については、Amazon EC2 User Guide for Linux Instancesの「リージョンとアベイラビリティーゾーン」を参照してください。

これらの例を実行するために使用できるコードについては、「完全な例 (p. 49)」を参照してください。

リージョンおよびエンドポイントに関する情報の取得

使用可能なリージョンに関する情報を取得するには、describe_regions メソッドを呼び出します。

```
describe_regions_result = ec2.describe_regions()
```

このコードでは、ec2 は Aws::EC2::Client オブジェクトを表す変数です。詳細については、「完全な例 (p. 49)」を参照してください。

リージョン名とエンドポイントを取得するには:

1. describe_regions メソッドから返され、このコードで describe_regions_result 変数で表される Aws::EC2::Types::DescribeRegionsResult オブジェクトを取得します。
2. リージョンを表す Aws::EC2::Types::Region オブジェクトの配列を取得するには、DescribeRegionsResult オブジェクトの regions 属性を使用します。
3. Region オブジェクトの region_name および endpoint 属性を使用してリージョンの名前およびエンドポイントを取得します。

```
describe_regions_result.regions.each do |region|
  puts "#{region.region_name} (#{region.endpoint})"
end
```

アベイラビリティーゾーンについて説明します

アベイラビリティーゾーンに関する情報を取得するには、describe_availability_zones メソッドを呼び出します。

```
describe_availability_zones_result = ec2.describe_availability_zones()
```

Aws::EC2::Types::DescribeAvailabilityZonesResult オブジェクトには、アベイラビリティーゾーンを表す Aws::EC2::Types::AvailabilityZone オブジェクトの配列が含まれています。DescribeAvailabilityZonesResult オブジェクトは、describe_availability_zones メソッドによって返され、このコードで describe_availability_zones_result 変数によって表されます。

このコードでは、ec2 は Aws::EC2::Client オブジェクトを表す変数です。詳細については、「完全な例 (p. 49)」を参照してください。

各アベイラビリティーゾーンの名前と状態を取得するには、AvailabilityZone オブジェクトの zone_name と state 属性を使用します。

```
describe_availability_zones_result.availability_zones.each do |zone|
  puts "#{zone.zone_name} is #{zone.state}"
  if zone.messages.count > 0
    zone.messages.each do |message|
      "  #{message.message}"
    end
  end
end
```

アベイラビリティーゾーンに関するメッセージを取得するには:

1. Aws::EC2::Types::AvailabilityZoneMessage 配列を返す AvailabilityZone オブジェクトの messages 属性を使用します。

2. 配列に少なくとも 1 つのメッセージがある場合は、各 `AvailabilityZoneMessage` オブジェクトの message 属性を使用してメッセージを取得します。

完全な例

適応して実行できる次のコードは、前の例を 1 つの例にまとめています。

```
require 'aws-sdk'

ec2 = Aws::EC2::Client.new(region: 'us-east-1')

puts "Amazon EC2 region(s) (and their endpoint(s)) that are currently available to you:\n
\n"
describe_regions_result = ec2.describe_regions()

describe_regions_result.regions.each do |region|
  puts "#{region.region_name} (#{region.endpoint})"
end

puts "\nAmazon EC2 availability zone(s) that are available to you for your current region:
\n\n"
describe_availability_zones_result = ec2.describe_availability_zones()

describe_availability_zones_result.availability_zones.each do |zone|
  puts "#{zone.zone_name} is #{zone.state}"
  if zone.messages.count > 0
    zone.messages.each do |message|
      "  #{message.message}"
    end
  end
end
```

このコードを実行するには:

1. AWS SDK for Ruby をインストールします。詳細については、「AWS SDK for Ruby のインストール (p. 4)」を参照してください。
2. AWS のサービスおよびリソースへのアクセスを確認するために AWS SDK for Ruby が使用する、AWS アクセス認証情報を設定します。詳細については、「AWS SDK for Ruby の設定 (p. 5)」を参照してください。AWS 認証情報が、この例で説明されている AWS のアクションおよびリソースへのアクセス権を持つ AWS Identity and Access Management (IAM) のエンティティにマッピングされることを確認してください。この例では、認証情報が AWS 認証情報プロファイルのファイル、またはローカルシステムの `AWS_ACCESS_KEY_ID` および `AWS_SECRET_ACCESS_KEY` 環境変数で設定されていることを前提としています。

AWS Elastic Beanstalk の例

AWS SDK for Ruby を使用して AWS Elastic Beanstalk (Elastic Beanstalk) にアクセスするには、次の例を使用できます。Elastic Beanstalk の詳細については、Elastic Beanstalk 開発者ガイドを参照してください。

例

トピック

- すべてのアプリケーションに関する情報の取得 (p. 50)
- 特定のアプリケーションに関する情報の取得 (p. 50)
- Ruby on Rails アプリケーションの更新 (p. 50)

すべてのアプリケーションに関する情報の取得

次の例では、us-west-2 リージョンのすべての Elastic Beanstalk アプリケーションの名前、説明、および URL をリスト表示します。

```
require 'aws-sdk'

elb = Aws::ElasticBeanstalk::Client.new(region: 'us-west-2')

elb.describe_applications.applications.each do |a|
  puts "Name:         #{a.application_name}"
  puts "Description:  #{a.description}"

  elb.describe_environments({application_name: a.application_name}).environments.each do |
env|
    puts "  Environment:  #{env.environment_name}"
    puts "    URL:        #{env.cname}"
    puts "    Health:     #{env.health}"
  end
end
```

特定のアプリケーションに関する情報の取得

次の例では、us-west-2 リージョンのアプリケーション MyRailsApp の名前、説明、URL をリスト表示します。

```
require 'aws-sdk'

elb = Aws::ElasticBeanstalk::Client.new(region: 'us-west-2')

app = elb.describe_applications({application_names: [args[0]]})

if app.exists?
  puts "Name:         #{app.application_name}"
  puts "Description:  #{app.description}"

  envs = elb.describe_environments({application_name: app.application_name})
  puts "URL:          #{envs.environments[0].cname}"
end
```

Ruby on Rails アプリケーションの更新

次の例では、us-west-2 リージョンの Rails アプリケーション MyRailsApp の Ruby を更新します。

> **Note**
>
> 正常にスクリプトを実行するには、Rails アプリケーションのルートにいる必要があります。

```
require 'aws-sdk'

Aws.config.update({region: 'us-west-2'})

elb = Aws::ElasticBeanstalk::Client.new
s3 = Aws::S3::Client.new

app_name = 'MyRailsApp'

# Get S3 bucket containing app
app_versions = elb.describe_application_versions({ application_name: app_name })
av = app_versions.application_versions[0]
```

```
bucket = av.source_bundle.s3_bucket
s3_key = av.source_bundle.s3_key

# Get info on environment
envs = elb.describe_environments({ application_name: app_name })
env = envs.environments[0]
env_name = env.environment_name

# Create new storage location
resp = elb.create_storage_location()

puts "Created storage location in bucket #{resp.s3_bucket}"

resp = s3.list_objects({
  prefix: s3_key,
  bucket: bucket
})

# Create ZIP file
zip_file_basename = SecureRandom.urlsafe_base64.to_s
zip_file_name = zip_file_basename + '.zip'

# Call out to OS to produce ZIP file
cmd = "git archive --format=zip -o #{zip_file_name} HEAD"
%x[ #{cmd} ]

# Get ZIP file contents
zip_contents = File.read(zip_file_name)

key = app_name + "\\" + zip_file_name

resp = s3.put_object({
  body: zip_contents,
  bucket: bucket,
  key: key
})

date = Time.new
today = date.day.to_s + "/" + date.month.to_s + "/" + date.year.to_s

elb.create_application_version({
  process: false,
  application_name: app_name,
  version_label: zip_file_basename,
  source_bundle: {
    s3_bucket: bucket,
    s3_key: key
  },
  description: "Updated #{today}"
})

elb.update_environment({
  environment_name: env_name,
  version_label: zip_file_basename
})
```

AWS Identity and Access Management の例

AWS SDK for Ruby を使用して AWS Identity and Access Management (IAM) にアクセスするには、次の例を使用できます。IAM の詳細については、IAM ドキュメントを参照してください。

例

トピック

すべてのユーザーに関する情報の取得

次の例では、us-west-2 リージョン内のすべての IAM ユーザーのグループ、ポリシーおよびアクセスキー ID がリスト表示されます。

```
require 'aws-sdk'

iam = Aws::IAM::Client.new(region: 'us-west-2')

iam.list_users.users.each do |user|
  name = user.user_name
  puts "For user #{name}"
  puts "  In groups:"

  iam.list_groups_for_user({user_name: name}).groups.each do |group|
    puts "    #{group.group_name}"
  end

  puts "  Policies:"

  iam.list_user_policies({user_name: name}).policy_names.each do |policy|
    puts "    #{policy}"
  end

  puts "  Access keys:"

  iam.list_access_keys({user_name: name}).access_key_metadata.each do |key|
    puts "    #{key.access_key_id}"
  end
end
```

新しいユーザーの追加

次の例では、パスワード REPLACE_ME で us-west-2 リージョンの IAM ユーザー my_groovy_user を作成し、ユーザーのアカウント ID を表示します。その名前のユーザーがすでに存在する場合は、メッセージが表示され、新しいユーザーは作成されません。

```
require 'aws-sdk'

iam = Aws::IAM::Client.new(region: 'us-west-2')

begin
  user = iam.create_user(user_name: 'my_groovy_user')
  iam.wait_until(:user_exists, user_name: 'my_groovy_user')
```

```
  user.create_login_profile({password: 'REPLACE_ME'})

  arn_parts = user.arn.split(':')
  puts 'Account ID:        ' + arn_parts[4]
rescue Aws::IAM::Errors::EntityAlreadyExists
  puts 'User already exists'
end
```

ユーザーのアクセスキーを作成する

次の例では、:code-:us-west-2 において、IAM ユーザー `my_groovy_user` のアクセスキーおよびシークレットキーを作成します。

```
require 'aws-sdk'

iam = Aws::IAM::Client.new(region: 'us-west-2')

begin
  user = iam.user(user_name: 'my_groovy_user')

  key_pair = user.create_access_key_pair

  puts "Access key: #{key_pair.access_key_id}"
  puts "Secret key: #{key_pair.secret}"
rescue Aws::IAM::Errors::NoSuchEntity => ex
  puts 'User does not exist'
end
```

管理ポリシーの追加

次の例では、リージョン `us-west-2` の IAM ユーザー `my_groovy_user` に管理ポリシー `AmazonS3FullAccess` を追加します。

```
require 'aws-sdk'

# Policy ARNs start with:
prefix = 'arn:aws:iam::aws:policy/'

policy_arn = prefix + 'AmazonS3FullAccess'

# In case the policy or user does not exist
begin
  client.attach_user_policy({user_name: 'my_groovy_user', policy_arn: policy_arn})
rescue Aws::IAM::Errors::NoSuchEntity => ex
  puts "Error attaching policy '#{policy_arn}'"
  puts ex.message
end
```

ロールの作成

次の例では、ロール `my_groovy_role` を作成し、Amazon EC2 が `us-west-2` リージョンで Amazon S3 と Amazon DynamoDB にアクセスできるようにします。

```
require 'aws-sdk'

client = Aws::IAM::Client.new(region: 'us-west-2')
iam = Aws::IAM::Resource.new(client: client)
```

```
# Let EC2 assume a role
policy_doc = {
  Version:"2012-10-17",
  Statement:[
    {
      Effect:"Allow",
      Principal:{
        Service:"ec2.amazonaws.com"
      },
      Action:"sts:AssumeRole"
  }]
}

role = iam.create_role({
  role_name: 'my_groovy_role',
  assume_role_policy_document: policy_doc.to_json
})

# Give the role full access to S3
role.attach_policy({
  policy_arn: 'arn:aws:iam::aws:policy/AmazonS3FullAccess'
})

# Give the role full access to DynamoDB
role.attach_policy({
  policy_arn: 'arn:aws:iam::aws:policy/AmazonDynamoDBFullAccess'
})
```

IAM ユーザーの管理

IAM ユーザーは、AWS とやり取りするユーザーまたはサービスを表します。IAM ユーザーの詳細については、「IAM ユーザー」を参照してください。

この例では、AWS SDK for Ruby を IAM で使用して、以下のことを行います。

1. Aws::IAM::Client#list_users を使用して、利用可能な AWS IAM ユーザーに関する情報を取得する。
2. Aws::IAM::Client#create_user を使用してユーザーを作成する。
3. Aws::IAM::Client#update_user を使用してユーザーの名前を更新する。
4. Aws::IAM::Client#delete_user を使用してユーザーを削除する。

前提条件

コード例を実行する前に、以下で説明されているように、AWS SDK for Ruby をインストールし、設定する必要があります。

- AWS SDK for Ruby のインストール (p. 4)
- AWS SDK for Ruby の設定 (p. 5)

例

```
require 'aws-sdk'

iam = Aws::IAM::Client.new(region: 'us-east-1')

user_name = "my-user"
changed_user_name = "my-changed-user"
```

```
# Get information about available AWS IAM users.
def list_user_names(iam)
  list_users_response = iam.list_users
  list_users_response.users.each do |user|
    puts user.user_name
  end
end

puts "User names before creating user..."
list_user_names(iam)

# Create a user.
puts "\nCreating user..."

iam.create_user({ user_name: user_name })

puts "\nUser names after creating user..."
list_user_names(iam)

# Update the user's name.
puts "\nChanging user's name..."

begin
  iam.update_user({
    user_name: user_name,
    new_user_name: changed_user_name
  })

  puts "\nUser names after updating user's name..."
  list_user_names(iam)
rescue Aws::IAM::Errors::EntityAlreadyExists
  puts "User '#{user_name}' already exists."
end

# Delete the user.
puts "\nDeleting user..."
iam.delete_user({ user_name: changed_user_name })

puts "\nUser names after deleting user..."
list_user_names(iam)
```

