Guia do desenvolvedor do SDK v2

AWS SDK for JavaScript

AWS SDK for JavaScript: Guia do desenvolvedor do SDK v2

Table of Contents

..... ix
O que é a AWS SDK for JavaScript? 1
Manutenção e suporte para as versões principais do SDK 1
Usar o SDK com o Node.js 2
Uso do SDK com o AWS Cloud9 2
Usar o SDK com o AWS Amplify 2
Usar o SDK com navegadores da web 2
Casos de uso comuns 3
Sobre os exemplos 3
Conceitos básicos 4
Conceitos básicos de um script de navegador 4
O cenário 4
Etapa 1: Criar um banco de identidades do Amazon Cognito 5
Etapa 2: Adicionar uma política ao perfil do IAM criado 6
Etapa 3: Criar a página HTML 7
Etapa 4: Escrever o script de navegador 8
Etapa 5: Executar o exemplo 10
Exemplo completo 10
Melhorias possíveis 11
Conceitos básicos de Node.js 12
O cenário 12
Tarefas de pré-requisito 12
Etapa 1: Instalar o SDK e as dependências 13
Etapa 2: Configurar as credenciais 13
Etapa 3: Criar o JSON do pacote para o projeto 14
Etapa 4: Escrever o código Node.js 15
Etapa 5: Executar o exemplo 16
Como usar o AWS Cloud9 com o SDK para JavaScript 17
Etapa 1: Configurar a conta da AWS para usar o AWS Cloud9 17
Etapa 2: Configurar o ambiente de desenvolvimento do AWS Cloud9 18
Etapa 3: Configurar o SDK para JavaScript 18
Para instalar o SDK para JavaScript para Node.js 18
Para configurar o SDK para JavaScript no navegador 19
Etapa 4: Fazer download do código de exemplo 19

Etapa 5: Executar e depurar o código de exemplo ... 20
Configuração do SDK para JavaScript ... 21
Pré-requisitos ... 21
Configurar um ambiente Node.js da AWS ... 22
Navegadores compatíveis ... 22
Instalação do SDK ... 23
Instalar usando o Bower ... 24
Carregar o SDK ... 24
Fazer upgrade da versão 1 ... 25
Conversão automática dos tipos Base64 e Timestamp Types na entrada/saída ... 26
response.data.RequestId transferido para response.requestId ... 27
Elementos expostos do Wrapper ... 27
Propriedades do cliente eliminadas ... 32
Configuração do SDK para JavaScript ... 33
Usar o objeto de configuração global ... 33
Definir configuração global ... 34
Definir configuração por serviço ... 36
Dados de configuração imutáveis ... 36
Definir a região da AWS ... 37
Em um construtor de classes do cliente ... 37
Usar o objeto de configuração global ... 37
Usar uma variável de ambiente ... 37
Usar um arquivo de configuração compartilhado ... 37
Ordem de precedência para definir a região ... 38
Especificar endpoints personalizados ... 39
Formato do string de endpoint ... 39
Endpoints da região ap-northeast-3 ... 39
Endpoints para MediaConvert ... 39
Autenticação do SDK com AWS ... 40
Iniciar uma sessão do portal de acesso AWS ... 41
Mais informações de autenticação ... 42
Definir credenciais ... 43
Melhores práticas para credenciais ... 43
Definir credenciais em Node.js ... 44
Definir credenciais em um navegador da Web ... 49
Bloquear versões de API ... 59

Obter versões de API ... 60
Considerações sobre Node.js ... 60
Usar módulos de Node.js integrados ... 60
Usar pacotes NPM ... 61
Configurar maxSockets em Node.js ... 61
Reutilizar conexões com Keep-alive no Node.js ... 62
Configurar proxies para Node.js ... 64
Registrar pacotes de certificados em Node.js ... 64
Considerações sobre o script de navegador ... 65
Criar o SDK para navegadores ... 65
Cross-Origin Resource Sharing (CORS, Compartilhamento de recursos de origem cruzada) ... 68
Empacotamento com o Webpack ... 73
Instalar o Webpack ... 73
Configurar o Webpack ... 74
Executar o Webpack ... 75
Usar o pacote Webpack ... 76
Importar serviços individuais ... 76
Empacotamento para o Node.js ... 77
Trabalhar com os serviços da ... 79
Criar e chamar objetos de serviço ... 80
Exigir serviços individuais ... 81
Criar objetos de serviço ... 82
Bloquear a versão da API de um objeto de serviço ... 83
Especificar os parâmetros do objeto de serviço ... 83
Registrar em log as chamadas a AWS SDK for JavaScript ... 84
Usar um registrador de terceiros ... 84
Chamar serviços assincronamente ... 85
Gerenciar chamadas assíncronas ... 85
Usar a função de retorno de chamada ... 87
Usar um listener de evento do objeto de solicitação ... 88
Usar async/await ... 93
Usar promessas ... 94
Usar o objeto de resposta ... 97
Acessar dados retornados no objeto de resposta ... 97
Paginar pelos dados retornados ... 98

Acessar informações de erro de um objeto de resposta 99
Acessar o objeto de solicitação de origem 99
Trabalhar com o JSON 99
JSON como parâmetros do objeto de serviço 100
Retornar dados como JSON 101
Exemplos de código do SDK para JavaScript 103
Exemplos do Amazon CloudWatch 103
Criação de alarmes no Amazon CloudWatch 104
Usar ações de alarmes no Amazon CloudWatch 108
Obter métricas do Amazon CloudWatch 112
Enviar eventos para o Amazon CloudWatch Events 115
Uso de filtros de assinatura do Amazon CloudWatch Logs 121
Exemplos do Amazon DynamoDB 125
Criação e uso de tabelas no DynamoDB 126
Ler e gravar um único item no DynamoDB 131
Ler e gravar itens em lote no DynamoDB 135
Consultar e verificar uma tabela do DynamoDB 138
Uso do cliente de documentos do DynamoDB 141
Exemplos do Amazon EC2 148
Criar uma instância do Amazon EC2 148
Gerenciar instâncias do Amazon EC2 151
Trabalhar com pares de chaves do Amazon EC2 157
Uso de regiões e zonas de disponibilidade para o Amazon EC2 161
Trabalhar com grupos de segurança no Amazon EC2 163
Usar endereços IP elásticos no Amazon EC2 167
Exemplos do MediaConvert 172
Obtendo seu endpoint específico da região 172
Criar e gerenciar tarefas 174
Usar modelos de tarefa 182
Exemplos de Amazon S3 Glacier 190
Criação de um S3 Glacier Vault 191
Fazer upload de um arquivamento para o S3 Glacier 192
Fazer um multipart upload para o S3 Glacier 193
Exemplos do IAM AWS 195
Gerenciar usuários do IAM 196
Trabalhar com políticas do IAM 201

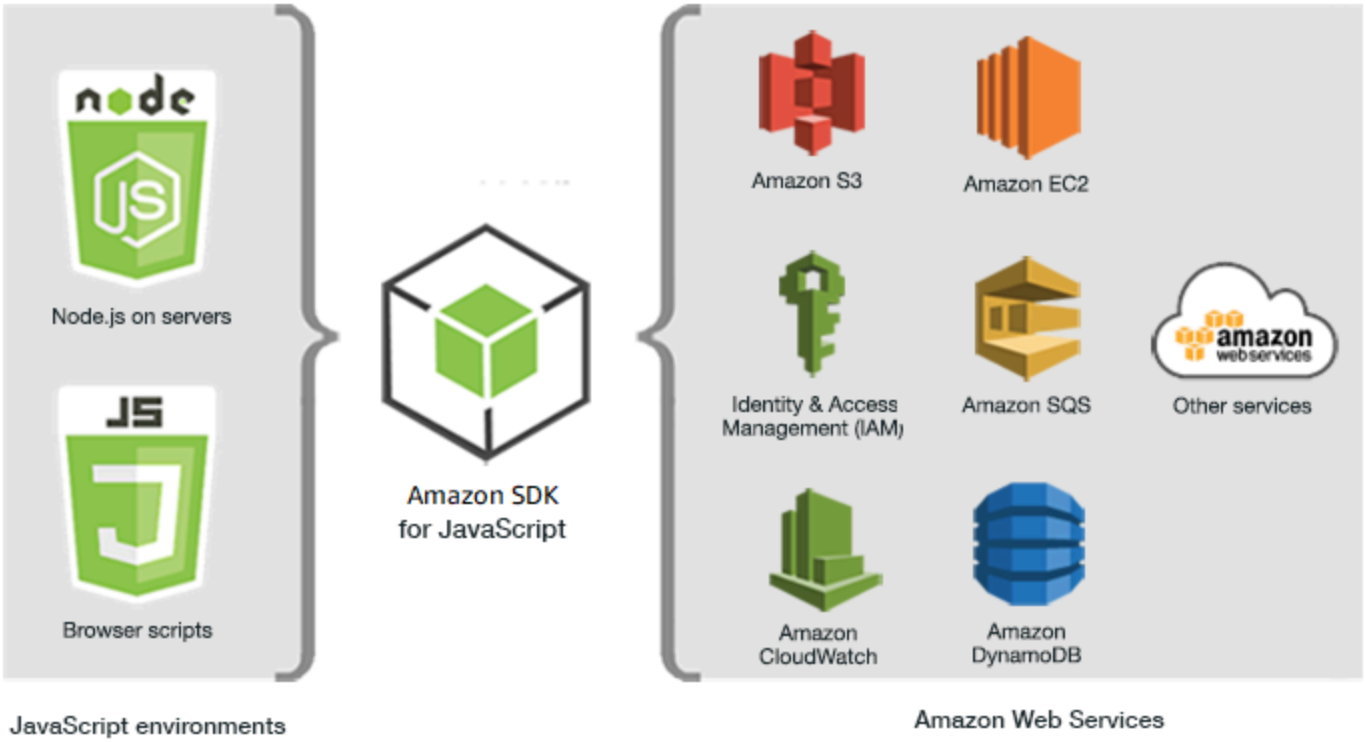
Gerenciar chaves de acesso do IAM ... 207
Trabalhar com certificados de servidor do IAM ... 212
Gerenciar aliases de conta do IAM ... 216
Exemplos do Amazon Kinesis ... 220
Captura do progresso de rolagem da página da web com Amazon Kinesis ... 220
Exemplos do Amazon S3 ... 227
Exemplos de navegador Amazon S3 ... 228
Exemplos de Node.js do Amazon S3 ... 257
Exemplos do Amazon SES ... 278
Gerenciar identidades ... 279
Lidar com modelos de e-mail ... 284
Envio de e-mail usando o Amazon SES ... 290
Usar filtros de endereço IP ... 296
Uso de regras de recebimento ... 301
Exemplos do Amazon SNS ... 306
Gerenciamento de tópicos ... 307
Publicar mensagens em um tópico ... 313
Gerenciamento de inscrições ... 315
Envio de mensagens SMS ... 321
Exemplos do Amazon SQS ... 328
Uso de filas no Amazon SQS ... 329
Enviar e receber mensagens no Amazon SQS ... 333
Gerenciar o tempo limite de visibilidade no Amazon SQS ... 337
Habilitar a sondagem longa no Amazon SQS ... 339
Usar dead letter queues no Amazon SQS ... 343
Tutoriais ... 346
Tutorial: Configuração do Node.js em uma instância do Amazon EC2 ... 346
Pré-requisitos ... 346
Procedimento ... 346
Criar uma imagem de máquina da Amazon ... 348
Recursos relacionados ... 348
Referência de API e Changelog ... 349
Changelog do SDK no GitHub ... 349
Segurança ... 350
Proteção de dados ... 350
Identity and Access Management ... 352

Público ... 352
Autenticando com identidades ... 353
Gerenciando acesso usando políticas ... 356
Como Serviços da AWS funcionam com o IAM ... 359
Solução de problemas de identidade e acesso do AWS ... 359
Compliance Validation ... 361
Resilience ... 363
Infrastructure Security ... 363
Aplicar uma versão mínima do TLS ... 364
Verificar e impor o TLS no Node.js ... 365
Verificar e impor o TLS em um script de navegador ... 367
Recursos adicionais ... 370
Guia de referência de AWS SDKs e ferramentas ... 370
Fórum do SDK do JavaScript ... 370
SDK do JavaScript e Guia do desenvolvedor no GitHub ... 370
SDK do JavaScript no Gitter ... 370
Histórico do documento ... 371
Histórico do documento ... 371
Atualizações anteriores ... 372

Anunciamos o próximo fim do suporte para o AWS SDK for JavaScript v2. Recomendamos migrar para o AWS SDK for JavaScript v3. Para saber as datas e receber detalhes adicionais e informações sobre como migrar, consulte o anúncio vinculado.