IAM ポリシーの使用

IAM ポリシーは、1 つ以上のアクセス権限を記述したドキュメントです。IAM ポリシーの詳細については、「IAM ポリシーの概要」を参照してください。

この例では、AWS SDK for Ruby を IAM で使用して、以下のことを行います。

1. Aws::IAM::Client#create_policy を使用してポリシーを作成する。
2. Aws::IAM::Client#get_policy を使用してポリシーに関する情報を取得する。
3. Aws::IAM::Client#attach_role_policy を使用してロールにポリシーをアタッチする。
4. Aws::IAM::Client#list_attached_role_policies を使用してロールにアタッチされたポリシーをリストする。
5. Aws::IAM::Client#detach_role_policy を使用してロールからポリシーをデタッチする。

前提条件

コード例を実行する前に、以下で説明されているように、AWS SDK for Ruby をインストールし、設定する必要があります。

- AWS SDK for Ruby のインストール (p. 4)
- AWS SDK for Ruby の設定 (p. 5)

スクリプトで指定されたロール (my-role) も作成する必要があります。これは、IAM コンソールで行うことができます。

例

```
require 'aws-sdk'

iam = Aws::IAM::Client.new(region: 'us-east-1')

role_name = "my-role"
policy_name = "my-policy"
policy_document = {
  "Version" => "2012-10-17",
  "Statement" => [
    {
      "Effect" => "Allow",
      "Action" => "s3:ListAllMyBuckets",
      "Resource" => "arn:aws:s3:::*"
    }
  ]
}.to_json

# Create a policy.
puts "Creating policy..."

create_policy_response = iam.create_policy({
  policy_name: policy_name,
  policy_document: policy_document
})

policy_arn = create_policy_response.policy.arn

# Get information about the policy.
get_policy_response = iam.get_policy({ policy_arn: policy_arn })

puts "\nCreated policy, ID = #{get_policy_response.policy.policy_id}"

# Attach the policy to a role.
puts "\nAttaching policy to role..."

iam.attach_role_policy({
  role_name: role_name,
  policy_arn: policy_arn
})

# List policies attached to the role.
puts "\nAttached role policy ARNs..."

iam.list_attached_role_policies({ role_name: role_name }).attached_policies.each do |
attached_policy|
  puts "  #{attached_policy.policy_arn}"
end

# Detach the policy from the role.
puts "\nDetaching role policy..."

iam.detach_role_policy({
  role_name: role_name,
  policy_arn: policy_arn
```

```
})
```

IAM アクセスキーの管理

ユーザーが SDK for Ruby からプログラムで AWS を呼び出すには、独自のアクセスキーが必要です。このニーズを満たすために、IAM ユーザーのアクセスキー (アクセスキー ID およびシークレットアクセスキー) を作成、修正、表示、および更新できます。デフォルトでは、アクセスキーを作成したときのステータスは Active です。これは、ユーザーが API コールにそのアクセスキーを使用できることを意味します。 アクセスキーの詳細については、「IAM ユーザーのアクセスキーの管理」を参照してください。

この例では、AWS SDK for Ruby を IAM で使用して、以下のことを行います。

1. Aws::IAM::Client#list_access_keys を使用して AWS IAM ユーザーアクセスキーをリストする。
2. Aws::IAM::Client#create_access_key を使用してアクセスキーを作成する。
3. Aws::IAM::Client#get_access_key_last_used を使用して、アクセスキーが最後にいつ使用されたか確認する。
4. Aws::IAM::Client#update_access_key を使用してアクセスキーを無効化する。
5. Aws::IAM::Client#delete_access_key を使用してアクセスキーを削除する。

前提条件

コード例を実行する前に、以下で説明されているように、AWS SDK for Ruby をインストールし、設定する必要があります。

- AWS SDK for Ruby のインストール (p. 4)
- AWS SDK for Ruby の設定 (p. 5)

スクリプトで指定されたユーザー (my-user) も作成する必要があります。新しい IAM ユーザーは、「新しいユーザーの追加 (p. 52)」で説明しているように、IAM コンソールでまたはプログラムで作成できます。

例

```
require 'aws-sdk'

iam = Aws::IAM::Client.new(region: 'us-east-1')

user_name = "my-user"

# List user access keys.
def list_keys(iam, user_name)
  begin
    list_access_keys_response = iam.list_access_keys({ user_name: user_name })

    if list_access_keys_response.access_key_metadata.count == 0
      puts "No access keys."
    else
      puts "Access keys:"
      list_access_keys_response.access_key_metadata.each do |key_metadata|
        puts "  Access key ID: #{key_metadata.access_key_id}"
      end
    end

  rescue Aws::IAM::Errors::NoSuchEntity
    puts "Cannot find user '#{user_name}'."
```

```
    exit(false)
  end
end

puts "Before creating access key..."
list_keys(iam, user_name)

# Create an access key.
puts "\nCreating access key..."

begin
  iam.create_access_key({ user_name: user_name })
  puts "\nAfter creating access key..."
  list_keys(iam, user_name)
rescue Aws::IAM::Errors::LimitExceeded
  puts "Too many access keys. Can't create any more."
end

# Determine when access keys were last used.
puts "\nKey(s) were last used..."

list_access_keys_response = iam.list_access_keys({ user_name: user_name })

list_access_keys_response.access_key_metadata.each do |key_metadata|
  resp = iam.get_access_key_last_used({ access_key_id: key_metadata.access_key_id })

  puts "  Key '#{key_metadata.access_key_id}' last used on
 #{resp.access_key_last_used.last_used_date}"

  # Deactivate access keys.
  puts "  Trying to deactivate this key..."

  iam.update_access_key({
    user_name: user_name,
    access_key_id: key_metadata.access_key_id,
    status: "Inactive"
  })
end

puts "\nAfter deactivating access key(s)..."
list_keys(iam, user_name)

# Delete the access key.
puts "\nDeleting access key..."

iam.delete_access_key({
  user_name: user_name,
  access_key_id: list_access_keys_response.access_key_metadata[0].access_key_id
})

puts "\nAfter deleting access key..."
list_keys(iam, user_name)
```

IAM サーバー証明書の使用

ウェブサイトまたは AWS のアプリケーションへの HTTPS 接続を有効にするには、SSL/TLS サーバー証明書が必要です。外部プロバイダーから入手した証明書をウェブサイトまたは AWS のアプリケーションで使用するには、証明書を IAM にアップロードするか、AWS Certificate Manager にインポートする必要があります。サーバー証明書の詳細については、「サーバー証明書の使用」を参照してください。

この例では、AWS SDK for Ruby を IAM で使用して、以下のことを行います。

1. Aws::IAM::Client#update_server_certificate を使用してサーバー証明書を更新する。

2. Aws::IAM::Client#delete_server_certificate を使用してサーバー証明書を削除する。
3. Aws::IAM::Client#list_server_certificates を使用して、残りのサーバー証明書に関する情報をリスト表示する。

前提条件

コード例を実行する前に、以下で説明されているように、AWS SDK for Ruby をインストールし、設定する必要があります。

- AWS SDK for Ruby のインストール (p. 4)
- AWS SDK for Ruby の設定 (p. 5)

サーバー証明書は常に存在している必要があります。存在しない場合、スクリプトは Aws::IAM::Errors::NoSuchEntity エラーをスローします。

例

```
require 'aws-sdk'

iam = Aws::IAM::Client.new(region: 'us-east-1')

server_certificate_name = "my-server-certificate"
changed_server_certificate_name = "my-changed-server-certificate"

# Update a server certificate.
iam.update_server_certificate({
  server_certificate_name: server_certificate_name,
  new_server_certificate_name: changed_server_certificate_name
})

# Delete the server certificate.
iam.delete_server_certificate({
  server_certificate_name: changed_server_certificate_name
})

# List information about any remaining server certificates.
list_server_certificates_response = iam.list_server_certificates

if list_server_certificates_response.server_certificate_metadata_list.count == 0
  puts "No server certificates."
else
  list_server_certificates_response.server_certificate_metadata_list.each do |
certificate_metadata|
    puts "-" * certificate_metadata.server_certificate_name.length
    puts "Name: #{certificate_metadata.server_certificate_name}"

    get_server_certificate_response = iam.get_server_certificate({
      server_certificate_name: "certificate_metadata.server_certificate_name"
    })
    puts "ID:
 #{get_server_certificate_response.server_certificate.server_certificate_metadata.server_certificate_id
  end
end
```

IAM アカウントエイリアスの管理

サインインページの URL に、AWS アカウント ID ではなく企業の名前または他のわかりやすい識別子を含めるには、AWS アカウント ID のエイリアスを作成します。AWS アカウントのエイリアスを作成する

と、サインインページの URL は変更され、エイリアスが組み込まれます。IAM アカウントエイリアスの詳細については、「AWS アカウント ID とその別名」を参照してください。

この例では、AWS SDK for Ruby を IAM で使用して、以下のことを行います。

1. Aws::IAM::Client#list_account_aliases を使用して AWS アカウントエイリアスをリストする。
2. Aws::IAM::Client#create_account_alias を使用してアカウントエイリアスを作成する。
3. Aws::IAM::Client#delete_account_alias を使用してアカウントエイリアスを削除する。

前提条件

コード例を実行する前に、以下で説明されているように、AWS SDK for Ruby をインストールし、設定する必要があります。

- AWS SDK for Ruby のインストール (p. 4)
- AWS SDK for Ruby の設定 (p. 5)

コード例では、my-account-alias 文字列を、すべてのアマゾン ウェブ サービスの製品の中で一意となる文字列に変更にします。

例

```
require 'aws-sdk'

iam = Aws::IAM::Client.new(region: 'us-east-1')

account_alias = "my-account-alias"

# List account aliases.
def list_aliases(iam)
  list_account_aliases_response = iam.list_account_aliases

  if list_account_aliases_response.account_aliases.count == 0
    puts "No account aliases."
  else
    puts "Aliases:"
    list_account_aliases_response.account_aliases.each do |account_alias|
      puts account_alias
    end
  end

end

puts "Before creating account alias..."
list_aliases(iam)

# Create an account alias.
puts "\nCreating account alias..."
iam.create_account_alias({ account_alias: account_alias })

puts "\nAfter creating account alias..."
list_aliases(iam)

# Delete the account alias.
puts "\nDeleting account alias..."
iam.delete_account_alias({ account_alias: account_alias })

puts "\nAfter deleting account alias..."
list_aliases(iam)
```

Lambda 例

AWS SDK for Ruby を使用して AWS Lambda (Lambda) にアクセスするには、次の例を使用できます。Lambda の詳細については、Lambda のドキュメントを参照してください。

例

トピック

すべての Lambda 関数に関する情報の表示

次の例では、`us-west-2` リージョンのすべての Lambda 関数の名前、ARN、およびロールを表示します。

```
require 'aws-sdk'

client = Aws::Lambda::Client.new(region: 'us-west-2')

client.list_functions.each do |function|
  puts 'Name: ' + function.function_name
  puts 'ARN:  ' + function.function_arn
  puts 'Role: ' + function.role
```

Lambda 関数の作成

次の例では、これらの値を使用して `us-west-2` リージョンで Lambda 関数 `my-notification-function` を作成します。

- ロールの ARN: `my-resource-arn`. ほとんどの場合、このロールのポリシーに `AWSLambdaExecute` 管理ポリシーのみをアタッチする必要があります。
- 関数のエントリポイント: `my-package.my-class`
- ランタイム: `java8`
- Zip ファイル: `my-zip-file.zip`
- バケット: `my-notification-bucket`
- キー: `my-zip-file`

```
require 'aws-sdk'

client = Aws::Lambda::Client.new(region: 'us-west-2')

args = {}
args[:role] = 'my-resource-arn'
args[:function_name] = 'my-notification-function'
args[:handler] = 'my-package.my-class'

# Also accepts nodejs, nodejs4.3, and python2.7
args[:runtime] = 'java8'
```

```
code = {}
code[:zip_file] = 'my-zip-file.zip'
code[:s3_bucket] = 'my-notification-bucket'
code[:s3_key] = 'my-zip-file'

args[:code] = code

client.create_function(args)
```

Lambda 関数の実行

次の例では、us-west-2 リージョンで Lambda 関数 MyGetitemsFunction を実行します。この関数は、データベースからの項目のリストを返します。入力 JSON は次のようになります。

```
{
   "SortBy": "name|time",
   "SortOrder": "ascending|descending",
   "Number": 50
}
```

各パラメータの意味は次のとおりです。

- SortBy は、結果のソート基準です。この例では time を使用します。この場合、返される項目はデータベースに追加された順にソートされます。
- SortOrder はソート順位です。この例では descending を使用します。この場合、最新の項目がリストの最後に表示されます。
- Number は、取得する項目の最大数です (デフォルトは 50)。この例では 10 を使用します。この場合、最新の 10 個の項目を取得します。

出力 JSON は次のようになります。

```
{
   "statusCode": 200|...,
   "body": {
      "result": "'success' or 'failure'",
      "error": "Error message if 'failure', '' otherwise"
      "data": [{"item1"}, ..., {"itemN"}]
   }
}
```

各パラメータの意味は次のとおりです。

- statusCode は HTTP ステータスコードで、200 は呼び出しが成功したことを意味します。
- body は返される JSON の本文です。
- result は呼び出しの結果 (success または failure) です。
- error は、result が failure の場合のエラーメッセージです。それ以外の場合は、空の文字列が返されます。
- data は、result が success の場合に返されます。それ以外の場合は、nil が返されます。

最初のステップでは、使用するモジュールをロードします。

- aws-sdk は、Lambda 関数を呼び出すために使用する AWS SDK for Ruby モジュールをロードします。
- json は、リクエストとレスポンスのペイロードをマーシャリングおよびアンマーシャリングするために使用する JSON モジュールをロードします。