O que é a AWS SDK for JavaScript?

O AWS SDK for JavaScript fornece uma API JavaScript para serviços da AWS. Você pode usar a API JavaScript para criar bibliotecas ou aplicativos para Node.js ou o navegador.



Nem todos os serviços estão disponíveis imediatamente no SDK. Para descobrir quais serviços atualmente são compatíveis com o AWS SDK for JavaScript, consulte https://github.com/aws/aws-sdk-js/blob/master/SERVICES.md. Para obter mais informações sobre o SDK para Java no GitHub, consulte Recursos adicionais.

Manutenção e suporte para as versões principais do SDK

Para obter informações sobre manutenção e suporte para versões principais do SDK e suas dependências subjacentes, consulte o seguinte no Guia de referência de AWS SDKs e ferramentas:

- Política de manutenção de ferramentas e SDKs da AWS
- Matriz de suporte a versões de ferramentas e SDKs da AWS

Usar o SDK com o Node.js

Node.js é um tempo de execução de plataforma cruzada para a execução de aplicativos em JavaScript no lado do servidor. Você pode configurar o Node.js em uma instância do Amazon EC2 rodar em um servidor. Você também pode usar o Node.js para gravar funções do AWS Lambda sob demanda.

O uso do SDK para Node.js é diferente da maneira como você o usa para JavaScript em um navegador da web. A diferença refere-se à maneira como você carrega o SDK e obtém as credenciais necessárias para acessar serviços da web específicos. Quando o uso de determinadas APIs diferir entre o Node.js e o navegador, essas diferenças serão destacadas.

Uso do SDK com o AWS Cloud9

Você também pode desenvolver aplicativos de Node.js usando o SDK para JavaScript no IDE do AWS Cloud9. Para obter um exemplo de como usar o AWS Cloud9 no desenvolvimento de Node.js, consulte Exemplo de Node.js para o AWS Cloud9 no Guia do usuário do AWS Cloud9. Para obter mais informações sobre usar o AWS Cloud9 com SDK para JavaScript, consulte Como usar o AWS Cloud9 com o AWS SDK for JavaScript.

Usar o SDK com o AWS Amplify

Para aplicativos da web baseados em navegador, de dispositivos móveis e híbridos, você também pode usar a Biblioteca AWS Amplify no GitHub da AWS, que estende o SDK para JavaScript, fornecendo uma interface declarativa.

> **Note**
>
> Talvez as estruturas, como o AWS Amplify, não ofereçam o mesmo suporte a navegadores que o SDK para JavaScript. Verifique a documentação de uma estrutura para obter detalhes.

Usar o SDK com navegadores da web

Todos os principais navegadores são compatíveis com a execução de JavaScript. O código JavaScript em execução em um navegador da web normalmente é chamado de JavaScript no lado do cliente.

O uso do SDK para JavaScript em um navegador da web é diferente da maneira como você o usa para Node.js. A diferença refere-se à maneira como você carrega o SDK e obtém as credenciais necessárias para acessar serviços da web específicos. Quando o uso de determinadas APIs diferir entre o Node.js e o navegador, essas diferenças serão destacadas.

Para obter uma lista dos navegadores compatíveis com o AWS SDK for JavaScript, consulte Navegadores compatíveis.

Casos de uso comuns

Usar o SDK para JavaScript nos scripts do navegador possibilita realizar uma série de casos de uso irrefutáveis. Veja a seguir algumas ideias para itens que você pode criar em um aplicativo de navegador usando o SDK para JavaScript para acessar vários serviços Web.

- Crie um console personalizado para serviços da AWS no qual você acessa e combina atributos entre regiões e serviços para melhor atender às necessidades da organização ou do projeto.
- Use o Amazon Cognito Identity para habilitar o acesso do usuário autenticado aos aplicativos de navegador e sites, incluindo o uso de autenticação de terceiros pelo Facebook e outros.
- Use o Amazon Kinesis para processar clickstreams ou outros dados de marketing em tempo real.
- Use o Amazon DynamoDB para persistência de dados sem servidor, como preferências de usuários individuais quanto a visitantes do site ou usuários de aplicativos.
- Use o AWS Lambda para encapsular a lógica proprietária que você pode invocar pelos scripts do navegador sem fazer download dos scripts e revelar sua propriedade intelectual aos usuários.

Sobre os exemplos

Você pode procurar exemplos de JavaScript no SDK na Biblioteca de exemplos de código da AWS.

Conceitos básicos do AWS SDK for JavaScript

O AWS SDK for JavaScript oferece acesso a serviços da web nos scripts do navegador ou em Node.js. Esta seção tem dois exercícios de conceitos básicos que mostram como trabalhar com o SDK para JavaScript em cada um desses ambientes do JavaScript.

Você também pode desenvolver aplicativos de Node.js usando o SDK para JavaScript no IDE do AWS Cloud9. Para obter um exemplo de como usar o AWS Cloud9 no desenvolvimento de Node.js, consulte Exemplo de Node.js para o AWS Cloud9 no Guia do usuário do AWS Cloud9.

Tópicos

- Conceitos básicos de um script de navegador
- Conceitos básicos de Node.js

Conceitos básicos de um script de navegador

Este exemplo de script de navegador mostra:

- Como acessar serviços da AWS a partir de um script de navegador usando o Amazon Cognito.
- Como transformar texto em fala sintetizada usando o .
- Como usar um objeto presigner para criar um pre-signed URL.

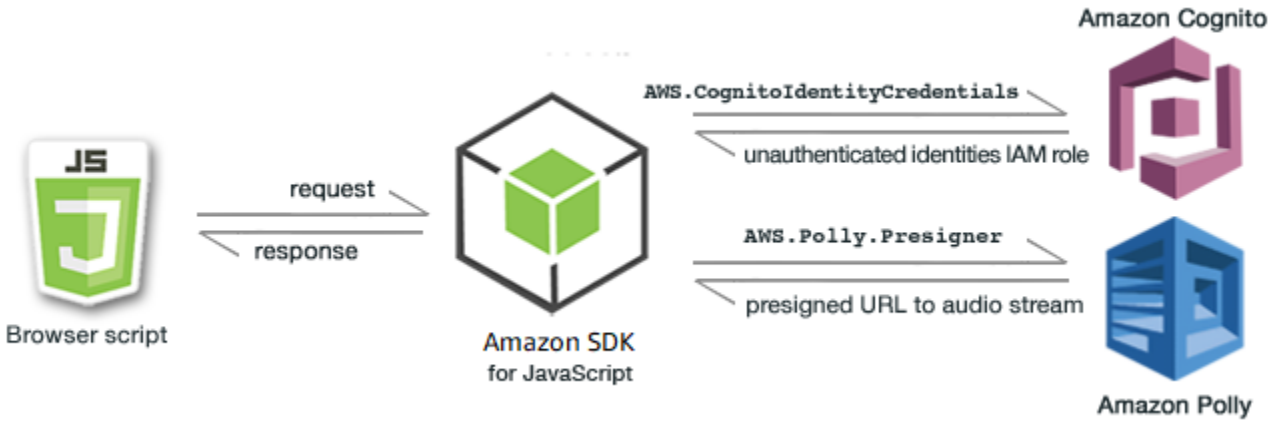
O cenário

O Amazon Polly é um serviço na nuvem que converte texto em fala realista. Você pode usar o Amazon Polly para desenvolver aplicações que aumentam o envolvimento e a acessibilidade. O Amazon Polly oferece suporte a vários idiomas e inclui uma variedade de vozes realistas. Para obter mais informações sobre o Amazon Polly, consulte o Guia do desenvolvedor do Amazon Polly.

O exemplo mostra como configurar e executar um script de navegador simples que pega o texto digitado, envia para o Amazon Polly e retorna o URL do áudio sintetizado do texto para você reproduzir. O script do navegador usa o Amazon Cognito Identity para fornecer as credenciais

necessárias para acessar os serviços da AWS. Você verá os padrões básicos para carregar e usar o SDK para JavaScript em scripts do navegador.

> **Note**
>
> A reprodução da fala sintetizada neste exemplo depende da execução em um navegador compatível com áudio HTML 5.



O script de navegador usa o SDK para JavaScript para sintetizar texto usando estas APIs:

- Construtor `AWS.CognitoIdentityCredentials`
- Construtor `AWS.Polly.Presigner`
- `getSynthesizeSpeechUrl`

Etapa 1: Criar um banco de identidades do Amazon Cognito

Neste exercício, você cria e usa um banco de identidades do Amazon Cognito para fornecer acesso não autenticado ao script de navegador do serviço Amazon Polly. Criar um grupo de identidades também cria duas funções do IAM, uma para oferecer suporte aos usuários autenticados por um provedor de identidades e outra para oferecer suporte a usuários convidados não autenticados.

Neste exercício, vamos trabalhar apenas com a função de usuário não autenticado para manter o enfoque na tarefa. Você poderá integrar o suporte para um provedor de identidade e os usuários autenticados depois. Para ter mais informações, consulte Grupo de identidades do Amazon Cognito no Guia do desenvolvedor do Amazon Cognito.

Como criar um banco de identidades do Amazon Cognito

1. Faça login no AWS Management Console e abra o console do Amazon Cognito em https://console.aws.amazon.com/cognito/.
2. No painel de navegação à esquerda, escolha Bancos de identidades.
3. Selecione Criar banco de identidades.
4. Em Configurar confiança do grupo de identidades, escolha Acesso de convidado para autenticação do usuário.
5. Em Configurar permissões, escolha Criar um novo perfil do IAM e insira um nome (por exemplo, getStartedRole) no nome do perfil do IAM.
6. Em Configurar propriedades, insira um nome (por exemplo, getStartedPool) em Nome do grupo de identidades.
7. Em Revisar e criar, confirme as seleções que você fez para o novo banco de identidades. Selecione Editar para retornar ao assistente e alterar as configurações. Quando terminar, selecione Criar banco de identidades.
8. Observe o ID do grupo de identidades e a Região do banco de identidades recém-criado do Amazon Cognito. Você precisa desses valores para substituir *IDENTITY_POOL_ID* d *REGION* no Etapa 4: Escrever o script de navegador.