- os は、Microsoft Windows で Ruby アプリケーションを実行するために使用する OS モジュールをロードします。別のオペレーティングシステムを使用中の場合は、これらの行を削除できます。

```
require 'aws-sdk'
require 'json'

# To run on Windows:
require 'os'
if OS.windows?
  Aws.use_bundled_cert!
end
```

次に、Lambda 関数を呼び出すために使用する Lambda クライアントを作成します。

```
client = Aws::Lambda::Client.new(region: 'us-west-2')
```

次に、リクエスト引数のハッシュを作成し、MyGetItemsFunction を呼び出します。

```
req_payload = {:SortBy => 'time', :SortOrder => 'descending', :NumberToGet => 10}
payload = JSON.generate(req_payload)

resp = client.invoke({
                        function_name: 'MyGetItemsFunction',
                        invocation_type: 'RequestResponse',
                        log_type: 'None',
                        payload: payload
                      })
```

最後に、応答を解析し、成功した場合は項目を出力します。

```
resp_payload = JSON.parse(resp.payload.string) # , symbolize_names: true)

# If the status code is 200, the call succeeded
if resp_payload["statusCode"] == 200
  # If the result is success, we got our items
  if resp_payload["body"]["result"] == "success"
    # Print out items
    resp_payload["body"]["data"].each do |item|
      puts item
    end
  end
end
```

GitHub で完全な例をご覧ください。

通知を受け取るように Lambda 関数を設定する

次の例では、ARN my-resource-arn でリソースから通知を受け入れるために、us-west-2 リージョンの Lambda 関数 my-notification-function を設定します。

```
require 'aws-sdk'

client = Aws::Lambda::Client.new(region: 'us-west-2')

args = {}
args[:function_name] = 'my-notification-function'
args[:statement_id] = 'lambda_s3_notification'
```

```
args[:action] = 'lambda:InvokeFunction'
args[:principal] = 's3.amazonaws.com'
args[:source_arn] = 'my-resource-arn'

client.add_permission{args]
```

Amazon Relational Database Service の例

AWS SDK for Ruby を使用して Amazon Relational Database Service (Amazon RDS) にアクセスするには、次の例を使用できます。Amazon RDS の詳細については、Amazon Relational Database Service ユーザーガイドを参照してください。

> Note
>
> `Aws::RDS::Resource` クラスの `2.2.18` バージョンで導入されたメソッドを使用している例を次にいくつか示します。これらの例を実行するには、`aws-sdk` gem のそのバージョン以降を使用する必要があります。

例

トピック

すべてのインスタンスに関する情報の取得

次の例では、`us-west-2` リージョンのすべての Amazon RDS インスタンスの名前 (ID) とステータスをリスト表示します。

```
require 'aws-sdk'

rds = Aws::RDS::Resource.new(region: 'us-west-2')

rds.db_instances.each do |i|
  puts "Name (ID): #{i.id}"
  puts "Status   : #{i.db_instance_status}"
  puts
end
```

すべてのスナップショットに関する情報の取得

次の例では、`us-west-2` リージョンのすべての Amazon RDS (インスタンス) スナップショットの名前 (ID) とステータスをリスト表示します。

```
require 'aws-sdk'
```

```
rds = Aws::RDS::Resource.new(region: 'us-west-2')

rds.db_snapshots.each do |s|
  puts "Name (ID): #{s.snapshot_id}"
  puts "Status:    #{s.status}"
end
```

すべてのクラスターとスナップショットに関する情報の取得

次の例では、us-west-2 リージョンのすべての Amazon RDS クラスターの名前 (ID) とステータス、およびスナップショットの名前 (ID) とステータスをリスト表示します。

```
require 'aws-sdk'

rds = Aws::RDS::Resource.new(region: 'us-west-2')

rds.db_clusters.each do |c|
  puts "Name (ID): #{c.id}"
  puts "Status:    #{c.status}"

  c.snapshots.each do |s|
    puts "  Snapshot: #{s.snapshot_id}"
    puts "  Status:   #{s.status}"
  end
end
```

すべてのセキュリティグループに関する情報の取得

次の例では、us-west-2 リージョンのすべての Amazon RDS セキュリティグループの名前をリスト表示します。

> **Note**
>
> Amazon RDS セキュリティグループは、Amazon EC2 Classic プラットフォームを使用している場合にのみ適用されます。Amazon EC2 VPC を使用している場合は、VPC セキュリティグループを使用します。どちらも例に示します。

```
require 'aws-sdk'

rds = Aws::RDS::Resource.new(region: 'us-west-2')

rds.db_instances.each do |i|
  # Show any security group IDs and descriptions
  puts 'Security Groups:'

  i.db_security_groups.each do |sg|
    puts sg.db_security_group_name
    puts '  ' + sg.db_security_group_description
    puts
  end

  # Show any VPC security group IDs and their status
  puts 'VPC Security Groups:'

  i.vpc_security_groups.each do |vsg|
    puts vsg.vpc_security_group_id
```

```
    puts '  ' + vsg.status
    puts
  end
end
```

すべてのサブネットグループに関する情報の取得

次の例では、us-west-2 リージョンのすべての Amazon RDS サブネットグループの名前とステータスをリスト表示します。

```
require 'aws-sdk'

rds = Aws::RDS::Resource.new(region: 'us-west-2')

rds.db_subnet_groups.each do |s|
  puts s.name
  puts '  ' + s.subnet_group_status
end
```

すべてのパラメーターグループに関する情報を取得する

次の例では、us-west-2 リージョンのすべての Amazon RDS パラメーターグループの名前と説明をリスト表示します。

```
require 'aws-sdk'

rds = Aws::RDS::Resource.new(region: 'us-west-2')

rds.db_parameter_groups.each do |p|
  puts p.db_parameter_group_name
  puts '  ' + p.description
end
```

インスタンスのスナップショットを作成する

次の例では、us-west-2 リージョンで instance_name で表される Amazon RDS インスタンスのスナップショットを作成します。

> **Note**
>
> インスタンスがクラスターのメンバーである場合、インスタンスのスナップショットを作成することはできませんが、代わりに、クラスターのスナップショットを作成する必要があります (クラスターのスナップショットを作成する (p. 67)を参照してください)。

```
require 'aws-sdk'

rds = Aws::RDS::Resource.new(region: 'us-west-2')

instance = rds.db_instance(instance_name)

date =  Time.new
date_time =  date.year.to_s +  '-' +  date.month.to_s +  '-' +  date.day.to_s +  '-' +
 date.hour.to_s +  '-' +  date.min.to_s
```

```
id = instance_name + '-' + date_time

instance.create_snapshot({db_snapshot_identifier: id})

puts "Created snapshot #{id}"
```

クラスターのスナップショットを作成する

次の例では、us-west-2 リージョンで cluster_name で表される Amazon RDS クラスターのスナップショットを作成します。

```
require 'aws-sdk'

rds = Aws::RDS::Resource.new(region: 'us-west-2')

cluster = rds.db_cluster(cluster_name)

date =  Time.new
date_time =  date.year.to_s +  '-' +  date.month.to_s +  '-' +  date.day.to_s +  '-' +
 date.hour.to_s +  '-' +  date.min.to_s

id = cluster_name + '-' + date_time

cluster.create_snapshot({db_cluster_snapshot_identifier: id})

puts "Created cluster snapshot #{id}"
```

Amazon S3 例

AWS SDK for Ruby を使用して Amazon Simple Storage Service (Amazon S3) にアクセスするには、次の例を使用できます。Amazon S3 に関する詳細は、Amazon S3 ドキュメントを参照してください。

例

トピック

すべてのバケットに関する情報の取得

次の例では、最大で 50 個の Amazon S3 バケットの名前をリスト表示します。コードをコピーし、buckets.rb という名前で保存します。Resource オブジェクトが us-west-2 リージョンで作成されますが、Amazon S3 はリージョンに関係なく、アクセスできるバケットを返すことに注意してください。

```
require 'aws-sdk'

region = 'us-west-2'
s3 = Aws::S3::Resource.new(region: region)

s3.buckets.limit(50).each do |b|
  puts "#{b.name}"
end
```

Note

リージョンを指定する場合、buckets メソッドが Client#list_buckets メソッドを呼び出し、リクエストの認証された送信者が所有するすべてのバケットのリストを返します。特定のリージョンのバケットのみを取得するためにこのリストをフィルタリングする方法については、リージョン内のすべてのバケットに関する情報の取得 (p. 68)を参照してください。

リージョン内のすべてのバケットに関する情報の取得

次の例では、us-west-2 リージョンの最初の 50 のバケットの名前がリスト表示されます。制限を指定しない場合、Amazon S3 は us-west-2 にあるすべてのバケットを表示します。

```
require 'aws-sdk'

region = 'us-west-2'
s3 = Aws::S3::Resource.new(region: region)

s3.buckets.limit(50).each do |b|
  if s3.client.get_bucket_location(bucket: b.name).location_constraint == region
    puts "#{b.name}"
  end
end
```

Note

バケットが Resource オブジェクトをインスタンス化したリージョンに存在しない場合、SDK は get_bucket_location を呼び出す時に警告メッセージを発します。STDERR をリダイレクトすると、このメッセージを抑制できます。
Windows では、2> nul をコマンドに追加します。
Linux または iOS では、2> /dev/null をコマンドに追加します。

Amazon S3 バケットを作成して使用する

この例では、AWS SDK for Ruby を使用して以下を行う方法を示しています。

1. Amazon S3 のバケットのリストを表示します。
2. バケットを作成します。
3. オブジェクト (ファイル) をバケットにアップロードします。
4. バケットにファイルをコピーします。

5. バケットからファイルを削除します。

この例の完全なコードについては、完全な例 (p. 71)を参照してください。

前提条件タスク

この例をセットアップして実行するには、まず以下の条件を満たす必要があります。

1. AWS SDK for Ruby をインストールします。詳細については、「AWS SDK for Ruby のインストール (p. 4)」を参照してください。
2. AWS のサービスおよびリソースへのアクセスを確認するために AWS SDK for Ruby が使用する、AWS アクセス認証情報を設定します。詳細については、「AWS SDK for Ruby の設定 (p. 5)」を参照してください。

AWS 認証情報が、この例で説明されている AWS のアクションおよびリソースへのアクセス権を持つ AWS Identity and Access Management (IAM) のエンティティにマッピングされることを確認してください。

この例では、認証情報が AWS 認証情報プロファイルのファイルおよび `david` という名前のファイルで設定されていることを前提としています。

SDK を設定する

この例では、Amazon S3 で AWS SDK for Ruby が提供するクラスとメソッドを使用し、JSON 形式のデータを取り扱うことができるように、`require` ステートメントを追加します。その後、バケットと指定の AWS プロファイルを作成する AWS リージョンに Aws::S3::Client オブジェクトを作成します。このコードは、`us-east-1` リージョンに `Aws::S3::Client` オブジェクトを作成します。追加変数は、この例で使用される 2 つのバケットにも宣言されます。

```
require 'aws-sdk'
require 'json'

profile_name = 'david'
region = "us-east-1"
bucket = 'doc-sample-bucket'
my_bucket = 'david-cloud'

# S3

# Configure SDK
s3 = Aws::S3::Client.new(profile: profile_name, region: region)
```

バケットのリストの取得

list_buckets メソッドを呼び出します。これはバケットのリストを表わす Aws::S3::Types::ListBucketsOutput クラスのインスタンスを返します。次に、`ListBucketsOutput` クラスの `buckets` 属性を使用して、各バケット名の `name` など、バケットのプロパティにアクセスします。

```
resp = s3.list_buckets
resp.buckets.each do |bucket|
  puts bucket.name
end
```

バケットの作成

バケット名を指定してバケット作成メソッドを呼び出します。

Note

バケット名は AWS アカウントで一意であるだけでなく、Amazon S3 で一意である必要があります。

```
s3.create_bucket(bucket: bucket)
```

オブジェクト (ファイル) のバケットへのアップロード

バケットの名前や作成するファイルの名前などの設定を指定して put_object メソッドを呼び出します。ファイルのコンテンツには、Ruby `File` クラスのインスタンスを指定するか、この例のように、ファイルのデータを表す文字列を指定できます。

ファイルが正常にアップロードされたかどうかを確認するには、list_objects_v2 メソッドを呼び出します。これはバケット内のオブジェクトを表す Aws::S3::Types::ListObjectsV2Output クラスのインスタンスを返します。次に、`ListObjectsV2Output` クラスの `contents` メソッドを使用して、各オブジェクト名の `key` など、オブジェクトのプロパティにアクセスします。

```
s3.put_object(bucket: bucket, key: "file1", body: "My first s3 object")

# Check the file exists
resp = s3.list_objects_v2(bucket: bucket)
resp.contents.each do |obj|
  puts obj.key
end
```

バケット間のファイルのコピー

オブジェクトを受信するターゲットバケットの名前 (`bucket`)、コピーするソースバケットとオブジェクトの名前 (`copy_source`)、ターゲットバケットにコピーする新規オブジェクトの名前 (`key`) を指定して、copy_object メソッドを呼び出します。

この例では、コピーするオブジェクトを含むバケットの名前は `#{my_bucket}` であり、`david-cloud` という名前のバケットです。コピーオペレーション後、`test_file` バケットの `david-cloud` は `doc-sample-bucket` バケットの `file2` に名前変更され、`david-cloud` バケットの `test_file1` は `doc-sample-bucket` バケットの `file3` に名前変更されます。

```
s3.copy_object(bucket: bucket,
               copy_source: "#{my_bucket}/test_file",
               key: 'file2')
s3.copy_object(bucket: bucket,
               copy_source: "#{my_bucket}/test_file1",
               key: 'file3')
```