Depois de criar o grupo de identidades do Amazon Cognito, você estará pronto para adicionar permissões do Amazon Polly necessárias para o script de navegador.

Etapa 2: Adicionar uma política ao perfil do IAM criado

Para habilitar o acesso do script de navegador para o Amazon Polly à síntese de fala, use o perfil do IAM não autenticado criado para o grupo de identidades do Amazon Cognito. Isso exige que você adicione uma política do IAM à função. Para obter mais informações sobre modificar os perfis do IAM, consulte Modificação de uma política de perrmissões de perfil no Guia do usuário do IAM.

Para adicionar uma política do Amazon Polly ao perfil do IAM associada a usuários não autenticados

1. Faça login no AWS Management Console e abra o console do IAM em https://console.aws.amazon.com/iam/.
2. No painel de navegação à esquerda, escolha Roles.
3. Escolha o nome do perfil que você deseja modificar (por exemplo, getStartedRole) e escolha a guia Permissões.

4. Escolha Adicionar permissões e depois Anexar políticas.
5. Na página Adicionar permissões desse perfil, encontre e marque a caixa de seleção de AmazonPollyReadOnly.

> **Note**
>
> Você pode usar esse processo para habilitar o acesso a qualquer serviço da AWS.

6. Escolha Add permissions (Adicionar permissões).

Depois de criar o banco de identidades do Amazon Polly e adicionar permissões do Amazon Polly ao perfil do IAM para usuários não autenticados, você estará pronto para criar a página da web e o script de navegador.

Etapa 3: Criar a página HTML

O aplicativo de exemplo consiste em uma única página HTML que contém a interface do usuário e o script de navegador. Para começar, crie um documento HTML e copie o conteúdo a seguir. A página inclui um campo de entrada e um botão, um elemento `<audio>` para reproduzir a fala sintetizada, e um elemento <p> para exibir mensagens. (O exemplo completo é mostrado na parte inferior desta página.)

Para obter mais informações sobre o elemento `<audio>`, consulte o áudio.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>AWS SDK for JavaScript - Browser Getting Started Application</title>
  </head>

  <body>
    <div id="textToSynth">
      <input autofocus size="23" type="text" id="textEntry" value="It's very good to
 meet you."/>
      <button class="btn default" onClick="speakText()">Synthesize</button>
      <p id="result">Enter text above then click Synthesize</p>
    </div>
    <audio id="audioPlayback" controls>
      <source id="audioSource" type="audio/mp3" src="">
```

```
    </audio>
    <!-- (script elements go here) -->
 </body>
</html>
```

Salve o arquivo HTML, nomeando-o como `polly.html`. Depois de criar a interface do usuário do aplicativo, você estará pronto para adicionar o código de script do navegador que executa o aplicativo.

Etapa 4: Escrever o script de navegador

A primeira coisa a fazer ao criar o script de navegador é incluir o SDK para JavaScript adicionando um elemento `<script>` depois do elemento `<audio>` na página: Para encontrar o SDK_VERSION_NUMBER atual, consulte a Referência da API para o SDK para JavaScript no Guia de referência da API do AWS SDK for JavaScript.

```
<script src="https://sdk.amazonaws.com/js/aws-sdk-SDK_VERSION_NUMBER.min.js"></script>
```

Adicione um novo elemento `<script type="text/javascript">` após a entrada do SDK. Você vai adicionar o script de navegador a esse elemento. Defina a região da AWS e as credenciais do SDK. Crie uma função chamada `speakText()` que será invocada como um manipulador de eventos pelo botão.

Para sintetizar a fala com o Amazon Polly, você deve fornecer uma variedade de parâmetros, inclusive o formato do som da saída, a taxa de amostragem, o ID da voz a ser usada e o texto a ser reproduzido. Ao criar inicialmente os parâmetros, defina o parâmetro `Text:` como uma string vazia; o parâmetro `Text:` será definido como o valor recuperado do elemento `<input>` na página da web. Substitua *IDENTITY_POOL_ID* e *REGION* no código a seguir pelos valores indicados em Etapa 1: Criar um banco de identidades do Amazon Cognito.

```
    <script type="text/javascript">

        // Initialize the Amazon Cognito credentials provider
        AWS.config.region = 'REGION';
        AWS.config.credentials = new AWS.CognitoIdentityCredentials({IdentityPoolId:
 'IDENTITY_POOL_ID'});

        // Function invoked by button click
        function speakText() {
```

```
            // Create the JSON parameters for getSynthesizeSpeechUrl
            var speechParams = {
                OutputFormat: "mp3",
                SampleRate: "16000",
                Text: "",
                TextType: "text",
                VoiceId: "Matthew"
            };
            speechParams.Text = document.getElementById("textEntry").value;
```

O Amazon Polly retorna a fala sintetizada como um fluxo de áudio. A maneira mais fácil de reproduzir o áudio em um navegador é fazer o Amazon Polly disponibilizar o áudio em um URL pré-assinado que você pode definir como o atributo `src` do elemento `<audio>` na página da web.

Crie um novo objeto de serviço `AWS.Polly`. Crie o objeto `AWS.Polly.Presigner` que você usará para criar o pre-signed URL a partir do qual o áudio da fala sintetizada poderá ser recuperado. Você deve passar os parâmetros de fala definidos, bem como o objeto de serviço `AWS.Polly` criado para o construtor `AWS.Polly.Presigner`.

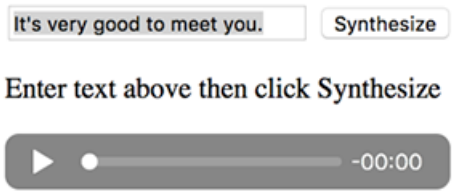
Depois de criar o objeto presigner, chame o método `getSynthesizeSpeechUrl` desse objeto, passando os parâmetros de fala. Se for bem-sucedido, esse método retornará o URL da fala sintetizada, atribuído ao elemento `<audio>` para reprodução.

```
            // Create the Polly service object and presigner object
            var polly = new AWS.Polly({apiVersion: '2016-06-10'});
            var signer = new AWS.Polly.Presigner(speechParams, polly)

            // Create presigned URL of synthesized speech file
            signer.getSynthesizeSpeechUrl(speechParams, function(error, url) {
            if (error) {
                document.getElementById('result').innerHTML = error;
            } else {
                document.getElementById('audioSource').src = url;
                document.getElementById('audioPlayback').load();
                document.getElementById('result').innerHTML = "Speech ready to play.";
            }
          });
        }
    </script>
```

Etapa 5: Executar o exemplo

Para executar o aplicativo de exemplo, carregue `polly.html` em um navegador da web. É assim que a apresentação do navegador deve ser.



Digite uma frase que você deseja transformar em uma fala na caixa de entrada e escolha Synthesize (Sintetizar). Quando o áudio está pronto para ser reproduzido, uma mensagem é exibida. Use os controles do player de áudio para ouvir a fala sintetizada.

Exemplo completo

Esta é a página HTML completa com o script de navegador. Ele também está disponível aqui no GitHub.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>AWS SDK for JavaScript - Browser Getting Started Application</title>
  </head>

  <body>
    <div id="textToSynth">
      <input autofocus size="23" type="text" id="textEntry" value="It's very good to
 meet you."/>
      <button class="btn default" onClick="speakText()">Synthesize</button>
      <p id="result">Enter text above then click Synthesize</p>
    </div>
    <audio id="audioPlayback" controls>
      <source id="audioSource" type="audio/mp3" src="">
    </audio>
    <script src="https://sdk.amazonaws.com/js/aws-sdk-2.410.0.min.js"></script>
    <script type="text/javascript">

        // Initialize the Amazon Cognito credentials provider
```

```
        AWS.config.region = 'REGION';
        AWS.config.credentials = new AWS.CognitoIdentityCredentials({IdentityPoolId:
 'IDENTITY_POOL_ID'});

        // Function invoked by button click
        function speakText() {
            // Create the JSON parameters for getSynthesizeSpeechUrl
            var speechParams = {
                OutputFormat: "mp3",
                SampleRate: "16000",
                Text: "",
                TextType: "text",
                VoiceId: "Matthew"
            };
            speechParams.Text = document.getElementById("textEntry").value;

            // Create the Polly service object and presigner object
            var polly = new AWS.Polly({apiVersion: '2016-06-10'});
            var signer = new AWS.Polly.Presigner(speechParams, polly)

            // Create presigned URL of synthesized speech file
            signer.getSynthesizeSpeechUrl(speechParams, function(error, url) {
            if (error) {
                document.getElementById('result').innerHTML = error;
            } else {
                document.getElementById('audioSource').src = url;
                document.getElementById('audioPlayback').load();
                document.getElementById('result').innerHTML = "Speech ready to play.";
            }
          });
        }
    </script>
  </body>
</html>
```

Melhorias possíveis

Aqui estão variações desse aplicativo que você pode usar para explorar ainda mais o uso do SDK para JavaScript em um script de navegador.

- Experimente usando outros formatos de saída de som.
- Adicione a opção para selecionar qualquer uma das diversas vozes fornecidas pelo Amazon Polly.

- Integre um provedor de identidade como Facebook ou Amazon a ser usado com a função do IAM autenticada.

Conceitos básicos de Node.js

Este exemplo de código Node.js mostra:

- Como criar o manifesto `package.json` para o projeto.
- Como instalar e incluir os módulos usados pelo projeto.
- Como criar um objeto de serviço Amazon Simple Storage Service (Amazon S3) da classe de cliente `AWS.S3`.
- Como criar um bucket do Amazon S3 e fazer upload de um objeto para esse bucket.

O cenário

O exemplo mostra como configurar e executar um módulo Node.js simples que cria um bucket do Amazon S3 e adiciona um objeto de texto a ele.

Como nomes de bucket no Amazon S3 devem ser exclusivos globalmente, este exemplo inclui um módulo Node.js de terceiros que gera um valor de ID exclusivo que você pode incorporar ao nome do bucket. Este módulo adicional se chama `uuid`.

Tarefas de pré-requisito

Para configurar e executar este exemplo, você deve primeiro concluir estas tarefas:

- Crie um diretório de trabalho para desenvolver o módulo Node.js. Nomeie este diretório `awsnodesample`. Observe que o diretório deve ser criado em um local que possa ser atualizado por aplicativos. Por exemplo, no Windows, não crie o diretório em "C:\Arquivos de Programas".
- Instale o Node.js. Para obter mais informações, consulte o website Node.js. Você pode encontrar downloads das versões atuais e LTS do Node.js para uma grande variedade de sistemas operacionais em https://nodejs.org/en/download/current/.

Sumário

- Etapa 1: Instalar o SDK e as dependências
- Etapa 2: Configurar as credenciais
- Etapa 3: Criar o JSON do pacote para o projeto
- Etapa 4: Escrever o código Node.js
- Etapa 5: Executar o exemplo

Etapa 1: Instalar o SDK e as dependências

Você instala o pacote SDK para JavaScript usando npm (o gerenciador de pacotes Node.js).

No diretório `awsnodesample` no pacote, digite o seguinte na linha de comando.

```
npm install aws-sdk
```

Esse comando instala o SDK para JavaScript no projeto e atualiza `package.json` para listar o SDK como uma dependência de projeto. Você pode encontrar informações sobre esse pacote procurando "aws-sdk" no site do npm.

Instale o módulo `uuid` no projeto digitando o seguinte na linha de comando, que instala o módulo e atualiza `package.json`. Para obter mais informações sobre `uuid`, consulte a página do módulo em https://www.npmjs.com/package/uuid.

```
npm install uuid
```

Esses pacotes e os códigos associados são instalados no subdiretório `node_modules` do projeto.