バケットからのファイルの削除

delete_objects メソッドを呼び出します。`delete` 引数には、削除するオブジェクトを表す Aws::S3::Types::Delete タイプのインスタンスを使用します。この例では、`objects` が削除する 2 つのファイルを示します。

ファイルが正常に削除されたかどうかを確認するには、以前と同様に `list_objects_v2` メソッドを呼び出します。今回は、クラスの `contents` メソッドを使用すると、削除済みファイル名 (ここでは `key` で示されます) は表示されないはずです。

```
s3.delete_objects(
```

```
  bucket: 'doc-sample-bucket',
  delete: {
    objects: [
      {
        key: 'file2'
      },
      {
        key: 'file3'
      }
    ]
  }
)

# Verify objects now have been deleted
resp = s3.list_objects_v2(bucket: bucket)
resp.contents.each do |obj|
  puts obj.key
end
```

完全な例

この例の完全なコードを次に示します。

```
require 'aws-sdk'
require 'json'

profile_name = 'david'
region = "us-east-1"
bucket = 'doc-sample-bucket'
my_bucket = 'david-cloud'

# S3

# Configure SDK
s3 = Aws::S3::Client.new(profile: profile_name, region: region)

# Display a List of Amazon S3 Buckets
resp = s3.list_buckets
resp.buckets.each do |bucket|
  puts bucket.name
end

# Create a S3 bucket from S3::client
s3.create_bucket(bucket: bucket)

# Upload a file to s3 bucket, directly putting string data
s3.put_object(bucket: bucket, key: "file1", body: "My first s3 object")

# Check the file exists
resp = s3.list_objects_v2(bucket: bucket)
resp.contents.each do |obj|
  puts obj.key
end

# Copy files from bucket to bucket
s3.copy_object(bucket: bucket,
               copy_source: "#{my_bucket}/test_file",
               key: 'file2')
s3.copy_object(bucket: bucket,
               copy_source: "#{my_bucket}/test_file1",
               key: 'file3')

# Delete multiple objects in a single HTTP request
s3.delete_objects(
```

```
  bucket: 'doc-sample-bucket',
  delete: {
    objects: [
      {
        key: 'file2'
      },
      {
        key: 'file3'
      }
    ]
  }
)

# Verify objects now have been deleted
resp = s3.list_objects_v2(bucket: bucket)
resp.contents.each do |obj|
  puts obj.key
end
```

別のアプローチ

次の例では、us-west-2 リージョンで my-bucket という名前のバケットを作成します。この例では、Aws::S3::Client クラスではなく Aws::S3::Resource クラスのインスタンスを使用します。

```
require 'aws-sdk'

s3 = Aws::S3::Resource.new(region: 'us-west-2')
s3.create_bucket(bucket: 'my-bucket')
```

バケットがあるかどうかを確認

バケットがすでにあるかどうかを判断したい 2 つのケースがあります。条件が失敗する場合は、例外を受け取る代わりに、これらのテストを実行します。

- 自分がアクセスできるかどうかに関わりなく、すべてのバケットの中に特定のバケットの名前がすでにあるかどうかを判断したいと思います。このテストは、既存の名前のバケットを作成しようとして例外が起きるのを防ぐのに役立ちます。
- バケットに項目を追加するなどのオペレーションを、アクセスできるバケットにのみ実行したいと思います。

次のテストの例では、my-bucket という名前のバケットがすでにある場合、bucket_exists を true と設定します。Resource の region: パラメーターは結果に影響しません。

```
require 'aws-sdk'

s3 = Aws::S3::Resource.new(region: 'us-west-2')
bucket_exists = s3.bucket('my-bucket').exists?
```

次の例では、my-bucket という名前のバケットがありバケットにアクセスできる場合、bucket_exists を true と設定します。これも、Client の region パラメーターは結果に影響しません。

```
require 'aws-sdk'

bucket_exists = false
client = Aws::S3::Client.new(region: 'us-west-2')

begin
```

```
  resp = client.head_bucket({bucket: bucket_name, use_accelerate_endpoint: false})
  bucket_exists = true
rescue
end
```

バケット項目に関する情報の取得

署名付き URL の作成者がそのオブジェクトへのアクセス許可を持っている場合、署名付き URL を使用して、URL で識別されるオブジェクトにアクセスすることができます。署名付き URL を使用すれば、項目を公開せずにユーザーがリンクをクリックして項目を見ることを許可できます。

次の例では、`us-west-2` リージョンのバケット `my-bucket` の最初の 50 個の項目の名前と署名付き URL をリスト表示します。制限を指定しない場合、Amazon S3 は最大 1000 個の項目を表示します。

```
require 'aws-sdk'

s3 = Aws::S3::Resource.new(region: 'us-west-2')

bucket = s3.bucket('my-bucket')

# Show only the first 50 items
bucket.objects.limit(50).each do |item|
  puts "Name:  #{item.key}"
  puts "URL:   #{item.presigned_url(:get)}"
end
```

バケットに項目をアップロードする

次の例では、`us-west-2` リージョンのバケット `my-bucket` に項目 (ファイル) `C:file.txt` をアップロードします。`C:file.txt` はファイルの完全修飾名であるため、項目の名前がファイルの名前に設定されます。

```
require 'aws-sdk'

s3 = Aws::S3::Resource.new(region: 'us-west-2')

file = 'C:\file.txt'
bucket = 'my-bucket'

# Get just the file name
name = File.basename(file)

# Create the object to upload
obj = s3.bucket(bucket).object(name)

# Upload it
obj.upload_file(file)
```

バケットにメタデータの項目をアップロードする

次の例では、メタデータキー/値ペア `answer` と `42` の項目 (ファイル) `C:file.txt` を `us-west-2` リージョンのバケット `my-bucket` へアップロードします。`C:file.txt` はファイルの完全修飾名であるため、項目の名前がファイル名に設定されます。

```
require 'aws-sdk'

s3 = Aws::S3::Resource.new(region: 'us-west-2')
```

```
file = 'C:\file.txt'
bucket = 'my-bucket'

# Get just the file name
name = File.basename(file)

# Create the object to upload
obj = s3.bucket(bucket).object(name)

# Metadata to add
metadata = {"answer" => "42"}

# Upload it
obj.upload_file(file, metadata: metadata)
```

バケットからファイルにオブジェクトをダウンロードする

次の例では、us-west-2 リージョンのバケット my-bucket から項目 my-item のコンテンツを取得し、./my-code ディレクトリのファイル my-item.txt に保存します。

```
require 'aws-sdk'

s3 = Aws::S3::Resource.new(region: 'us-west-2')

# Create the object to retrieve
obj = s3.bucket('my-bucket').object('my-item')

# Get the item's content and save it to a file
obj.get(response_target: './my-code/my-item.txt')
```

バケット項目のプロパティを変更する

次の例では、us-west-2 リージョンのバケット my-bucket にある項目 my-item のパブリックの読み取り専用アクセスを追加し、サーバー側の暗号化を AES-256 に設定し、ストレージクラスを低冗長化に設定します。

```
require 'aws-sdk'

args_list = {}
args_list[:bucket] = 'my-bucket'
args_list[:key]    = 'my-item'

# Where we are getting the source to copy from
args_list[:copy_source] = 'my-bucket/my-item'

# The acl can be any of:
# private, public-read, public-read-write, authenticated-read, aws-exec-read, bucket-owner-
read, bucket-owner-full-control
args_list[:acl] = 'public-read'

# The encryption can be any of:
# AES256, aws:kms
args_list[:server_side_encryption] = 'AES256'

# The storage_class can be any of:
# STANDARD, REDUCED_REDUNDANCY, STANDARD_IA
args_list[:storage_class] = 'REDUCED_REDUNDANCY'
```

```
client = Aws::S3::Client.new(region: 'us-west-2')

client.copy_object(args_list)
```

項目がバケットに追加されたときに通知をトリガー

バケットのオブジェクトに変更があったときに通知をトリガーできます。これらの変更には次の場合が含まれます。

- オブジェクトがバケットに追加された場合
- オブジェクトをバケットから削除する場合
- 低冗長で保存されているオブジェクトが失われる場合

サービスの設定により通知の送信先を以下のものにできます。

- Amazon SNS トピック
- Amazon SQS キュー
- Lambda 関数

バケットの通知を作成するには

1. 項目をキューまたはトピックへ発行するか、または Lambda 関数を呼び出す許可を Amazon S3 に与えます (p. 75)。
2. バケットの通知設定がキュー、トピック、または関数を指すように設定します (p. 75)。

これらの手順を行うと、アプリケーションは情報に応答できます。たとえば、Lambda のトピック「プログラミングモデル」では、Lambda がサポートする多様なプログラミング言語を使用する方法を説明しています。

Amazon S3 で通知の送信を有効にする

Amazon Simple Notification Service トピックまたは Amazon Simple Queue Service キューの設定方法、または Lambda 関数を作成し、Amazon S3 がその関数に通知を送信できるようにする方法について説明します。

- トピックにパブリッシュするためにリソースを有効にする (p. 92)
- Amazon SQS でキューにパブリッシュするためにリソースを有効にする (p. 101)
- 通知を受け取るように Lambda 関数を設定する (p. 63)

Amazon S3 バケットの通知を作成する

次の例では、Amazon S3 バケット `my-bucket` に項目が追加されたときに、次の送信先にバケットが通知を送信できるようにします。

- ARN `my-topic-arn` のある Amazon SNS トピック
- ARN `my-queue-arn` のある Amazon SQS キュー
- ARN `my-function-arn` のある Lambda 関数

```
require 'aws-sdk'
```

```
req = {}
req[:bucket] = bucket_name

events = ['s3:ObjectCreated:*']

notification_configuration = {}

# Add function
lc = {}

lc[:lambda_function_arn] = 'my-function-arn'
lc[:events] = events
lambda_configurations = []
lambda_configurations << lc

notification_configuration[:lambda_function_configurations] = lambda_configurations

# Add queue
qc = {}

qc[:queue_arn] = 'my-topic-arn'
qc[:events] = events
queue_configurations = []
queue_configurations << qc

notification_configuration[:queue_configurations] = queue_configurations

# Add topic
tc = {}

tc[:topic_arn] = 'my-topic-arn'
tc[:events] = events
topic_configurations = []
topic_configurations << tc

notification_configuration[:topic_configurations] = topic_configurations

req[:notification_configuration] = notification_configuration

req[:use_accelerate_endpoint] = false

s3 = Aws::S3::Client.new(region: 'us-west-2')

s3.put_bucket_notification_configuration(req)
```

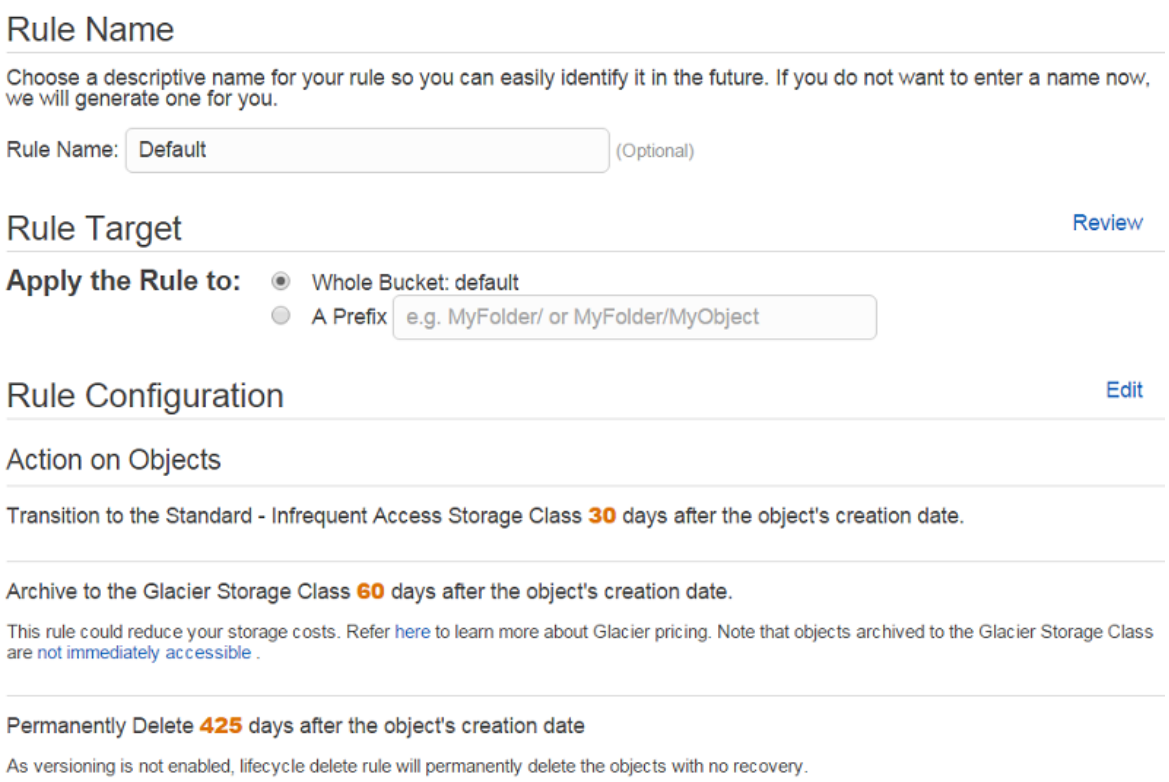
バケットのライフサイクルルール設定テンプレートの作成

オブジェクトが相当な数あり (または作成する予定があり)、それらを長期ストレージに移動するか削除するタイミングを特定したいと思う場合、ライフサイクルルールのテンプレートを作成し、そのテンプレートをすべてのバケットに適用することで時間を大幅に節約できます。