Para obter mais informações sobre como instalar pacotes Node.js, consulte Downloading and installing packages locally (Fazer download e instalar pacotes localmente) e Creating Node.js modules (Criar módulos Node.js) no site do npm (gerenciador de pacotes Node.js). Para obter informações sobre como baixar e instalar o AWS SDK for JavaScript, consulte Instalação do SDK for JavaScript.

Etapa 2: Configurar as credenciais

Você precisa fornecer credenciais para a AWS, de maneira que apenas a sua conta e os seus recursos sejam acessados pelo SDK. Para obter mais informações sobre como obter as credenciais da conta, consulte Autenticação do SDK com AWS.

Para armazenar essas informações, recomendamos criar um arquivo de credenciais compartilhadas. Para saber como, consulte Carregar credenciais em Node.js do arquivo de credenciais compartilhado. O arquivo de credenciais deve ser semelhante ao exemplo a seguir.

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

É possível determinar se você definiu as credenciais corretamente executando o seguinte código com Node.js:

```
var AWS = require("aws-sdk");

AWS.config.getCredentials(function(err) {
  if (err) console.log(err.stack);
  // credentials not loaded
  else {
    console.log("Access key:", AWS.config.credentials.accessKeyId);
  }
});
```

Da mesma maneira, se você tiver definido a região corretamente no arquivo `config`, poderá exibir esse valor definindo a variável de ambiente `AWS_SDK_LOAD_CONFIG` como qualquer valor e usando o seguinte código:

```
var AWS = require("aws-sdk");

console.log("Region: ", AWS.config.region);
```

Etapa 3: Criar o JSON do pacote para o projeto

Depois de criar o diretório do projeto `awsnodesample`, você vai criar e adicionar um arquivo `package.json` para manter os metadados do seu projeto Node.js. Para obter detalhes sobre como usar o `package.json` em um projeto Node.js, consulte Como criar im arquivo package.json.

No diretório de projeto, crie um novo arquivo chamado `package.json`. Adicione esse JSON ao arquivo.

```
{
  "dependencies": {},
```

```
  "name": "aws-nodejs-sample",
  "description": "A simple Node.js application illustrating usage of the SDK for
 JavaScript.",
  "version": "1.0.1",
  "main": "sample.js",
  "devDependencies": {},
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "NAME",
  "license": "ISC"
}
```

Salve o arquivo. À medida que você instalar os módulos de que você precisar, a parte `dependencies` do arquivo será concluída. Você pode encontrar um arquivo JSON que mostra um exemplo dessas dependências aqui no GitHub.

Etapa 4: Escrever o código Node.js

Crie um novo arquivo chamado `sample.js` para conter o código de exemplo. Comece adicionando as chamadas de função `require` para incluir o SDK para JavaScript e os módulos `uuid` para que eles estejam disponíveis para uso.

Crie um nome do bucket exclusivo que será usado para criar um bucket do Amazon S3 anexando um valor de ID exclusivo a um prefixo reconhecível, neste caso `'node-sdk-sample-'`. Você gera o ID exclusivo chamando o módulo `uuid`. Crie um nome para o parâmetro `Key` usado para fazer upload de um objeto no bucket.

Crie um objeto `promise` para chamar o método `createBucket` do objeto de serviço `AWS.S3`. Em uma resposta bem-sucedida, crie os parâmetros necessários para fazer upload de texto no bucket recém-criado. Usando outra promessa, chame o método `putObject` para fazer upload do objeto de texto no bucket.

```
// Load the SDK and UUID
var AWS = require("aws-sdk");
var uuid = require("uuid");

// Create unique bucket name
var bucketName = "node-sdk-sample-" + uuid.v4();
// Create name for uploaded object key
var keyName = "hello_world.txt";
```

```
// Create a promise on S3 service object
var bucketPromise = new AWS.S3({ apiVersion: "2006-03-01" })
  .createBucket({ Bucket: bucketName })
  .promise();

// Handle promise fulfilled/rejected states
bucketPromise
  .then(function (data) {
    // Create params for putObject call
    var objectParams = {
      Bucket: bucketName,
      Key: keyName,
      Body: "Hello World!",
    };
    // Create object upload promise
    var uploadPromise = new AWS.S3({ apiVersion: "2006-03-01" })
      .putObject(objectParams)
      .promise();
    uploadPromise.then(function (data) {
      console.log(
        "Successfully uploaded data to " + bucketName + "/" + keyName
      );
    });
  })
  .catch(function (err) {
    console.error(err, err.stack);
  });
```

Este código de exemplo pode ser encontrado aqui no GitHub.

Etapa 5: Executar o exemplo

Digite o seguinte comando para executar o exemplo.

```
node sample.js
```

Se o upload for bem-sucedido, você verá uma mensagem de confirmação na linha de comando. Você também pode encontrar o bucket e o objeto de texto carregado no console do Amazon S3.

Como usar o AWS Cloud9 com o AWS SDK for JavaScript

Use o AWS Cloud9 com o AWS SDK for JavaScript para gravar e executar o JavaScript no código do navegador, bem como gravar, executar e depurar o código Node.js, usando apenas um navegador. O AWS Cloud9 inclui ferramentas, como um editor de códigos e um terminal, além de um depurador do código Node.js. Como o AWS Cloud9 IDE é baseado na nuvem, você pode trabalhar em seus projetos no escritório, em casa ou em qualquer lugar usando um computador conectado à Internet. Para obter informações gerais sobre o AWS Cloud9, consulte o Guia do usuário do AWS Cloud9.

Siga estas etapas para configurar o AWS Cloud9 com o SDK para JavaScript:

Sumário

- Etapa 1: Configurar a conta da AWS para usar o AWS Cloud9
- Etapa 2: Configurar o ambiente de desenvolvimento do AWS Cloud9
- Etapa 3: Configurar o SDK para JavaScript
  - Para instalar o SDK para JavaScript para Node.js
  - Para configurar o SDK para JavaScript no navegador
- Etapa 4: Fazer download do código de exemplo
- Etapa 5: Executar e depurar o código de exemplo

Etapa 1: Configurar a conta da AWS para usar o AWS Cloud9

Comece a usar o AWS Cloud9 fazendo login no console do AWS Cloud9 como uma entidade do AWS Identity and Access Management (IAM) (por exemplo, um usuário do IAM) que tem permissões de acesso para o AWS Cloud9 na conta da AWS.

Para configurar uma entidade do IAM na sua conta da AWS para acessar o AWS Cloud9 e entrar no console do AWS Cloud9, consulte Configuração de equipe para o AWS Cloud9 no Guia do usuário do AWS Cloud9.

Etapa 2: Configurar o ambiente de desenvolvimento do AWS Cloud9

Depois de entrar no console do AWS Cloud9, use o console para criar um ambiente de desenvolvimento do AWS Cloud9. Depois de criar o ambiente, o AWS Cloud9 abre o IDE para esse ambiente.

Para obter detalhes, consulte Criação de um ambiente no AWS Cloud9 no Guia do usuário do AWS Cloud9.

> **Note**
>
> Ao criar o ambiente no console pela primeira vez, recomendamos selecionar a opção Create a new instance for environment (EC2) (Criar um nova instância para o ambiente (EC2)). Essa opção solicita que o AWS Cloud9 crie um ambiente, inicie uma instância do Amazon EC2 e, em seguida, conecte a nova instância ao novo ambiente. Essa é a maneira mais rápida de começar a usar o AWS Cloud9.

Etapa 3: Configurar o SDK para JavaScript

Depois que o AWS Cloud9 abrir o IDE para o ambiente de desenvolvimento, siga um ou ambos os procedimentos abaixo a fim de usar o IDE para configurar o SDK para JavaScript no ambiente.

Para instalar o SDK para JavaScript para Node.js

1. Se o terminal ainda não estiver aberto no IDE, abra-o. Para isso, na barra de menus no IDE, escolha Window, New Terminal (Janela, novo terminal).
2. Execute o comando a seguir para usar o nmp para instalar o SDK para JavaScript.

   ```
   npm install aws-sdk
   ```

   Caso o IDE não consiga encontrar npm, execute os comandos a seguir, um por vez, na ordem a seguir, para instalar npm. (Esses comandos pressupõem que você tenha escolhido a opção Create a new instance for environment (EC2) (Criar uma nova instância para o ambiente [EC2]), anteriormente neste tópico.)

> **Warning**
>
> AWS não controla o seguinte código. Antes de executar, certifique-se de verificar sua autenticidade e integridade. Mais informações sobre esse código podem ser encontradas no repositório GitHub do nvm.

```
curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.34.0/install.sh | bash #
 Download and install Node Version Manager (nvm).
. ~/.bashrc                                                                    #
 Activate nvm.
nvm install node                                                               #
 Use nvm to install npm (and Node.js at the same time).
```

Para configurar o SDK para JavaScript no navegador

Você não precisa instalar o SDK para JavaScript para usá-lo em scripts de navegador. Carregue o pacote do SDK para JavaScript hospedado diretamente pela AWS com um script nas suas páginas HTML.

Faça download de versões distribuíveis mini e não mini do SDK para JavaScript atual no GitHub em https://github.com/aws/aws-sdk-js/tree/master/dist.

Etapa 4: Fazer download do código de exemplo

Use o terminal que você abriu na etapa anterior para fazer download do código de exemplo para o SDK para JavaScript no ambiente de desenvolvimento do AWS Cloud9. (Se o terminal ainda não estiver aberto no IDE, abra-o escolhendo Window, New Terminal (Janela, novo terminal) na barra de menus no IDE.)

Para fazer download do código de exemplo, execute o comando a seguir. Esse comando faz download de uma cópia de todos os exemplos de código usados na documentação oficial do AWS SDK no diretório raiz do ambiente.

```
git clone https://github.com/awsdocs/aws-doc-sdk-examples.git
```

Para encontrar exemplos de código do SDK para JavaScript, use a janela Ambiente para abrir o *ENVIRONMENT_NAME*`\aws-doc-sdk-examples\javascript\example_code`, em que *ENVIRONMENT_NAME* é o nome do ambiente de desenvolvimento do AWS Cloud9.

Para saber como trabalhar com esses e outros exemplos de código, consulte Exemplos de código do SDK para JavaScript.

Etapa 5: Executar e depurar o código de exemplo

Para executar o código no ambiente de desenvolvimento do AWS Cloud9, consulte Executar seu código no Guia do usuário do AWS Cloud9.

Para depurar código Node.js, consulte Depurar o código no Guia do usuário do AWS Cloud9.

Configuração do SDK para JavaScript

Os tópicos desta seção explicam como instalar o SDK para JavaScript para uso em navegadores e com Node.js. Ele também mostra como carregar o SDK para que você possa acessar os serviços web compatíveis com o SDK.

> **Note**
>
> Desenvolvedores do React Native devem usar o AWS Amplify para criar projetos na AWS. Consulte o arquivamento aws-sdk-react-native para obter detalhes.

Tópicos

- Pré-requisitos
- Instalação do SDK for JavaScript
- Carregando o SDK para JavaScript
- Atualização da versão 1 do SDK para JavaScript

Pré-requisitos

Antes de usar o AWS SDK for JavaScript, determine se seu código precisa rodar em Node.js ou em navegadores. Depois disso, faça o seguinte:

- Para Node.js, instale o Node.js nos seus servidores, se ele ainda não estiver instalado.
- Para navegadores, identifique as versões do navegador para as quais você precisa oferecer compatibilidade.