手順は以下のとおりです。

1. 既存のバケットのライフサイクル設定を手動で変更します。
2. ルールを保存します。
3. 他のバケットにルールを適用します。

次のルールで開始します。



そのルールの JSON 形式を生成するため次のコードを実行します。`default.json` という名前で出力をファイルに保存します。

```
require 'aws-sdk'

s3 = Aws::S3::Client.new(region: 'us-west-2')
resp = s3.get_bucket_lifecycle_configuration(bucket: 'default')

resp.rules.each do |rule|
  rule.to_hash.to_json
end
```

出力は次のようになります。

```
[{"expiration":
{"date":null,"days":425},"id":"default","prefix":"","status":"Enabled","transitions":
[{"date":null,"days":30,"storage_class":"STANDARD_IA"},
{"date":null,"days":60,"storage_class":"GLACIER"}],"noncurrent_version_transitions":
[],"noncurrent_version_expiration":null}]
```

ライフサイクルルールの JSON があるので、次の例を使用して他のバケットにも適用できます。`default.json` のルールがバケット `other_bucket` に適用されます。

```
require 'aws-sdk'
require 'json'

class Aws::S3::Types::LifecycleExpiration
  def to_map
    map = Hash.new
    self.members.each { |m| map[m] = self[m] }
    map
```

```
  end

  def to_json(*a)
    to_map.to_json(*a)
  end
end

class Aws::S3::Types::Transition
  def to_map
    map = Hash.new
    self.members.each { |m| map[m] = self[m] }
    map
  end

  def to_json(*a)
    to_map.to_json(*a)
  end
end

class Aws::S3::Types::LifecycleRule
  def to_map
    map = Hash.new
    self.members.each { |m| map[m] = self[m] }
    map
  end

  def to_json(*a)
    to_map.to_json(*a)
  end
end

# Pull in contents as a string
value = File.open('default.json', "rb").read
json_data = JSON.parse(value, opts={symbolize_names: true})

s3 = Aws::S3::Client.new(region: 'us-west-2')
s3.put_bucket_lifecycle_configuration(:bucket => 'other_bucket', :lifecycle_configuration
 => {:rules => json_data})
```

<admonition>
<title>ベストプラクティス</title>

Amazon S3 バケットで AbortIncompleteMultipartUpload ライフサイクルルールを有効にすることをお勧めします。

このルールは、開始後、指定された日数内に完了しないマルチパートアップロードを中止するよう Amazon S3 に指示できます。設定した時間制限を超えると、Amazon S3 はアップロードを中止して、不完全なアップロードデータを削除します。

詳細については、Amazon S3 User Guide の「バージョニングが有効なバケットのライフサイクル設定」をご覧ください。
</admonition>

Ruby でバケットポリシーを作成する

この例では、AWS SDK for Ruby を使用して以下を行う方法を示しています。

1. Amazon Simple Storage Service (Amazon S3) でバケットを作成します。
2. バケットポリシーを定義します。
3. ポリシーをバケットに追加します。
4. ポリシーを変更します。

5. ポリシーをバケットから削除します。
6. バケットを削除します。

この例の完全なコードについては、完全な例 (p. 81)を参照してください。

前提条件タスク

この例をセットアップして実行するには、まず以下の条件を満たす必要があります。

1. AWS SDK for Ruby をインストールします。詳細については、「AWS SDK for Ruby のインストール (p. 4)」を参照してください。
2. AWS のサービスおよびリソースへのアクセスを確認するために AWS SDK for Ruby が使用する、AWS アクセス認証情報を設定します。詳細については、「AWS SDK for Ruby の設定 (p. 5)」を参照してください。

AWS 認証情報が、この例で説明されている AWS のアクションおよびリソースへのアクセス権を持つ AWS Identity and Access Management (IAM) のエンティティにマッピングされることを確認してください。

この例では、認証情報が AWS 認証情報プロファイルのファイル、またはローカルシステムの `AWS_ACCESS_KEY_ID` および `AWS_SECRET_ACCESS_KEY` 環境変数で設定されていることを前提としています。

SDK を設定する

この例の SDK を設定するには、`require` ステートメントを追加して、AWS SDK for Ruby for Amazon S3 から提供されるクラスとメソッドを使用できるようにします。その後、バケットを作成する AWS リージョンで Aws::S3::Client のオブジェクトを作成します。このコードは、`us-west-2` リージョンに `Aws::S3::Client` オブジェクトを作成します。

```
require 'aws-sdk'

s3 = Aws::S3::Client.new(region: "us-west-2")
```

バケットの作成

バケット名を指定してバケット作成メソッドを呼び出します。このコードでは、`bucket` という名前の変数を使用してバケット名を表します。`example-bucket-name` をバケット名に置き換えてください。

> **Note**
>
> バケット名は AWS アカウントで一意であるだけでなく、Amazon S3 で一意である必要があります。

既に使用するバケットある場合、`create_bucket` を呼び出す必要はありません。

```
bucket = "example-bucket-name"

s3.create_bucket(bucket: bucket)
```

バケットポリシーの定義

ポリシーを表す Ruby ハッシュを宣言します。次に、ハッシュで `to_json` メソッドを呼び出して JSON オブジェクトに変換します。このコードは、ポリシー定義を含む `policy` という変数を使用します。こ

のポリシーは、指定されたユーザーが example-bucket-name (#{bucket} で表される) を完全に制御できます。arn:aws:iam::111122223333:user/Alice を、使用する AWS Identity and Access Management (IAM) ユーザーの Amazon Resource Name (ARN) に置き換えます。

```
policy = {
  "Version" => "2012-10-17",
  "Statement" => [
    {
      "Effect" => "Allow",
      "Principal" => {
        "AWS" => [
          "arn:aws:iam::111122223333:user/Alice"
        ]
      },
      "Action" => "s3:*",
      "Resource" => [
        "arn:aws:s3:::#{bucket}"
      ]
    }
  ]
}.to_json
```

定義できるポリシーの種類の例については、Amazon S3 Developer Guideの「バケットポリシーの例」を参照してください。

ポリシーをバケットに追加する

バケットの名前とポリシー定義を指定して、put_bucket_policy メソッドを呼び出します。

```
s3.put_bucket_policy(
  bucket: bucket,
  policy: policy
)
```

ポリシーの変更

put_bucket_policy メソッドを呼び出してポリシーを完全に置き換えることもできます。ただし、既存のポリシーの増分更新を行うこともできます。こうすることで、作成が必要なコードの量を減らすことができます。これを行うには、get_bucket_policy メソッドを呼び出すことで、現在のポリシーを取得します。次に、Ruby ハッシュに返された JSON オブジェクトを解析します。その後、ポリシーに対する増分変更を行います。たとえば、このコードは IAM エンティティの ARN を変更します。変更を行った後は、put_bucket_policy メソッドを再度呼び出します。変更したポリシーをバケットに適用する前に、ハッシュで to_json メソッドを呼び出して JSON オブジェクトに変換して戻してください。

```
policy_string = s3.get_bucket_policy(bucket: bucket).policy.read
policy_json = JSON.parse(policy_string)

policy_json["Statement"][0]["Principal"]["AWS"] = "arn:aws:iam::111122223333:root"

s3.put_bucket_policy(
  bucket: bucket,
  policy: policy_json.to_json
)
```

クリーンアップ

バケットからポリシーを削除するには、バケット名を指定して delete_bucket_policy メソッドを呼び出します。

バケットを削除するには、バケット名を指定して delete_bucket メソッドを呼び出します。

```
s3.delete_bucket_policy(bucket: bucket)
s3.delete_bucket(bucket: bucket)
```

完全な例

この例の完全なコードを次に示します。

```
require 'aws-sdk'

s3 = Aws::S3::Client.new(region: "us-west-2")

bucket = "example-bucket-name"

s3.create_bucket(bucket: bucket)

policy = {
  "Version" => "2012-10-17",
  "Statement" => [
    {
      "Effect" => "Allow",
      "Principal" => {
        "AWS" => [
          "arn:aws:iam::111122223333:user/Alice"
        ]
      },
      "Action" => "s3:*",
      "Resource" => [
        "arn:aws:s3:::#{bucket}"
      ]
    }
  ]
}.to_json

s3.put_bucket_policy(
  bucket: bucket,
  policy: policy
)

policy_string = s3.get_bucket_policy(bucket: bucket).policy.read
policy_json = JSON.parse(policy_string)

policy_json["Statement"][0]["Principal"]["AWS"] = "arn:aws:iam::111122223333:root"

s3.put_bucket_policy(
  bucket: bucket,
  policy: policy_json.to_json
)

s3.delete_bucket_policy(bucket: bucket)
s3.delete_bucket(bucket: bucket)
```

Cross-Origin Resource Sharing (CORS) のバケットの設定

この例では、AWS SDK for Ruby を使用して以下を行う方法を示しています。

1. Amazon S3 バケットの CORS 設定を構成します。
2. バケットの CORS 設定を取得します。

Amazon S3 の CORS サポートについては、Amazon S3 Developer Guideの「Cross-Origin Resource Sharing (CORS)」を参照してください。

この例の完全なコードについては、完全な例 (p. 83)を参照してください。

前提条件タスク

この例をセットアップして実行するには、まず以下の条件を満たす必要があります。

1. AWS SDK for Ruby をインストールします。詳細については、「AWS SDK for Ruby のインストール (p. 4)」を参照してください。
2. AWS のサービスおよびリソースへのアクセスを確認するために AWS SDK for Ruby が使用する、AWS アクセス認証情報を設定します。詳細については、「AWS SDK for Ruby の設定 (p. 5)」を参照してください。
3. Amazon S3 バケットを作成するか、AWS アカウントの既存バケットを指定します。

AWS 認証情報が、この例で説明されている AWS のアクションおよびリソースへのアクセス権を持つ AWS Identity and Access Management (IAM) のエンティティにマッピングされることを確認してください。

この例は以下を前提としています。

- AWS 認証情報プロファイルのファイルおよび `david` という名前のプロファイルに認証情報を設定してある。
- バケット名が `doc-sample-bucket` である。

SDK を設定する

この例では、AWS SDK for Ruby for Amazon S3 により提供されるクラスとメソッドを使用できるように、`require` ステートメントを追加します。その後、バケットと指定の AWS プロファイルを作成する AWS リージョンに Aws::S3::Client オブジェクトを作成します。このコードは、`us-east-1` リージョンに `Aws::S3::Client` オブジェクトを作成します。追加変数は、この例で使用されるバケットにも宣言されます。

```
require 'aws-sdk'

profile_name = 'david'
region = 'us-east-1'
bucket = 'doc-sample-bucket'

# S3 - Configuring an S3 Bucket

# Create a S3 client
s3 = Aws::S3::Client.new(profile: profile_name, region:region)
```

バケットの CORS の設定

バケットの名前と CORS 構成設定を指定して、put_bucket_cors メソッドを呼び出します。

```
s3.put_bucket_cors(
  bucket: bucket,
  cors_configuration: cors_configuration
)
```

CORS 構成設定で、Aws::S3::Types::CORSConfiguration ハッシュを宣言します。指定されたオリジンが実行を許可されている (`allowed_methods`) HTTP メソッド、顧客がバケットにアクセスできるオリ

ジン (allowed_origins)、顧客がアプリケーションからアクセスできるレスポンスのヘッダー (たとえば、JavaScript XMLHttpRequest オブジェクトから、expose_headers に示されているように) などを指定します。

```
cors_configuration = {
  cors_rules: [
    {
      allowed_methods: allowed_methods,
      allowed_origins: ["*"],
      expose_headers: ["ExposeHeader"],
    },
  ]
}
```

指定されたオリジンが実行を許可される HTTP メソッドでは、インラインで指定するか、またはここで示されているようにコマンドラインでユーザーから取得できます。

```
allowed_methods = []
ARGV.each do |arg|
  case arg.upcase
  when "POST"
    allowed_methods << "POST"
  when "GET"
    allowed_methods << "GET"
  when "PUT"
    allowed_methods << "PUT"
  when "PATCH"
    allowed_methods << "PATCH"
  when "DELETE"
    allowed_methods << "DELETE"
  when "HEAD"
    allowed_methods << "HEAD"
  else
    puts "#{arg} is not a valid HTTP method"
  end
end
```

たとえば、コードファイルが doc_sample_code_s3_bucket_cors.rb という名前であり、指定されたオリジンに GET および POST メソッドのみを許可する場合、ユーザーは次のようにコマンドラインからコードを実行します。

```
ruby doc_sample_code_s3_bucket_cors.rb get post
```

バケットの CORS 設定の取得

バケット名を指定して、get_bucket_cors メソッドを呼び出します。get_bucket_cors メソッドは Aws::S3::Types::GetBucketCorsOutput オブジェクトを返します。このオブジェクトの cors_rules 属性は、バケットの CORS 設定を表わす Aws::S3::Types::CORSRule オブジェクトの配列を返します。

```
resp = s3.get_bucket_cors(bucket: bucket)
puts resp.cors_rules
```

完全な例

この例の完全なコードを次に示します。

```
require 'aws-sdk'
```

```
profile_name = 'david'
region = 'us-east-1'
bucket = 'doc-sample-bucket'

# S3 - Configuring an S3 Bucket

# Create a S3 client
s3 = Aws::S3::Client.new(profile: profile_name, region:region)

# Setting a Bucket CORS Configuration

# Create array of allowed methods parameter based on command line parameters
allowed_methods = []
ARGV.each do |arg|
  case arg.upcase
  when "POST"
    allowed_methods << "POST"
  when "GET"
    allowed_methods << "GET"
  when "PUT"
    allowed_methods << "PUT"
  when "PATCH"
    allowed_methods << "PATCH"
  when "DELETE"
    allowed_methods << "DELETE"
  when "HEAD"
    allowed_methods << "HEAD"
  else
    puts "#{arg} is not a valid HTTP method"
  end
end

# Create CORS configuration hash
cors_configuration = {
  cors_rules: [
    {
      allowed_methods: allowed_methods,
      allowed_origins: ["*"],
      expose_headers: ["ExposeHeader"],
    },
  ]
}

# Set the new CORS configuration on the selected bucket
s3.put_bucket_cors(
  bucket: bucket,
  cors_configuration: cors_configuration
)

# Retrieving a Bucket CORS Configuration
resp = s3.get_bucket_cors(bucket: bucket)
puts resp.cors_rules

# To run the example, type the following at the command line including one or more HTTP
 methods as shown
# ruby doc_sample_code_s3_bucket_cors.rb get post
```

Amazon S3 バケットおよびオブジェクトのアクセス許可の管理

この例では、AWS SDK for Ruby を使用して以下を行う方法を示しています。

1. Amazon S3 のバケットに事前定義された許可 (既定 ACL ともいいます) を設定します。

2. バケットにオブジェクトを追加します。
3. バケット内のオブジェクトの既定 ACL を設定します。
4. バケットの現在の ACL を取得します。

この例の完全なコードについては、完全な例 (p. 86)を参照してください。

前提条件タスク

この例をセットアップして実行するには、まず以下の条件を満たす必要があります。

1. AWS SDK for Ruby をインストールします。詳細については、「AWS SDK for Ruby のインストール (p. 4)」を参照してください。
2. AWS のサービスおよびリソースへのアクセスを確認するために AWS SDK for Ruby が使用する、AWS アクセス認証情報を設定します。詳細については、「AWS SDK for Ruby の設定 (p. 5)」を参照してください。

AWS 認証情報が、この例で説明されている AWS のアクションおよびリソースへのアクセス権を持つ AWS Identity and Access Management (IAM) のエンティティにマッピングされることを確認してください。

この例では、認証情報が AWS 認証情報プロファイルのファイル、またはローカルシステムの `AWS_ACCESS_KEY_ID` および `AWS_SECRET_ACCESS_KEY` 環境変数で設定されていることを前提としています。

SDK を設定する

この例では、AWS SDK for Ruby for Amazon S3 により提供されるクラスとメソッドを使用できるように、`require` ステートメントを追加します。その後、バケットを作成する AWS リージョンで Aws::S3::Client のオブジェクトを作成します。このコードは、`us-west-2` リージョンに `Aws::S3::Client` オブジェクトを作成します。このコードでは、バケットを表す変数も宣言されます。

```
require 'aws-sdk'

# Create a S3 client
client = Aws::S3::Client.new(region: 'us-west-2')
```

バケットの既定 ACL の設定

既定 ACL 名とバケットを指定して put_bucket_acl メソッドを呼び出します。このコードは `public-read` 既定 ACL をバケットに設定して、バケットの所有者に完全コントロールを許可し、その他には読み取り専用アクセスを許可します。

```
client.put_bucket_acl({
  acl: "public-read",
  bucket: bucket,
})
```

既定 ACL の詳細については、Amazon S3 Developer Guideの「アクセスコントロールリスト (ACL) の概要」の「既定 ACL」を参照してください。

この設定を確認するには、Ruby の `Net::HTTP.get` メソッドを呼び出してバケットのコンテンツの取得を試みます。

```
bucket_path = "http://#{bucket}.s3-us-west-2.amazonaws.com/"
```

```
resp = Net::HTTP.get(URI(bucket_path))
puts "Content of unsigned request to #{bucket_path}:\n\n#{resp}\n\n"
```

オブジェクトのバケットへのアップロード

バケットとオブジェクトの名前とオブジェクトのコンテンツを指定して put_object メソッドを呼び出します。このコードでは、オブジェクトを表わす変数が宣言されます。

```
object_key = "my-key"
# Put an object in the public bucket
client.put_object({
  bucket: bucket,
  key: object_key,
  body: 'Hello World',
})
```

オブジェクトの既定 ACL の設定

デフォルトでは、バケットのオブジェクトのコンテンツは取得できません。この動作を確認するには、Ruby の `Net::HTTP.get` メソッドを呼び出してオブジェクトのコンテンツの取得を試みます。

```
object_path = "http://#{bucket}.s3-us-west-2.amazonaws.com/#{object_key}"
resp = Net::HTTP.get(URI(object_path))
puts "Content of unsigned request to #{object_path}:\n\n#{resp}\n\n"
```

この動作を変更するには、既定 ACL、バケット、オブジェクトの名前を指定して、put_object_acl メソッドを呼び出します。このコードは `public-read` 既定 ACL をオブジェクトに設定して、オブジェクトの所有者に完全コントロールを許可し、その他には読み取り専用アクセスを許可します。呼び出した後、再びオブジェクトのコンテンツの取得を試みます。

```
client.put_object_acl({
  acl: "public-read",
  bucket: bucket,
  key: object_key,
})
object_path = "http://#{bucket}.s3-us-west-2.amazonaws.com/#{object_key}"
puts "Now I can access object (#{object_key}) :\n#{Net::HTTP.get(URI(object_path))}\n\n"
```

バケットの現在の ACL の取得

バケットの名前を指定して、get_bucket_acl メソッドを呼び出します。`get_bucket_acl` メソッドは、Aws::S3::Types::GetBucketAclOutput クラスのインスタンスを返します。`GetBucketAclOutput` クラスの `grants` 属性を使用して、バケットの現行 ACL をリストします。

```
resp = client.get_bucket_acl(bucket: bucket)
puts resp.grants
```

完全な例

この例の完全なコードを次に示します。

```
require 'aws-sdk'

# Create a S3 client
client = Aws::S3::Client.new(region: 'us-west-2')
```

```
bucket = 'my-bucket'
# Sets a bucket to public-read
client.put_bucket_acl({
  acl: "public-read",
  bucket: bucket,
})

object_key = "my-key"
# Put an object in the public bucket
client.put_object({
  bucket: bucket,
  key: object_key,
  body: 'Hello World',
})

# Accessing an object in the bucket with unauthorize request
bucket_path = "http://#{bucket}.s3-us-west-2.amazonaws.com/"
resp = Net::HTTP.get(URI(bucket_path))
puts "Content of unsigned request to #{bucket_path}:\n\n#{resp}\n\n"

# However, accessing the object is denied since object Acl is not public-read
object_path = "http://#{bucket}.s3-us-west-2.amazonaws.com/#{object_key}"
resp = Net::HTTP.get(URI(object_path))
puts "Content of unsigned request to #{object_path}:\n\n#{resp}\n\n"

# Setting the object to public-read
client.put_object_acl({
  acl: "public-read",
  bucket: bucket,
  key: object_key,
})
object_path = "http://#{bucket}.s3-us-west-2.amazonaws.com/#{object_key}"
puts "Now I can access object (#{object_key}) :\n#{Net::HTTP.get(URI(object_path))}\n\n"

# Setting bucket to private again
client.put_bucket_acl({
  bucket: bucket,
  acl: 'private',
})

# Get current bucket Acl
resp = client.get_bucket_acl(bucket: bucket)
puts resp.grants

resp = Net::HTTP.get(URI(bucket_path))
puts "Content of unsigned request to #{bucket_path}:\n\n#{resp}\n\n"
```

Amazon S3 バケットを使用してウェブサイトをホストする

この例では、AWS SDK for Ruby を使用して以下を行う方法を示しています。

1. Amazon S3 バケットを作成します。
2. バケットウェブサイトの設定を取得します。
3. バケットにオブジェクトを追加します。
4. バケットウェブサイト設定を設定します。
5. バケットウェブサイトのドキュメントにアクセスします。
6. バケットウェブサイトを削除します。
7. バケットを削除します。

バケットウェブサイトのホスティングについての詳細は、Amazon S3 Developer Guideの「バケットをウェブサイトホスティング用に設定」を参照してください。

この例の完全なコードについては、完全な例 (p. 90)を参照してください。

前提条件タスク

この例をセットアップして実行するには、まず以下の条件を満たす必要があります。

1. AWS SDK for Ruby をインストールします。詳細については、「AWS SDK for Ruby のインストール (p. 4)」を参照してください。
2. AWS のサービスおよびリソースへのアクセスを確認するために AWS SDK for Ruby が使用する、AWS アクセス認証情報を設定します。詳細については、「AWS SDK for Ruby の設定 (p. 5)」を参照してください。

AWS 認証情報が、この例で説明されている AWS のアクションおよびリソースへのアクセス権を持つ AWS Identity and Access Management (IAM) のエンティティにマッピングされることを確認してください。

この例では、認証情報が AWS 認証情報プロファイルのファイル、またはローカルシステムの `AWS_ACCESS_KEY_ID` および `AWS_SECRET_ACCESS_KEY` 環境変数で設定されていることを前提としています。

SDK を設定する

この例では、AWS SDK for Ruby for Amazon S3 により提供されるクラスとメソッドを使用できるように、`require` ステートメントを追加します。その後、バケットと指定の AWS プロファイルを作成する AWS リージョンに Aws::S3::Client オブジェクトを作成します。このコードは、`us-east-2` リージョンに `Aws::S3::Client` オブジェクトを作成します。

追加変数は、この例で使用されるバケットにも宣言されます。バケット名がすべての AWS のアカウントで一意であることを確認するために、追加の `require` ステートメントが追加され、`SecureRandom` モジュールの `uuid` メソッドが呼び出されて一意の識別子が生成されます。この識別名は、このサンプルの後半に作成されるバケット名に挿入されます。

```
require 'aws-sdk'

# Using Random UUIDs to Avoid Collisions when Testing
require 'securerandom'
bucket = "example-test-bucket-#{SecureRandom.uuid}"

# Setup
s3 = Aws::S3::Client.new(region: "us-west-2")
```

バケットの作成

create_bucket メソッドを呼び出して、作成するバケット名を指定します。

```
s3.create_bucket(bucket: bucket)
```

バケットウェブサイト設定の取得

get_bucket_website メソッドを呼び出して、バケット名を指定します。デフォルトでは、バケットはウェブサイトとして設定されていません。この動作を確認するには、`get_bucket_website` メソッドを呼び出します。バケットのウェブサイト設定がないため、エラーが返されます。

```
begin
```

```
  s3.get_bucket_website(bucket: bucket)
rescue Aws::S3::Errors::NoSuchWebsiteConfiguration
  puts "No bucket website configuration present."
end
```

バケットにオブジェクトを追加

put_object メソッドを呼び出して、バケットおよびオブジェクトの名前、オブジェクトの内容、およびオブジェクトのアクセス権限の設定を指定します。この例では、2 つのウェブページにバケットを追加します。

```
s3.put_object(
  bucket: bucket,
  key: "index.html",
  body: "Hello, Amazon S3!",
  acl: "public-read"
)
s3.put_object(
  bucket: bucket,
  key: "error.html",
  body: "Page not found!",
  acl: "public-read"
)
```

バケットウェブサイト設定の設定

put_bucket_cors メソッドを呼び出して、バケット名およびウェブサイトの設定を指定します。ウェブサイトの設定では、Aws::S3::Types::WebsiteConfiguration ハッシュを使用して、ウェブサイトのインデックスおよびエラーウェブページを指定します。

```
s3.put_bucket_website(
  bucket: bucket,
  website_configuration: {
    index_document: {
      suffix: "index.html"
    },
    error_document: {
      key: "error.html"
    }
  }
)
```

バケットウェブサイトのドキュメントにアクセス

Ruby の `Net::HTTP.get` メソッドを呼び出して、バケットウェブサイトのドキュメントにアドレスを指定します。

```
index_path = "http://#{bucket}.s3-website-us-west-2.amazonaws.com/"
error_path = "http://#{bucket}.s3-website-us-west-2.amazonaws.com/nonexistant.html"

puts "Index Page Contents:\n#{Net::HTTP.get(URI(index_path))}\n\n"
puts "Error Page Contents:\n#{Net::HTTP.get(URI(error_path))}\n\n"
```

バケットウェブサイトの削除

delete_bucket_website メソッドを呼び出して、バケット名を指定します。

```
s3.delete_bucket_website(bucket: bucket)
```

バケットの削除

Aws::S3::Resource オブジェクトの bucket メソッドを呼び出して、バケット名を指定します。これにより、Aws::S3::Bucket オブジェクトが返されます。次に、`Aws::S3::Bucket` オブジェクトの delete メソッドを呼び出します。

```
b = Aws::S3::Resource.new(region: "us-west-2").bucket(bucket)
b.delete! # Recursively deletes objects as well.
```

完全な例

この例の完全なコードを次に示します。

```
require 'aws-sdk'

# Using Random UUIDs to Avoid Collisions when Testing
require 'securerandom'
bucket = "example-test-bucket-#{SecureRandom.uuid}"

# Setup
s3 = Aws::S3::Client.new(region: "us-west-2")
s3.create_bucket(bucket: bucket)

# When Bucket Has No Website Configuration
begin
  s3.get_bucket_website(bucket: bucket)
rescue Aws::S3::Errors::NoSuchWebsiteConfiguration
  puts "No bucket website configuration present."
end

# Adding Simple Pages & Website Configuration
s3.put_object(
  bucket: bucket,
  key: "index.html",
  body: "Hello, Amazon S3!",
  acl: "public-read"
)
s3.put_object(
  bucket: bucket,
  key: "error.html",
  body: "Page not found!",
  acl: "public-read"
)
s3.put_bucket_website(
  bucket: bucket,
  website_configuration: {
    index_document: {
      suffix: "index.html"
    },
    error_document: {
      key: "error.html"
    }
  }
)

# Accessing as a Website
index_path = "http://#{bucket}.s3-website-us-west-2.amazonaws.com/"
error_path = "http://#{bucket}.s3-website-us-west-2.amazonaws.com/nonexistant.html"

puts "Index Page Contents:\n#{Net::HTTP.get(URI(index_path))}\n\n"
puts "Error Page Contents:\n#{Net::HTTP.get(URI(error_path))}\n\n"
```

```
# Removing Website Configuration
s3.delete_bucket_website(bucket: bucket)

# Cleanup
b = Aws::S3::Resource.new(region: "us-west-2").bucket(bucket)
b.delete! # Recursively deletes objects as well.
```