Tópicos

- Configurar um ambiente Node.js da AWS
- Navegadores compatíveis

Configurar um ambiente Node.js da AWS

Para configurar um ambiente Node.js da AWS no qual você possa executar seu aplicativo, use qualquer um dos seguintes métodos:

- Escolha uma imagem de máquina da Amazon (AMI) com o Node.js pré-instalado e crie uma instância do Amazon EC2 usando essa AMI. Ao criar sua instância do Amazon EC2, selecione sua AMI no AWS Marketplace. Pesquise no AWS Marketplace por Node.js e escolha uma opção de AMI que inclua uma versão do Node.js (32 ou 64 bits) pré-instalada.
- Crie uma instância do Amazon EC2 e instale o Node.js. Para obter mais informações sobre como instalar o Node.js em uma instância do Amazon Linux, consulte Tutorial: Configuração do Node.js em uma instância do Amazon EC2.
- Crie um ambiente sem servidor usando o AWS Lambda para executar o Node.js como uma função do Lambda. Para obter mais informações sobre como usar o Node.js em uma função do Lambda, consulte Modelo de programação (Node.js) no Guia do desenvolvedor do AWS Lambda.
- Implante seu aplicativo do Node.js no AWS Elastic Beanstalk. Para obter mais informações sobre como usar o Node.js com Elastic Beanstalk, consulte Implantar aplicativos do Node.js no AWS Elastic Beanstalk no Guia do Desenvolvedor do AWS Elastic Beanstalk.
- Crie um servidor de aplicativos do Node.js usando AWS OpsWorks. Para obter mais informações sobre como usar Node.js com AWS OpsWorks, consulte Criar sua primeira pilha do Node.js no Guia do usuário do AWS OpsWorks.

Navegadores compatíveis

O SDK para JavaScript oferece suporte a todos os navegadores modernos, incluindo estas versões mínimas:

Navegador	Version (Versão)
Google Chrome	28.0+
Mozilla Firefox	26.0+
Opera	17.0+
Microsoft Edge	25.10+

Navegador	Version (Versão)
Windows Internet Explorer	N/D
Apple Safari	5+
Navegador do Android	4.3+

> **Note**
>
> Talvez as estruturas, como o AWS Amplify, não ofereçam o mesmo suporte a navegadores que o SDK para JavaScript. Verifique a documentação de uma estrutura para obter detalhes.

Instalação do SDK for JavaScript

Se você instala o AWS SDK for JavaScript e como você faz isso depende de o código executar nos módulos do Node.js ou nos scripts do navegador.

Nem todos os serviços estão disponíveis imediatamente no SDK. Para descobrir quais serviços atualmente são compatíveis com o AWS SDK for JavaScript, consulte https://github.com/aws/aws-sdk-js/blob/master/SERVICES.md

Node

A forma preferida de instalar o AWS SDK for JavaScript para Node.js é usar npm, o gerenciador de pacotes Node.js. Para fazer isso, digite isso na linha de comando.

```
npm install aws-sdk
```

Caso você veja essa mensagem de erro:

```
npm WARN deprecated node-uuid@1.4.8: Use uuid module instead
```

Digite estes comandos na linha de comando:

```
npm uninstall --save node-uuid
npm install --save uuid
```

Browser

Você não precisa instalar o SDK para usá-lo em scripts de navegador. Carregue o pacote do SDK hospedado diretamente pela Amazon Web Services com um script nas suas páginas HTML. O pacote SDK hospedado é compatível com o subconjunto de serviços da AWS que impõem o CORS (cross-origin resource sharing, compartilhamento de recursos de origem cruzada). Para ter mais informações, consulte Carregando o SDK para JavaScript.

Você pode criar uma compilação personalizada do SDK na qual seleciona os serviços web e versões específicos que deseja usar. Em seguida, faça download do pacote personalizado do SDK para desenvolvimento local e hospede-o ao seu aplicativo para usar. Para obter mais informações sobre a criação de um build personalizado do SDK, consulte Criar o SDK para navegadores.

Você pode fazer download das versões minimizada e não minimizada distribuíveis do AWS SDK for JavaScript atual no GitHub em:

https://github.com/aws/aws-sdk-js/tree/master/dist

Instalar usando o Bower

Bower é um gerenciador de pacotes para a web. Após instalar Bower, você pode usá-lo para instalar o SDK. Para instalar o SDK usando o Bower, digite o seguinte em uma janela de terminal:

```
bower install aws-sdk-js
```

Carregando o SDK para JavaScript

A forma como você carrega o SDK para JavaScript depende se você estiver o carregando para execução em um navegador ou em Node.js.

Nem todos os serviços estão disponíveis imediatamente no SDK. Para descobrir quais serviços atualmente são compatíveis com o AWS SDK for JavaScript, consulte https://github.com/aws/aws-sdk-js/blob/master/SERVICES.md

Node.js

Depois de instalar o SDK, você pode carregar o pacote da AWS no aplicativo do seu nó usando `require`.

```
var AWS = require('aws-sdk');
```

React Native

Para usar o SDK em um projeto React Native, primeiro instale o SDK usando npm:

```
npm install aws-sdk
```

No seu aplicativo, faça referência à versão compatível do React Native do SDK com o seguinte código:

```
var AWS = require('aws-sdk/dist/aws-sdk-react-native');
```

Browser

A maneira mais rápida de começar a usar o SDK é carregar o pacote do SDK hospedado diretamente da Amazon Web Services. Para fazer isso, adicione um elemento `<script>` às suas páginas HTML no seguinte formato:

```
<script src="https://sdk.amazonaws.com/js/aws-sdk-SDK_VERSION_NUMBER.min.js"></
script>
```

Para encontrar o SDK_VERSION_NUMBER atual, consulte a Referência da API para o SDK para JavaScript no Guia de referência da API. AWS SDK for JavaScript

Depois que o SDK carregar na sua página, o SDK estará disponível na variável global `AWS` (ou `window.AWS`).

Se você agrupa seu código e as dependências do módulo usando browserify, você carrega o SDK usando `require`, da mesma forma como em Node.js.

Atualização da versão 1 do SDK para JavaScript

As notas a seguir ajudam você a atualizar o SDK para JavaScript da versão 1 para a versão 2.

Conversão automática dos tipos Base64 e Timestamp Types na entrada/saída

O SDK agora codifica e decodifica automaticamente os valores codificados em base64, bem como valores de timestamp, em nome do usuário. Essa alteração afeta qualquer operação em que os valores de base64 ou timestamp foram enviados por uma solicitação ou retornados em uma resposta que permite valores codificados em base64.

O código do usuário que converteu previamente base64 não é mais necessário. Os valores codificados como base64 agora são apresentados como objetos de buffer de respostas do servidor e também podem ser passados como entrada do buffer. Por exemplo, os seguintes parâmetros `SQS.sendMessage` da versão 1:

```
var params = {
   MessageBody: 'Some Message',
   MessageAttributes: {
      attrName: {
         DataType: 'Binary',
         BinaryValue: new Buffer('example text').toString('base64')
      }
   }
};
```

Podem ser reescritos da forma a seguir.

```
var params = {
   MessageBody: 'Some Message',
   MessageAttributes: {
      attrName: {
         DataType: 'Binary',
         BinaryValue: 'example text'
      }
   }
};
```

Veja como a mensagem é lida.

```
sqs.receiveMessage(params, function(err, data) {
  // buf is <Buffer 65 78 61 6d 70 6c 65 20 74 65 78 74>
  var buf = data.Messages[0].MessageAttributes.attrName.BinaryValue;
```

```
  console.log(buf.toString()); // "example text"
});
```

response.data.RequestId transferido para response.requestId

O SDK agora armazena IDs de solicitação para todos os serviços em um único lugar do objeto `response`, em vez de dentro da propriedade `response.data`. Isso melhora a consistência entre serviços que expõem IDs de solicitação de diferentes formas. Isso também é uma alteração inovadora que renomeia a propriedade `response.data.RequestId` para `response.requestId` (`this.requestId` dentro de uma função de retorno de chamada).

No seu código, altere o seguinte:

```
svc.operation(params, function (err, data) {
  console.log('Request ID:', data.RequestId);
});
```

Para o seguinte:

```
svc.operation(params, function () {
  console.log('Request ID:', this.requestId);
});
```

Elementos expostos do Wrapper

Se você usar `AWS.ElastiCache`, `AWS.RDS` ou `AWS.Redshift`, deve acessar a resposta por meio da propriedade de saída de nível superior na resposta para algumas operações.

Por exemplo, o método `RDS.describeEngineDefaultParameters` usado para retornar o seguinte:

```
{ Parameters: [ ... ] }
```

Agora retorna o seguinte:

```
{ EngineDefaults: { Parameters: [ ... ] } }
```

A lista de operações afetadas para cada serviço é exibida na tabela a seguir.

Classe do cliente	Operações
AWS.ElastiCache	authorizeCacheSecurityGroupIngress createCacheCluster createCacheParameterGroup createCacheSecurityGroup createCacheSubnetGroup createReplicationGroup deleteCacheCluster deleteReplicationGroup describeEngineDefaultParameters modifyCacheCluster modifyCacheSubnetGroup modifyReplicationGroup purchaseReservedCacheNodesOffering rebootCacheCluster revokeCacheSecurityGroupIngress
AWS.RDS	addSourceIdentifierToSubscription authorizeDBSecurityGroupIngress copyDBSnapshot createDBInstance createDBInstanceReadReplica

Classe do cliente	Operações
	createDBParameterGroup createDBSecurityGroup createDBSnapshot createDBSubnetGroup createEventSubscription createOptionGroup deleteDBInstance deleteDBSnapshot deleteEventSubscription describeEngineDefaultParameters modifyDBInstance modifyDBSubnetGroup modifyEventSubscription modifyOptionGroup promoteReadReplica purchaseReservedDBInstances Offering rebootDBInstance removeSourceIdentifierFromS ubscription restoreDBInstanceFromDBSnapshot restoreDBInstanceToPointInTime

Classe do cliente	Operações
	`revokeDBSecurityGroupIngress`

Classe do cliente	Operações
AWS.Redshift	authorizeClusterSecurityGroupIngress authorizeSnapshotAccess copyClusterSnapshot createCluster createClusterParameterGroup createClusterSecurityGroup createClusterSnapshot createClusterSubnetGroup createEventSubscription createHsmClientCertificate createHsmConfiguration deleteCluster deleteClusterSnapshot describeDefaultClusterParameters disableSnapshotCopy enableSnapshotCopy modifyCluster modifyClusterSubnetGroup modifyEventSubscription modifySnapshotCopyRetentionPeriod

Classe do cliente	Operações

	`purchaseReservedNodeOffering`
	`rebootCluster`
	`restoreFromClusterSnapshot`
	`revokeClusterSecurityGroupIngress`
	`revokeSnapshotAccess`
	`rotateEncryptionKey`

Propriedades do cliente eliminadas

As propriedades `.client` e `.Client` foram removidos dos objetos de serviço. Se você usar a propriedade `.Client` em uma classe de serviço ou uma propriedade `.client` na instância de um objeto de serviço, remova essas propriedades do seu código.