Amazon SNS 例

AWS SDK for Ruby を使用して Amazon Simple Notification Service (Amazon SNS) にアクセスするには、次の例を使用できます。Amazon SNS に関する詳細は、Amazon SNS ドキュメントを参照してください。

例

トピック

すべてのトピックに関する情報を取得する

次の例では、us-west-2 リージョンの SNS トピックの ARN をリスト表示します。

```
require 'aws-sdk'

sns = Aws::SNS::Resource.new(region: 'us-west-2')

sns.topics.each do |topic|
  puts topic.arn
end
```

トピックの作成

次の例では、us-west-2 リージョンでトピック MyGroovyTopic を作成し、結果として出るトピック ARN を表示します。

```
require 'aws-sdk'

sns = Aws::SNS::Resource.new(region: 'us-west-2')

topic = sns.create_topic(name: 'MyGroovyTopic')
puts topic.arn
```

トピックのすべてのサブスクリプションに関する情報を取得する

次の例では、us-west-2 リージョンの ARN arn:aws:sns:us-west-2:123456789:MyGroovyTopic でトピックの SNS サブスクリプションの E メールアドレスをリスト表示します。

```
require 'aws-sdk'

sns = Aws::SNS::Resource.new(region: 'us-west-2')

topic = sns.topic('arn:aws:sns:us-west-2:123456789:MyGroovyTopic')

topic.subscriptions.each do |s|
  puts s.attributes['Endpoint']
end
```

トピックのサブスクリプションを作成する

次の例では、us-west-2 リージョンでメールアドレス MyGroovyUser@MyGroovy.com を持つユーザーの ARN arn:aws:sns:us-west-2:123456789:MyGroovyTopic でトピックのサブスクリプションを作成し、結果として出る ARN を表示します。最初、ARN 値は確認が保留中です。ユーザーが E メールアドレスを確認すると、この値は正しい ARN になります。

```
require 'aws-sdk'

sns = Aws::SNS::Resource.new(region: 'us-west-2')

topic = sns.topic('arn:aws:sns:us-west-2:123456789:MyGroovyTopic')

sub = topic.subscribe({
  protocol: 'email',
  endpoint: 'MyGroovyUser@MyGroovy.com'
})

puts sub.arn
```

トピックのサブスクライバーすべてにメッセージを送信

次の例では、「Hello!」というメッセージを ARN arn:aws:sns:us-west-2:123456789:MyGroovyTopic のトピックのサブスクライバーすべてに送信します。

```
require 'aws-sdk'

sns = Aws::SNS::Resource.new(region: 'us-west-2')

topic = sns.topic('arn:aws:sns:us-west-2:123456789:MyGroovyTopic')

topic.publish({
  message: 'Hello!'
})
```

トピックにパブリッシュするためにリソースを有効にする

次の例では、us-west-2 リージョンで ARN my-topic-arn のトピックに発行する ARN my-resource-arn でリソースを有効にします。

```
require 'aws-sdk'

sns = Aws::SNS::Resource.new(region: 'us-west-2')
```

```
policy  = '{
  "Version":"2008-10-17",
  "Id":"__default_policy_ID",
  "Statement":[{
    "Sid":"__default_statement_ID",
    "Effect":"Allow",
    "Principal":{
      "AWS":"*"
    },
    "Action":["SNS:Publish"],
    "Resource":"' + my-topic-arn + '",
    "Condition":{
      "ArnEquals":{
        "AWS:SourceArn":"' + my-resource-arn + '"}
     }
  }]
}'

topic.set_attributes({
  attribute_name: "Policy",
  attribute_value: policy
})
```

Amazon SQS 例

AWS SDK for Ruby を使用して Amazon Simple Queue Service (Amazon SQS) にアクセスするには、次の例を使用できます。Amazon SQS に関する詳細は、Amazon SQS ドキュメントを参照してください。

例

トピック

Amazon SQS でのすべてのキューに関する情報の取得

次の例では、`us-west-2` リージョンで、Amazon SQS キューの URL、ARN、利用可能なメッセージ、および処理中のメッセージをリスト表示します。

```
require 'aws-sdk'

sqs = Aws::SQS::Client.new(region: 'us-west-2')
```

```
queues = sqs.list_queues

queues.queue_urls.each do |url|
  puts 'URL:                ' + url

  # Get ARN, messages available, and messages in flight for queue
  req = sqs.get_queue_attributes(
    {
      queue_url: url, attribute_names:
        [
          'QueueArn',
          'ApproximateNumberOfMessages',
          'ApproximateNumberOfMessagesNotVisible'
        ]
    }
  )

  arn = req.attributes['QueueArn']
  msgs_available = req.attributes['ApproximateNumberOfMessages']
  msgs_in_flight = req.attributes['ApproximateNumberOfMessagesNotVisible']

  puts 'ARN:                ' + arn
  puts 'Messages available: ' + msgs_available
  puts 'Messages in flight: ' + msgs_in_flight
  puts
end
```

Amazon SQS でのキューの作成

次の例では、`us-west-2` リージョンで Amazon SQS キュー `MyGroovyQueue` を作成し、URL を表示します。

```
require 'aws-sdk'

sqs = Aws::SQS::Client.new(region: 'us-west-2')

queue = sqs.create_queue(queue_name: 'MyGroovyQueue')

puts queue.queue_url
```

Amazon SQS でのキューの使用

Amazon SQS は、アプリケーションまたはマイクロサービス間をメッセージが移動する間に、それらを保存するための、スケーラブルなホストされたキューを提供します。キューの詳細については、「Amazon SQS キューの操作」を参照してください。

この例では、AWS SDK for Ruby を Amazon SQS で使用して、以下のことを行います。

1. Aws::SQS::Client#list_queues を使用してキューのリストを取得する。
2. Aws::SQS::Client#create_queue を使用してキューを作成する。
3. Aws::SQS::Client#get_queue_url を使用してキューの URL を取得する。
4. Aws::SQS::Client#delete_queue を使用してキューを削除する。

前提条件

コード例を実行する前に、以下で説明されているように、AWS SDK for Ruby をインストールし、設定する必要があります。

- AWS SDK for Ruby のインストール (p. 4)
- AWS SDK for Ruby の設定 (p. 5)

例

```
require 'aws-sdk'

sqs = Aws::SQS::Client.new(region: 'us-east-1')

# Get a list of your queues.
sqs.list_queues.queue_urls.each do |queue_url|
  puts queue_url
end

# Create a queue.
queue_name = "my-queue"

begin
  sqs.create_queue({
    queue_name: queue_name,
    attributes: {
      "DelaySeconds" => "60", # Delay message delivery for 1 minute (60 seconds).
      "MessageRetentionPeriod" => "86400" # Delete message after 1 day (24 hours * 60
 minutes * 60 seconds).
    }
  })
rescue Aws::SQS::Errors::QueueDeletedRecently
  puts "A queue with the name '#{queue_name}' was recently deleted. Wait at least 60
 seconds and try again."
  exit(false)
end

# Get the queue's URL.
queue_url = sqs.get_queue_url(queue_name: queue_name).queue_url
puts queue_url

# Delete the queue.
sqs.delete_queue(queue_url: queue_url)
```

Amazon SQS メッセージの送信

次の例では、`us-west-2` リージョンの URL `URL` を使用して Amazon SQS キューを通して「Hello world」というメッセージを送信します。

```
require 'aws-sdk'

sqs = Aws::SQS::Client.new(region: 'us-west-2')

sqs.send_message(queue_url: URL, message_body: 'Hello world')
```

次の例では、「Hello world」および「How is the weather?」というメッセージを `us-west-2` リージョンで URL `URL` を使用して Amazon SQS キューを通して送信します。

```
require 'aws-sdk'

sqs = Aws::SQS::Client.new(region: 'us-west-2')

resp = sqs.send_message_batch({
```

```
  queue_url: URL,
  entries: [
    {
      id: 'msg1',
      message_body: 'Hello world'
    },
    {
      id: 'msg2',
      message_body: 'How is the weather?'
    }
  ],
})
```

Amazon SQS でのメッセージの送受信

Amazon SQS でキューを作成したら、そのキューにメッセージを送信して消費できます。詳細については、「チュートリアル: Amazon SQS キューへのメッセージの送信」および「チュートリアル: Amazon SQS キューからのメッセージの受信および削除」を参照してください。

この例では、AWS SDK for Ruby を Amazon SQS で使用して、以下のことを行います。

1. Aws::SQS::Client#send_message を使用して、キューにメッセージを送信します。
2. Aws::SQS::Client#receive_message を使用してキューでメッセージを受信します。
3. メッセージに関する情報を表示します。
4. Aws::SQS::Client#delete_message を使用してキューからメッセージを削除します。

前提条件

コード例を実行する前に、以下で説明されているように、AWS SDK for Ruby をインストールし、設定する必要があります。

- AWS SDK for Ruby のインストール (p. 4)
- AWS SDK for Ruby の設定 (p. 5)

また、キュー my-queue を作成する必要があります。これは Amazon SQS コンソールで行うことができます。

例

```
require 'aws-sdk'

sqs = Aws::SQS::Client.new(region: 'us-east-1')

# Send a message to a queue.
queue_name = "my-queue"

begin
  queue_url = sqs.get_queue_url(queue_name: queue_name).queue_url

  # Create a message with three custom attributes: Title, Author, and WeeksOn.
  send_message_result = sqs.send_message({
    queue_url: queue_url,
    message_body: "Information about current NY Times fiction bestseller for week of
 2016-12-11.",
    message_attributes: {
      "Title" => {
```

```
        string_value: "The Whistler",
        data_type: "String"
      },
      "Author" => {
        string_value: "John Grisham",
        data_type: "String"
      },
      "WeeksOn" => {
        string_value: "6",
        data_type: "Number"
      }
    }
  })
rescue Aws::SQS::Errors::NonExistentQueue
  puts "A queue named '#{queue_name}' does not exist."
  exit(false)
end

puts send_message_result.message_id

# Receive the message in the queue.
receive_message_result = sqs.receive_message({
  queue_url: queue_url,
  message_attribute_names: ["All"], # Receive all custom attributes.
  max_number_of_messages: 1, # Receive at most one message.
  wait_time_seconds: 0 # Do not wait to check for the message.
})

# Display information about the message.
# Display the message's body and each custom attribute value.
receive_message_result.messages.each do |message|
  puts message.body
  puts "Title: #{message.message_attributes["Title"]["string_value"]}"
  puts "Author: #{message.message_attributes["Author"]["string_value"]}"
  puts "WeeksOn: #{message.message_attributes["WeeksOn"]["string_value"]}"

  # Delete the message from the queue.
  sqs.delete_message({
    queue_url: queue_url,
    receipt_handle: message.receipt_handle
  })
end
```

Amazon SQS でのメッセージの受信

次の例では、`us-west-2` リージョンの URL `URL` を使用して Amazon SQS キューに最大 10 通のメッセージの本文を表示します。

> **Note**
>
> `receive_message` はすべてのメッセージの受信を保証してはいません (分散キューのプロパティを参照)。また、デフォルトではメッセージは削除されません。

```
require 'aws-sdk'

sqs = Aws::SQS::Client.new(region: 'us-west-2')

resp = sqs.receive_message(queue_url: URL, max_number_of_messages: 10)

resp.messages.each do |m|
  puts m.body
end
```

Amazon SQS でのロングポーリングを使用したメッセージの受信

次の例では、us-west-2 リージョンの URL URL を使用して Amazon SQS キューに最大 10 通のメッセージの本文を表示するため、10 秒まで待機します。

待機時間を指定しなければ、デフォルト値は 0 です (Amazon SQS は待機しません)。

```
require 'aws-sdk'

sqs = Aws::SQS::Client.new(region: 'us-west-2')

resp = sqs.receive_message(queue_url: URL, max_number_of_messages: 10, wait_time_seconds:
 10)

resp.messages.each do |m|
  puts m.body
end
```

Amazon SQS でのロングポーリングの有効化

ロングポーリングにより、空のレスポンスを減らし、偽の空のレスポンスを除外することで、Amazon SQSの使用コストの削減に役立ちます。ロングポーリングの詳細については、「Amazon SQS ロングポーリング」を参照してください。

この例では、AWS SDK for Ruby を Amazon SQS で使用して、以下のことを行います。

1. Aws::SQS::Client#create_queue を使用してキューを作成し、ロングポーリング用に設定する。
2. Aws::SQS::Client#set_queue_attributes を使用して既存のキュー用のロングポーリングを設定する。
3. Aws::SQS::Client#receive_message を使用して、キューのメッセージを受信するときにロングポーリングを設定する。

前提条件

コード例を実行する前に、以下で説明されているように、AWS SDK for Ruby をインストールし、設定する必要があります。

- AWS SDK for Ruby のインストール (p. 4)
- AWS SDK for Ruby の設定 (p. 5)

また、キュー existing-queue と receive-queue を作成する必要があります。これは Amazon SQS コンソールで行うことができます。

例

```
require 'aws-sdk'

sqs = Aws::SQS::Client.new(region: 'us-east-1')

# Create a queue and set it for long polling.
new_queue_name = "new-queue"

create_queue_result = sqs.create_queue({
  queue_name: new_queue_name,
```

```
  attributes: {
    "ReceiveMessageWaitTimeSeconds" => "20" # Wait 20 seconds to receive messages.
  },
})

puts create_queue_result.queue_url

# Set long polling for an existing queue.
begin
  existing_queue_name = "existing-queue"
  existing_queue_url = sqs.get_queue_url(queue_name: existing_queue_name).queue_url

  sqs.set_queue_attributes({
    queue_url: existing_queue_url,
    attributes: {
      "ReceiveMessageWaitTimeSeconds" => "20" # Wait 20 seconds to receive messages.
    },
  })
rescue Aws::SQS::Errors::NonExistentQueue
  puts "Cannot set long polling for a queue named '#{existing_queue_name}', as it does not
 exist."
end

# Set long polling when receiving messages for a queue.

# 1. Using receive_message.
begin
  receive_queue_name = "receive-queue"
  receive_queue_url = sqs.get_queue_url(queue_name: receive_queue_name).queue_url

  puts "Begin receipt of any messages using receive_message..."
  receive_message_result = sqs.receive_message({
    queue_url: receive_queue_url,
    attribute_names: ["All"], # Receive all available built-in message attributes.
    message_attribute_names: ["All"], # Receive any custom message attributes.
    max_number_of_messages: 10 # Receive up to 10 messages, if there are that many.
  })

  puts "Received #{receive_message_result.messages.count} message(s)."
rescue Aws::SQS::Errors::NonExistentQueue
  puts "Cannot receive messages using receive_message for a queue named
 '#{receive_queue_name}', as it does not exist."
end

# 2. Using Aws::SQS::QueuePoller.
begin
  puts "Begin receipt of any messages using Aws::SQS::QueuePoller..."
  puts "(Will keep polling until no more messages available for at least 60 seconds.)"
  poller = Aws::SQS::QueuePoller.new(receive_queue_url)

  poller_stats = poller.poll({
    max_number_of_messages: 10,
    idle_timeout: 60 # Stop polling after 60 seconds of no more messages available (polls
 indefinitely by default).
  }) do |messages|
    messages.each do |message|
      puts "Message body: #{message.body}"
    end
  end
  # Note: If poller.poll is successful, all received messages are automatically deleted
 from the queue.

  puts "Poller stats:"
  puts "  Polling started at: #{poller_stats.polling_started_at}"
  puts "  Polling stopped at: #{poller_stats.polling_stopped_at}"
  puts "  Last message received at: #{poller_stats.last_message_received_at}"
```

```
  puts "  Number of polling requests: #{poller_stats.request_count}"
  puts "  Number of received messages: #{poller_stats.received_message_count}"
rescue Aws::SQS::Errors::NonExistentQueue
  puts "Cannot receive messages using Aws::SQS::QueuePoller for a queue named
 '#{receive_queue_name}', as it does not exist."
end
```

Amazon SQS で QueuePoller クラスを使用してメッセージを受信

次の例では、us-west-2 リージョンの URL URL を持つ SQS キューのすべてのメッセージの本文を表示する QueuePoller ユーティリティクラスを使用し、メッセージを削除します。約 15 秒間のアイドル状態の後、スクリプトはタイムアウトします。

```
require 'aws-sdk'

Aws.config.update({region: 'us-west-2'})

poller = Aws::SQS::QueuePoller.new(URL)

poller.poll(idle_timeout: 15) do |msg|
  puts msg.body
end
```

次の例では、URL URL を持つ Amazon SQS キューをループし、最大時間 (秒) 待機します。

「Amazon SQS でのすべてのキューに関する情報の取得 (p. 93)」の Amazon SQS の例を実行することで、正しい URL を取得できます。

```
require 'aws-sdk'

Aws.config.update({region: 'us-west-2'})

poller = Aws::SQS::QueuePoller.new(URL)

poller.poll(wait_time_seconds: duration, idle_timeout: duration + 1) do |msg|
  puts msg.body
end
```

次の例では、URL URL を持つ Amazon SQS キューをループし、可視性タイムアウトの秒数の範囲内で do_something メソッドで表されるように、メッセージを処理します。

```
require 'aws-sdk'

# Process the message
def do_something(msg)
  puts msg.body
end

Aws.config.update({region: 'us-west-2'})

poller = Aws::SQS::QueuePoller.new(URL)

poller.poll(timeout_visibility: timeout, idle_timeout: timeout + 1) do |msg|
  do_something(msg)
end
```

次の例では、URL URL を持つ Amazon SQS キューをループし、do_something2 メソッドにより追加の処理が必要なすべてのメッセージについて可視性タイムアウトの秒数を変更します。

```
require 'aws-sdk'

# Process the message
def do_something(msg)
  true
end

# Do additional processing
def do_something2(msg)
  puts msg.body
end

Aws.config.update({region: 'us-west-2'})

poller = Aws::SQS::QueuePoller.new(URL)

poller.poll(idle_timeout: timeout + 1) do |msg|
  if do_something(msg)
    # need more time for processing
    poller.change_message_visibility_timeout(msg, timeout)

    do_something2(msg)
  end
end
```

Amazon SQS でのデッドレターのリダイレクト

次の例では、URL `URL` を持つキューから ARN `ARN` を持つキューにデッドレターをリダイレクトします。

```
require 'aws-sdk'

sqs = Aws::SQS::Client.new(region: 'us-west-2')

sqs.set_queue_attributes({
  queue_url: URL,
  attributes:
    {
      'RedrivePolicy' => "{\"maxReceiveCount\":\"5\", \"deadLetterTargetArn\":\"#{ARN}\"}"
    }
})
```

Amazon SQS でのキューの削除

次の例では、`us-west-2` リージョンの URL `URL` を持つ Amazon SQS キューを削除します。

```
require 'aws-sdk'

sqs = Aws::SQS::Client.new(region: 'us-west-2')

sqs.delete_queue(queue_url: URL)
```

Amazon SQS でキューにパブリッシュするためにリソースを有効にする

次の例では、`us-west-2` リージョンの ARN `my-queue-arn` と URL `my-queue-url` を持つキューに発行するために ARN `my-resource-arn` でリソースを有効にします。

```
require 'aws-sdk'

sqs = Aws::SQS::Client.new(region: 'us-west-2')

policy  = '{
  "Version":"2008-10-17",
  "Id":' + my-queue-arn + '/SQSDefaultPolicy",
  "Statement":[{
    "Sid":"__default_statement_ID",
    "Effect":"Allow",
    "Principal":{
      "AWS":"*"
    },
    "Action":["SQS:SendMessage"],
    "Resource":"' + my-queue-arn + '",
    "Condition":{
      "ArnEquals":{
        "AWS:SourceArn":"' + my-resource-arn + '"}
     }
  }]
}'

sqs.set_queue_attributes({
  queue_url: my-queue-url,
  attributes: {
    Policy: policy
  }
})
```

Amazon SQS でのデッドレターキューの操作

Amazon SQS では、デッドレターキューがサポートされます。デッドレターキューは、正常に処理できないメッセージの送信先として他の (送信元) キューが使用できるキューです。これらのメッセージは、処理が成功しなかった理由を判断するためにデッドレターキューに分離できます。デッドレターキューの詳細については、「Amazon SQS デッドレターキューの使用」を参照してください。

この例では、AWS SDK for Ruby を Amazon SQS で使用して、以下のことを行います。

1. Aws::SQS::Client#create_queue を使用して、デッドレターキューを表すキューを作成する。
2. Aws::SQS::Client#set_queue_attributes を使用して、デッドレターキューを既存のキューに関連付ける。
3. Aws::SQS::Client#send_message を使用して、既存のキューにメッセージを送信する。
4. Aws::SQS::QueuePoller を使用してキューをポーリングする。
5. Aws::SQS::Client#receive_message を使用してデッドレターキューでメッセージを受信する。

前提条件

コード例を実行する前に、以下で説明されているように、AWS SDK for Ruby をインストールし、設定する必要があります。

- AWS SDK for Ruby のインストール (p. 4)
- AWS SDK for Ruby の設定 (p. 5)

AWS マネジメントコンソールを使用して、既存のキュー my-queue を作成する必要もあります。

注意: 単純化するために、以下のコード例では Aws::SQS::Client#add_permission について示しません。実際のシナリオでは、SendMessage、ReceiveMessage、DeleteMessage、DeleteQueue などのアクション

へのアクセスは常に制限する必要があります。そうしないと、情報漏洩、サービス拒否、またはキューへのメッセージのインジェクションが発生する可能性があります。

例

```
require 'aws-sdk'

# Uncomment for Windows.
# Aws.use_bundled_cert!

sqs = Aws::SQS::Client.new(region: 'us-east-1')

# Create a queue representing a dead letter queue.
dead_letter_queue_name = "dead-letter-queue"

sqs.create_queue({
  queue_name: dead_letter_queue_name
})

# Get the dead letter queue's URL and ARN, so that you can associate it with an existing
 queue.
dead_letter_queue_url = sqs.get_queue_url(queue_name: dead_letter_queue_name).queue_url

dead_letter_queue_arn = sqs.get_queue_attributes({
  queue_url: dead_letter_queue_url,
  attribute_names: ["QueueArn"]
}).attributes["QueueArn"]

# Associate the dead letter queue with an existing queue.
begin
  queue_name = "my-queue"
  queue_url = sqs.get_queue_url(queue_name: queue_name).queue_url

  # Use a redrive policy to specify the dead letter queue and its behavior.
  redrive_policy = {
    "maxReceiveCount" => "5", # After the queue receives the same message 5 times, send
 that message to the dead letter queue.
    "deadLetterTargetArn" => dead_letter_queue_arn
  }.to_json

  sqs.set_queue_attributes({
    queue_url: queue_url,
    attributes: {
      "RedrivePolicy" => redrive_policy
    }
  })

rescue Aws::SQS::Errors::NonExistentQueue
  puts "A queue named '#{queue_name}' does not exist."
  exit(false)
end

# Send a message to the queue.
puts "Sending a message..."

sqs.send_message({
  queue_url: queue_url,
  message_body: "I hope I get moved to the dead letter queue."
})

30.downto(0) do |i|
  print "\rWaiting #{i} second(s) for sent message to be receivable..."
  sleep(1)
end
```

```
puts "\n"

poller = Aws::SQS::QueuePoller.new(queue_url)
# Receive 5 messages max and stop polling after 20 seconds of no received messages.
poller.poll(max_number_of_messages:5, idle_timeout: 20) do |messages|
  messages.each do |msg|
    puts "Received message ID: #{msg.message_id}"
  end
end

# Check to see if Amazon SQS moved the message to the dead letter queue.
receive_message_result = sqs.receive_message({
  queue_url: dead_letter_queue_url,
  max_number_of_messages: 1
})

if receive_message_result.messages.count > 0
  puts "\n#{receive_message_result.messages[0].body}"
else
  puts "\nNo messages received."
end
```

Amazon SQS でメッセージの可視性タイムアウトを指定する

Amazon SQS で、メッセージが受信された直後は、メッセージはキューに残ったままです。他のコンシューマーが再度メッセージを処理しないようにするため、Amazon SQS は可視性タイムアウトを設定します。これは、Amazon SQS によって他の消費コンポーネントがそのメッセージを受信および処理できなくなる期間です。詳細については、「可視性タイムアウト」を参照してください。

この例では、AWS SDK for Ruby を Amazon SQS で使用して、以下のことを行います。

1. Aws::SQS::Client#get_queue_url を使用して、既存のキューの URL を取得する。
2. Aws::SQS::Client#receive_message を使用して最大 10 件のメッセージを受信する。
3. Aws::SQS::Client#change_message_visibility を使用して、受信後にメッセージが表示されない期間を指定する。

前提条件

コード例を実行する前に、以下で説明されているように、AWS SDK for Ruby をインストールし、設定する必要があります。

- AWS SDK for Ruby のインストール (p. 4)
- AWS SDK for Ruby の設定 (p. 5)

また、キュー my-queue を作成する必要があります。これは Amazon SQS コンソールで行うことができます。

例

```
require 'aws-sdk'

sqs = Aws::SQS::Client.new(region: 'us-east-1')

begin
```

```
  queue_name = "my-queue"
  queue_url = sqs.get_queue_url(queue_name: queue_name).queue_url

  receive_message_result_before = sqs.receive_message({
    queue_url: queue_url,
    max_number_of_messages: 10 # Receive up to 10 messages, if there are that many.
  })

  puts "Before attempting to change message visibility timeout: received
 #{receive_message_result_before.messages.count} message(s)."

  receive_message_result_before.messages.each do |message|
    sqs.change_message_visibility({
      queue_url: queue_url,
      receipt_handle: message.receipt_handle,
      visibility_timeout: 36000 # This message will not be visible for 10 hours (10 hours *
 60 minutes * 60 seconds) after first receipt.
    })
  end

  # Try to retrieve the original messages after setting their visibility timeout.
  receive_message_result_after = sqs.receive_message({
    queue_url: queue_url,
    max_number_of_messages: 10
  })

  puts "\nAfter attempting to change message visibility timeout: received
 #{receive_message_result_after.messages.count} message(s)."

rescue Aws::SQS::Errors::NonExistentQueue
  puts "Cannot receive messages for a queue named '#{receive_queue_name}', as it does not
 exist."
end
```

AWS SDK for Ruby のヒントやコツ

このセクションでは、AWS のサービスで AWS SDK for Ruby を使用する上でのヒントやコツを提供します。

トピック

Amazon EC2 のヒントとコツ

このセクションでは、AWS SDK for Ruby と Amazon Elastic Compute Cloud(Amazon EC2) サービスに関するいくつかのヒントを示します。Amazon EC2 の詳細については、「Amazon EC2 入門ガイド」を参照してください。

Elastic IP の切り替え

次の例では、Elastic IP アドレスを `i-12345678` で表されるインスタンスと関連付けます。

```
ec2 = Aws::EC2::Client.new

resp = ec2.allocate_address
ec2.associate_address(instance_id:"i-12345678", allocation_id: resp.allocation_id)
```

ドキュメント履歴

AWS SDK for Ruby およびそのドキュメントに対する変更のリストを表示するには、GitHub の `aws/aws-sdk-ruby` リポジトリの CHANGELOG.md ファイルを参照してください。