O código a seguir usado com a versão 1 do SDK para JavaScript:

```
var sts = new AWS.STS.Client();
// or
var sts = new AWS.STS();

sts.client.operation(...);
```

Deve ser alterado para o código a seguir:

```
var sts = new AWS.STS();
sts.operation(...)
```

Configuração do SDK para JavaScript

Antes de usar o SDK para JavaScript para invocar serviços da web usando a API, você deve configurar o SDK. Você deve configurar pelo menos estas configurações:

- A região na qual você solicitará serviços.
- As credenciais que autorizam o acesso aos recursos do SDK.

Além dessas configurações, talvez você também precise configurar permissões para os recursos da AWS. Por exemplo, limite o acesso a um bucket do Amazon S3 ou restrinja uma tabela do Amazon DynamoDB para acesso somente leitura.

O Guia de referência de AWS SDKs e Ferramentas também contém configurações, recursos e outros conceitos fundamentais comuns entre muitos dos AWS SDKs.

Os tópicos nesta seção descrevem várias maneiras de configurar o SDK para JavaScript para Node.js e JavaScript em execução em um navegador da web.

Tópicos

- Usar o objeto de configuração global
- Definir a região da AWS
- Especificar endpoints personalizados
- Autenticação do SDK com AWS
- Definir credenciais
- Bloquear versões de API
- Considerações sobre Node.js
- Considerações sobre o script de navegador
- Empacotamento de aplicativos com o Webpack

Usar o objeto de configuração global

Existem duas maneiras de configurar o SDK:

- Defina a configuração global usando `AWS.Config`.

- Passe informações de configuração extras para um objeto de serviço.

Definir a configuração global com `AWS.Config` normalmente é mais fácil como conceitos básicos, mas a configuração no nível de serviço pode oferecer mais controle sobre serviços individuais. A configuração global especificada pelo serviço `AWS.Config` fornece configurações padrão para objetos de serviço criados de maneira subsequente, o que simplifica a configuração. No entanto, é possível atualizar a configuração de objetos de serviço individuais quando as necessidades variam em relação à configuração global.

Definir configuração global

Depois de carregar o pacote `aws-sdk` em seu código, você poderá usar a variável global `AWS` para acessar as classes do SDK e interagir com serviços individuais. O SDK inclui um objeto de configuração global, `AWS.Config`, que pode ser usado para especificar as definições de configuração do SDK exigidas pelo aplicativo.

Configure o SDK definindo as propriedades de `AWS.Config` de acordo com as necessidades do aplicativo. A tabela a seguir resume as propriedades `AWS.Config` mais usadas para definir a configuração do SDK.

Opções de configuração	Descrição
`credentials`	Obrigatório. Especifica as credenciais usadas para determinar o acesso a serviços e recursos.
`region`	Obrigatório. Especifica a região qual as solicitações de serviços são feitas.
`maxRetries`	Opcional. Especifica o número máximo de vezes que uma determinada solicitação é executada novamente.
`logger`	Opcional. Especifica um objeto logger no qual as informações de depuração são gravadas.
`update`	Opcional. Atualiza a configuração atual com novos valores.

Para obter mais informações sobre o objeto de configuração, consulte `Class: AWS.Config` na Referência da API.

Exemplos de configuração global

Defina a região e as credenciais em `AWS.Config`. Defina essas propriedades como parte do construtor `AWS.Config`, conforme mostrado no seguinte exemplo de script do navegador:

```
var myCredentials = new
 AWS.CognitoIdentityCredentials({IdentityPoolId:'IDENTITY_POOL_ID'});
var myConfig = new AWS.Config({
  credentials: myCredentials, region: 'us-west-2'
});
```

Também é possível definir essas propriedades após a criação de `AWS.Config` usando o método `update`, conforme mostrado no seguinte exemplo que atualiza a região:

```
myConfig = new AWS.Config();
myConfig.update({region: 'us-east-1'});
```

É possível obter suas credenciais padrão chamando o método estático `getCredentials` de `AWS.config`:

```
var AWS = require("aws-sdk");

AWS.config.getCredentials(function(err) {
  if (err) console.log(err.stack);
  // credentials not loaded
  else {
    console.log("Access key:", AWS.config.credentials.accessKeyId);
  }
});
```

Da mesma forma, se você tiver definido a região corretamente no arquivo `config`, obterá esse valor definindo a variável de ambiente `AWS_SDK_LOAD_CONFIG` como qualquer valor e chamando a propriedade estática `region` de `AWS.config`:

```
var AWS = require("aws-sdk");
```

```
console.log("Region: ", AWS.config.region);
```

Definir configuração por serviço

Cada serviço usado no SDK para JavaScript é acessado por meio de um objeto de serviço que faz parte da API desse serviço. Por exemplo, para acessar o serviço Amazon S3, você cria o objeto de serviço Amazon S3. Especifique as definições de configuração específicas de um serviço como parte do construtor desse objeto de serviço. Quando você define valores de configuração em um objeto de serviço, o construtor utiliza todos os valores de configuração usados por `AWS.Config`, inclusive credenciais.

Por exemplo, se você precisar acessar objetos do Amazon EC2 em várias regiões, crie um objeto de serviço do Amazon EC2 para cada região e defina a configuração da região de cada objeto de serviço de acordo.

```
var ec2_regionA = new AWS.EC2({region: 'ap-southeast-2', maxRetries: 15, apiVersion:
 '2014-10-01'});
var ec2_regionB = new AWS.EC2({region: 'us-east-1', maxRetries: 15, apiVersion:
 '2014-10-01'});
```

Também é possível definir valores de configuração específicos de um serviço ao configurar o SDK com `AWS.Config`. O objeto de configuração global é compatível com várias opções de configuração específicas do serviço. Para obter mais informações sobre a configuração específica do serviço, consulte `Class: AWS.Config` na Referência da API do AWS SDK for JavaScript.

Dados de configuração imutáveis

As alterações na configuração global se aplicam a solicitações de todos os objetos de serviço recém-criados. Os objetos de serviço recém-criados são configurados com os dados de configuração global atuais primeiro e todas as opções de configuração local. As atualizações feitas no objeto `AWS.config` global não se aplicam a objetos de serviço criados anteriormente.

Os objetos de serviço existentes devem ser atualizados manualmente com novos dados de configuração ou você deve criar e usar um novo objeto de serviço que tenha os novos dados de configuração. O exemplo a seguir cria um novo objeto de serviço do Amazon S3 com novos dados de configuração:

```
s3 = new AWS.S3(s3.config);
```

Definir a região da AWS

Uma região é um conjunto de recursos da AWS nomeado na mesma área geográfica. Um exemplo de uma Região é `us-east-1`, que é a Região Leste dos EUA (Norte da Virgínia). Você especifica uma região ao configurar o SDK para , de maneira que o SDK acesse os recursos nessa região. Alguns serviços só estão disponíveis em regiões específicas.

O SDK para JavaScript. não seleciona uma região por padrão. No entanto, é possível definir a região usando uma variável de ambiente, um arquivo `config` compartilhado ou o objeto de configuração global.

Em um construtor de classes do cliente

Ao instanciar um objeto de serviço, especifique a região da desse recurso como parte do construtor de classe de cliente, conforme mostrado aqui.

```
var s3 = new AWS.S3({apiVersion: '2006-03-01', region: 'us-east-1'});
```

Usar o objeto de configuração global

Para definir a região no código JavaScript, atualize o objeto de configuração global `AWS.Config` conforme mostrado aqui.

```
AWS.config.update({region: 'us-east-1'});
```

Para obter mais informações sobre regiões atuais e serviços disponíveis em cada região, consulte Regiões e endpoints da AWS na Referência geral da AWS.

Usar uma variável de ambiente

Defina a região usando a variável de ambiente `AWS_REGION`. Se você definir essa variável, o SDK para JavaScript vai lê-la e usá-la.

Usar um arquivo de configuração compartilhado

Assim como o arquivo de credenciais compartilhado permite armazenar credenciais a serem usadas pelo SDK, é possível manter a região e outras definições de configuração compartilhadas em um

arquivo chamado `config` usado pelos SDKs. Caso a variável de ambiente `AWS_SDK_LOAD_CONFIG` tenha sido definida como qualquer valor, o SDK para JavaScript procurará automaticamente um arquivo `config` quando ele for carregado. Onde você salva o arquivo `config` depende do sistema operacional:

- Usuários do macOS ou do Unix: `~/.aws/config`
- Usuários do Windows: `C:\Users\USER_NAME\.aws\config`

Se não tiver um arquivo `config` compartilhado, você poderá criar um no diretório designado. No exemplo a seguir, o arquivo `config` define a região e o formato de saída.

```
[default]
   region=us-east-1
   output=json
```

Para obter mais informações sobre como usar arquivos de configuração e credenciais compartilhados, consulte o Carregar credenciais em Node.js do arquivo de credenciais compartilhado ou Arquivos de configuração e credencial no Guia de usuário do AWS Command Line Interface.

Ordem de precedência para definir a região

A ordem de precedência de definição da região é a seguinte:

- Se uma região for passada para um construtor de classe de cliente, essa região será usada. Do contrário...
- Se uma região for definida no objeto de configuração global, essa região será usada. Do contrário...
- Se a variável de ambiente `AWS_REGION` for um valor confiável, essa região será usada. Do contrário...
- Se a variável de ambiente `AMAZON_REGION` for um valor confiável, essa região será usada. Do contrário...
- Se a variável de ambiente `AWS_SDK_LOAD_CONFIG` for definida como qualquer valor e o arquivo de credenciais compartilhado (`~/.aws/credentials` ou o caminho indicado por `AWS_SHARED_CREDENTIALS_FILE`) contiver uma região para o perfil configurado, será usada essa região. Do contrário...

- Se a variável de ambiente `AWS_SDK_LOAD_CONFIG` for definida como qualquer valor e o arquivo de configuração (`~/.aws/config` ou o caminho indicado por `AWS_CONFIG_FILE`) contiver uma região para o perfil configurado, será usada essa região.

Especificar endpoints personalizados

Os métodos de chamadas para API no SDK para JavaScript são feitas para URIs de endpoint de serviço. Por padrão, esses endpoints são criados na região configurada para o código. No entanto, há situações em que você precisa especificar um endpoint personalizado para as chamadas de API.

Formato do string de endpoint

Os valores de endpoint devem ser uma string no formato:

`https://{service}.{region}.amazonaws.com`

Endpoints da região ap-northeast-3

A região `ap-northeast-3` no Japão não é retornada por APIs de enumeração da região, como `EC2.describeRegions`. Para definir endpoints para essa região, siga o formato descrito anteriormente. Dessa maneira, o endpoint do Amazon EC2 dessa região seria

`ec2.ap-northeast-3.amazonaws.com`

Endpoints para MediaConvert

Você precisa criar um endpoint personalizado a ser usado com o MediaConvert. Cada conta de cliente recebe o próprio endpoint, que você deve usar. Aqui está um exemplo de como usar um endpoint personalizado com o MediaConvert.

```
// Create MediaConvert service object using custom endpoint
var mcClient = new AWS.MediaConvert({endpoint: 'https://abcd1234.mediaconvert.us-
west-1.amazonaws.com'});

var getJobParams = {Id: 'job_ID'};

mcClient.getJob(getJobParams, function(err, data)) {
   if (err) console.log(err, err.stack); // an error occurred
   else console.log(data); // successful response
};
```

Para obter o endpoint da API de conta, consulte `MediaConvert.describeEndpoints` na Referência da API.

Certifique-se de especificar a mesma região no código da a região no URI de endpoint personalizado. Uma incompatibilidade entre a definição da região e o URI de endpoint personalizado pode causar uma falha nas chamadas de API.

Para obter mais informações sobre o MediaConvert, consulte a classe *`AWS.MediaConvert`* na Referência da API ou no Guia do usuário do AWS Elemental MediaConvert.

Autenticação do SDK com AWS

Você precisa estabelecer como seu código deve ser autenticado com a AWS ao desenvolver com Serviços da AWS. É possível configurar o acesso programático aos recursos da AWS de maneiras diferentes, dependendo do ambiente e do acesso da AWS disponível para você.

Para escolher seu método de autenticação e configurá-lo para o SDK, consulte Autenticação e acesso no Guia de referência de ferramentas e SDKs da AWS.

Recomendamos que novos usuários que estão desenvolvendo localmente e não receberam um método de autenticação do empregador configurem o AWS IAM Identity Center. Esse método inclui a instalação da AWS CLI para facilitar a configuração e entrar regularmente no portal de acesso da AWS. Se você escolher esse método, seu ambiente deverá conter os seguintes elementos depois de concluir o procedimento de autenticação do IAM Identity Center no Guia de referência de ferramentas e SDKs da AWS:

- A AWS CLI, que você usa para iniciar uma sessão do portal de acesso da AWS antes de executar a aplicação.
- Um arquivo AWS`config` compartilhado com um perfil de `[default]` com um conjunto de valores de configuração que podem ser referenciados a partir do SDK. Para encontrar a localização desse arquivo, consulte Localização dos arquivos compartilhados no Guia de referência de ferramentas e SDKs da AWS.
- O arquivo `config` compartilhado define a configuração do `region`. Isso define o Região da AWS padrão que o SDK usa para solicitações da AWS. Essa região é usada para solicitações de serviço do SDK que não são fornecidas com uma Região específica para uso.
- O SDK usa a configuração do provedor do token de SSO do perfil para adquirir credenciais antes de enviar solicitações para a AWS. O valor `sso_role_name`, que é um perfil do IAM conectada a

um conjunto de permissões do IAM Identity Center, deve permitir o acesso aos Serviços da AWS usados na aplicação.

O arquivo `config` de amostra a seguir mostra um perfil padrão configurado com o provedor de token de SSO. A configuração `sso_session` do perfil se refere à seção do `sso-session`. A seção `sso-session` contém configurações para iniciar uma sessão do portal de acesso da AWS.

```
[default]
sso_session = my-sso
sso_account_id = 111122223333
sso_role_name = SampleRole
region = us-east-1
output = json

[sso-session my-sso]
sso_region = us-east-1
sso_start_url = https://provided-domain.awsapps.com/start
sso_registration_scopes = sso:account:access
```

O SDK para JavaScript não precisa que pacotes adicionais (como `SSO` e`SSOOIDC`) sejam adicionados ao seu aplicativo para usar a autenticação do IAM Identity Center.

Iniciar uma sessão do portal de acesso AWS

Antes de executar um aplicativo que acessa os Serviços da AWS, você precisa de uma sessão ativa do portal de acesso da AWS para que o SDK use a autenticação do Centro de Identidade do IAM para resolver as credenciais. Dependendo da duração da sessão configurada, o seu acesso acabará expirando e o SDK encontrará um erro de autenticação. Para entrar no portal de acesso da AWS, execute o seguinte comando na AWS CLI.

```
aws sso login
```

Se você seguiu as orientações e tem um perfil padrão configurado, não precisará chamar o comando com uma opção de `--profile`. Se a configuração do provedor de token de SSO estiver usando um perfil nomeado, o comando será `aws sso login --profile named-profile`.

Para, como opção, testar se você já tem uma sessão ativa, execute o seguinte comando da AWS CLI.

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

Se a sua sessão estiver ativa, a resposta a este comando relata a conta do IAM Identity Center e o conjunto de permissões configurados no arquivo `config` compartilhado.

> **Note**
>
> Se você já tiver uma sessão ativa do portal de acesso da AWS e executar `aws sso login`, não será necessário fornecer credenciais.
> O processo de login pode solicitar que você permita que a AWS CLI acesse seus dados. Como a AWS CLI é construída sobre o SDK para Python, as mensagens de permissão podem conter variações do nome do `botocore`.

Mais informações de autenticação

Os usuários humanos, também conhecidos como identidades humanas, são as pessoas, os administradores, os desenvolvedores, os operadores e os consumidores de suas aplicações. Eles devem ter uma identidade para acessar seus ambientes e aplicações da AWS. Usuários humanos que são membros da sua organização (ou seja, você, o desenvolvedor) são conhecidos como identidades da força de trabalho.

Use credenciais temporárias ao acessar a AWS. Você pode usar um provedor de identidade para que seus usuários humanos recebam acesso federado a contas da AWS, assumindo funções que fornecem credenciais temporárias. Para gerenciamento de acesso centralizado, recomendamos que você use o AWS IAM Identity Center (IAM Identity Center) para gerenciar o acesso às suas contas e as permissões nessas contas. Para obter mais alternativas, consulte as informações a seguir.

- Para saber mais sobre as práticas recomendadas, consulte Práticas recomendadas de segurança no IAM no Guia do usuário do IAM.
- Para criar credenciais de curto prazo da AWS, consulte Credenciais de segurança temporárias no Guia do usuário do IAM.
- Para saber mais sobre outros provedores de credenciais, consulte Provedores de credenciais padronizados no Guia de referência de AWS SDKs e ferramentas.

Definir credenciais

A AWS usa credenciais para identificar quem está chamando serviços e se o acesso aos recursos solicitados é permitido.

Independentemente da execução em um navegador da Web ou em um servidor Node.js, o código JavaScript deve obter credenciais válidas para acessar serviços por meio da API. As credenciais podem ser definidas globalmente no objeto de configuração usando `AWS.Config`, ou por serviço, passando credenciais diretamente para um objeto de serviço.

Há várias maneiras de definir credenciais diferentes entre Node.js e JavaScript em navegadores da web. Os tópicos nesta seção descrevem como definir credenciais em Node.js ou navegadores da web. Em cada caso, as opções são apresentadas na ordem recomendada.

Melhores práticas para credenciais

A definição de credenciais apropriada garante que o aplicativo ou o script de navegador possa acessar os serviços e os recursos necessários ao mesmo tempo que minimiza a exposição a problemas de segurança que possam afetar aplicativos de missão crítica ou comprometer dados confidenciais.

Um princípio importante a ser aplicado durante a definição de credenciais é sempre conceder o menor privilégio necessário para a tarefa. É mais seguro fornecer permissões mínimas nos recursos e adicionar mais permissões adicionais conforme necessário, em vez de fornecer permissões que excedam o menor privilégio e, dessa forma, precisar corrigir problemas de segurança que possam ser descobertos depois. Por exemplo, a menos que você precise ler e gravar recursos individuais, como objetos em um bucket do Amazon S3 ou uma tabela do DynamoDB, defina essas permissões como somente leitura.

Para obter mais informações sobre como conceder o privilégio mínimo, consulte a seção Conceder privilégio mínimo do tópico Melhores práticas no Guia do usuário do IAM.

> **Warning**
>
> Embora seja possível fazer isso, recomendamos não codificar credenciais dentro de um aplicativo ou de um script de navegador. As credenciais de codificação física representam o risco de expor informações confidenciais.

Para obter mais informações sobre como gerenciar as chaves de acesso, consulte Práticas recomendadas para gerenciar chaves de acesso da AWS na Referência geral da AWS.

Tópicos

- Definir credenciais em Node.js
- Definir credenciais em um navegador da Web

Definir credenciais em Node.js

Há várias maneiras em Node.js de fornecer as credenciais para o SDK. Algumas dessas são mais seguras e outras oferecem mais comodidade durante o desenvolvimento de aplicativos. Ao obter credenciais em Node.js, tome cuidado ao confiar em mais de uma origem como uma variável de ambiente e um arquivo JSON carregado. Altere as permissões em que o código é executado sem perceber a alteração que aconteceu.

Aqui estão as maneiras como é possível fornecer as credenciais em ordem de recomendação:

1. Carregado a partir dos perfis do AWS Identity and Access Management (IAM) para o Amazon EC2
2. Carregadas a partir do arquivo de credenciais compartilhado (`~/.aws/credentials`)
3. Carregadas de variáveis de ambiente
4. Carregadas de um arquivo JSON no disco
5. Outras classes de provedor de credenciais fornecidas pelo SDK do JavaScript

Se mais de uma fonte de credenciais estiver disponível para o SDK, a precedência padrão da seleção será a seguinte:

1. Credenciais que são explicitamente definidas por meio do construtor do cliente de serviço
2. Variáveis de ambiente
3. O arquivo de credenciais compartilhado
4. Credenciais carregadas do provedor de credenciais do ECS (se aplicável)
5. Credenciais obtidas por meio de um processo de credenciais especificado no arquivo de configuração da AWS compartilhado ou no arquivo de credenciais compartilhado. Para ter mais informações, consulte the section called “Credenciais usando um processo de credenciais configuradas”.

6. Credenciais carregadas do IAM da AWS usando o provedor de credenciais da instância do Amazon EC2 (se configurado nos metadados da instância).

Para obter mais informações, consulte `Class: AWS.Credentials` e `Class: AWS.CredentialProviderChain` na Referência da API do .

> **Warning**
>
> Embora seja possível fazer isso, não recomendamos codificar as credenciais da AWS no aplicativo. Codificar credenciais oferece um risco de expor o ID de chave de acesso e a chave de acesso secreta.

Os tópicos nesta seção descrevem como carregar credenciais em Node.js.

Tópicos

- Carregar credenciais em Node.js de perfis do IAM para o Amazon EC2
- Carregar credenciais de uma função do Lambda de Node.js
- Carregar credenciais em Node.js do arquivo de credenciais compartilhado
- Carregar credenciais em Node.js de variáveis de ambiente.
- Carregar credenciais em Node.js de um arquivo JSON
- Carregamento de credenciais no Node.js usando um processo de credenciais configuradas

Carregar credenciais em Node.js de perfis do IAM para o Amazon EC2

Se executar o aplicativo Node.js em uma instância do Amazon EC2, você poderá aproveitar perfis do IAM para o Amazon EC2 fornecer credenciais automaticamente para a instância. Se você configurar a instância para usar perfis do IAM, o SDK selecionará automaticamente as credenciais do do aplicativo, eliminando a necessidade de fornecer credenciais manualmente.

Para obter mais informações sobre como adicionar perfis do IAM a uma instância do Amazon EC2, consulte Como usar funções do IAM para instâncias do Amazon EC2 no AWS Guia de referência dos SDKs e Ferramentas.

Carregar credenciais de uma função do Lambda de Node.js

Ao criar uma função do AWS Lambda, você deve criar um perfil do IAM especial com permissão para executar a função. Essa função é chamada de função de execução. Ao configurar uma função do Lambda, você deve especificar o perfil do IAM que criou como a função de execução correspondente.

A função de execução fornece a função do Lambda com as credenciais de que precisa para executar e invocar outros serviços da web. Dessa maneira, você não precisa fornecer credenciais para o código Node.js gravado em uma função do Lambda.

Para obter mais informações sobre como configurar uma função de execução do Lambda, consulte Gerenciar permissões: usar uma perfil do IAM (prtfil de execução) no Guia do desenvolvedor do AWS Lambda.

Carregar credenciais em Node.js do arquivo de credenciais compartilhado

Mantenha os dados das credenciais do AWS em um arquivo compartilhado usado por SDKs e a interface de linha de comando. Quando o SDK para JavaScript carregar, ele pesquisará automaticamente o arquivo de credenciais compartilhado, que é chamado "credentials". Onde você mantém o arquivo de credenciais compartilhado depende do sistema operacional:

- O arquivo de credenciais compartilhado no Linux, Unix e macOS: `~/.aws/credentials`
- O arquivo de credenciais compartilhado no Windows: `C:\Users\USER_NAME\.aws\credentials`

Se você ainda não tiver um arquivo de credenciais compartilhadas, consulte Autenticação do SDK com AWS. Após seguir essas instruções, você verá um texto semelhante ao seguinte no arquivo de credenciais, em que *<YOUR_ACCESS_KEY_ID>* é o ID de chave de acesso e *<YOUR_SECRET_ACCESS_KEY>* é a chave de acesso secreta:

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

Para obter um exemplo que mostra este arquivo sendo usado, consulte Conceitos básicos de Node.js.

O título da seção `[default]` especifica um perfil padrão e os valores associados das credenciais. Crie perfis adicionais no mesmo arquivo de configuração compartilhado, cada um com as próprias informações de credencial. O seguinte exemplo mostra um arquivo de configuração com o perfil padrão e dois perfis adicionais:

```
[default] ; default profile
aws_access_key_id = <DEFAULT_ACCESS_KEY_ID>
aws_secret_access_key = <DEFAULT_SECRET_ACCESS_KEY>

[personal-account] ; personal account profile
aws_access_key_id = <PERSONAL_ACCESS_KEY_ID>
aws_secret_access_key = <PERSONAL_SECRET_ACCESS_KEY>

[work-account] ; work account profile
aws_access_key_id = <WORK_ACCESS_KEY_ID>
aws_secret_access_key = <WORK_SECRET_ACCESS_KEY>
```

Por padrão, o SDK verifica a variável de ambiente `AWS_PROFILE` para determinar qual perfil usar. Se a variável `AWS_PROFILE` não estiver definida no ambiente, o SDK usará as credenciais do perfil `[default]`. Para usar um dos perfis alternativos, configure ou altere o valor da variável de ambiente `AWS_PROFILE`. Por exemplo, considerando o arquivo de configuração mostrado acima, para usar as credenciais da conta de serviço, defina a variável de ambiente `AWS_PROFILE` como `work-account` (conforme adequado para seu sistema operacional).

> **Note**
>
> Ao definir variáveis de ambiente, lembre-se de tomar as medidas adequadas posteriormente (de acordo com as necessidades do seu sistema operacional) para disponibilizar as variáveis no ambiente de comando ou shell.

Depois de definir a variável de ambiente (se necessário), você pode executar um arquivo de JavaScript que usa o SDK como uma arquivo chamado `script.js`, por exemplo.

```
$ node script.js
```

Também selecione explicitamente o perfil usado pelo SDK definindo `process.env.AWS_PROFILE` antes de carregar o SDK ou selecionando o provedor de credenciais, conforme mostrado no seguinte exemplo:

```
var credentials = new AWS.SharedIniFileCredentials({profile: 'work-account'});
AWS.config.credentials = credentials;
```

Carregar credenciais em Node.js de variáveis de ambiente.

O SDK detecta automaticamente credenciais da AWS definidas como variáveis no ambiente e as usa em solicitações do SDK, eliminando a necessidade de gerenciar credenciais no aplicativo. As variáveis de ambiente definidas para fornecer as credenciais são:

- `AWS_ACCESS_KEY_ID`
- `AWS_SECRET_ACCESS_KEY`
- `AWS_SESSION_TOKEN`

Para obter mais detalhes sobre a configuração de variáveis de ambiente, consulte Suporte a variáveis de ambiente no Guia de referência de AWS SDKs e Ferramentas.

Carregar credenciais em Node.js de um arquivo JSON

Carregue a configuração e as credenciais de um documento JSON no disco usando `AWS.config.loadFromPath`. O caminho especificado é relativo ao diretório de trabalho atual do processo. Por exemplo, para carregar credenciais de um arquivo `'config.json'` com o seguinte conteúdo:

```
{ "accessKeyId": <YOUR_ACCESS_KEY_ID>, "secretAccessKey": <YOUR_SECRET_ACCESS_KEY>,
 "region": "us-east-1" }
```

Depois, use o seguinte código:

```
var AWS = require("aws-sdk");
AWS.config.loadFromPath('./config.json');
```

> **Note**
>
> Carregar dados de configuração de um documento JSON redefine todos os dados de configuração existentes. Adicione dados de configuração depois de usar essa técnica. Carregar credenciais de um documento JSON não é suportado em scripts de navegador.

Carregamento de credenciais no Node.js usando um processo de credenciais configuradas

É possível obter credenciais usando um método que não é baseado no SDK. Para fazê-lo, especifique um processo de credenciais no arquivo de configuração da AWS compartilhado ou no arquivo de credenciais compartilhado. Se a variável de ambiente `AWS_SDK_LOAD_CONFIG` for definida como qualquer valor, o SDK preferirá o processo especificado no arquivo de configuração em vez de o processo especificado no arquivo de credenciais (se houver).

Para obter detalhes sobre como especificar um processo de credenciais no arquivo de configuração da AWS compartilhado ou no arquivo de credenciais compartilhado, consulte o Command Reference AWS CLI, especificamente as informações sobre Obtenção de credenciais a partir de processos externos.

Para obter informações sobre como usar a variável de ambiente `AWS_SDK_LOAD_CONFIG`, consulte the section called “Usar um arquivo de configuração compartilhado” neste documento.

Definir credenciais em um navegador da Web

Há várias maneiras de fornecer as credenciais para o SDK de scripts de navegador. Algumas dessas são mais seguras e outras oferecem mais comodidade durante o desenvolvimento de scripts. Aqui estão as maneiras como é possível fornecer as credenciais em ordem de recomendação:

1. Usar o Amazon Cognito Identity para autenticar usuários e fornecer credenciais
2. Usar identidade federada da web
3. Codificada no script

> **Warning**
>
> Não recomendamos codificar as credenciais da AWS nos scripts. A codificação rígida de credenciais oferece um risco de expor o ID de chave de acesso e a chave de acesso secreta.

Tópicos

- Usando do Amazon Cognito Identity para autenticar usuários
- Usar identidade federada da web para autenticar usuários

- Exemplos de identidades federadas da Web

Usando do Amazon Cognito Identity para autenticar usuários

A forma recomendada de obter credenciais da AWS para os scripts do seu navegador é usar o objeto de credenciais do Amazon Cognito Identity, `AWS.CognitoIdentityCredentials`. O Amazon Cognito permite a autenticação de usuários por meio de provedores de identidade terceirizados.

Para usar o Amazon Cognito Identity, você deve primeiro criar um banco de identidades no console do Amazon Cognito. Um grupo de identidades representa o grupo de identidades fornecido pelo aplicativo para os usuários. As identidades atribuídas a usuários identificam com exclusividade cada conta de usuário. As identidades do Amazon Cognito não são credenciais. Elas são trocadas por credenciais usando o suporte à federação de identidades da web no AWS Security Token Service (AWS STS).

O Amazon Cognito ajuda a gerenciar a abstração de identidades entre vários provedores de identidade com o objeto `AWS.CognitoIdentityCredentials`. A identidade carregada acaba sendo trocada por credenciais no AWS STS.

Configuração do objeto de credenciais do Amazon Cognito Identity

Se você ainda não tiver criado um, crie um banco de identidades a ser usado com os scripts do navegador no Console do Amazon Cognito antes de configurar o `AWS.CognitoIdentityCredentials`. Criar e associar os perfis do IAM autenticados e não autenticados para o grupo de identidades.

Usuários não autenticados não têm a identidade verificada, tornando essa função apropriada para usuários convidados de seu aplicativo ou nos casos em que não importa se os usuários têm suas identidades verificadas. Os usuários autenticados fazem login no aplicativo por meio de um provedor de identidade de terceiros que verifica as identidades. Certifique-se de definir o escopo das permissões dos recursos de forma apropriada para que você não conceda acesso a eles a partir de usuários não autenticados.

Depois de configurar um grupo de identidades com provedores de identidade anexados, você poderá usar `AWS.CognitoIdentityCredentials` para autenticar usuários. Para configurar as credenciais de seu aplicativo para usar `AWS.CognitoIdentityCredentials`, defina a propriedade `credentials` do `AWS.Config` ou uma configuração por serviço. O exemplo a seguir usa `AWS.Config`:

```
AWS.config.credentials = new AWS.CognitoIdentityCredentials({
  IdentityPoolId: 'us-east-1:1699ebc0-7900-4099-b910-2df94f52a030',
  Logins: { // optional tokens, used for authenticated login
    'graph.facebook.com': 'FBTOKEN',
    'www.amazon.com': 'AMAZONTOKEN',
    'accounts.google.com': 'GOOGLETOKEN'
  }
});
```

A propriedade opcional `Logins` é um mapa de nomes de provedor de identidade para os tokens de identidade para esses provedores. Como você obtém o token do seu provedor de identidade depende do provedor que usa. Por exemplo, se o Facebook for um de seus provedores de identidade, você poderá usar a `FB.login` função do Facebook SDK para obter um token de provedor de identidade:

```
FB.login(function (response) {
  if (response.authResponse) { // logged in
    AWS.config.credentials = new AWS.CognitoIdentityCredentials({
      IdentityPoolId: 'us-east-1:1699ebc0-7900-4099-b910-2df94f52a030',
      Logins: {
        'graph.facebook.com': response.authResponse.accessToken
      }
    });

    s3 = new AWS.S3; // we can now create our service object

    console.log('You are now logged in.');
  } else {
    console.log('There was a problem logging you in.');
  }
});
```

Alternar usuários não autenticados para usuários autenticados

O Amazon Cognito é compatível com usuários autenticados e não autenticados. Os usuários não autenticados receberão acesso aos recursos se eles não estiverem conectados a nenhum dos provedores de identidade. Esse nível de acesso é útil para exibir conteúdo para usuários antes de fazer login. Cada usuário não autenticado tem uma identidade exclusiva no Amazon Cognito, mesmo que não tenha feito login e sido autenticado individualmente.

Usuário não autenticado inicialmente

Os usuários normalmente começam com a função não autenticada para a qual você define a propriedade de credenciais do objeto de configuração sem uma propriedade `Logins`. Neste caso, sua configuração padrão pode parecer com o seguinte:

```
// set the default config object
var creds = new AWS.CognitoIdentityCredentials({
 IdentityPoolId: 'us-east-1:1699ebc0-7900-4099-b910-2df94f52a030'
});
AWS.config.credentials = creds;
```

Mudar para usuário autenticado

Quando um usuário autenticado faz login em um provedor de identidade e você tem um token, é possível alternar o usuário de não autenticado para autenticado chamando uma função personalizada que atualiza o objeto de credenciais e adiciona o token `Logins`:

```
// Called when an identity provider has a token for a logged in user
function userLoggedIn(providerName, token) {
  creds.params.Logins = creds.params.Logins || {};
  creds.params.Logins[providerName] = token;

  // Expire credentials to refresh them on the next request
  creds.expired = true;
}
```

Também é possível criar um objeto `CognitoIdentityCredentials`. Se fizer isso, você deverá redefinir as propriedades de credenciais de objetos de serviço existentes criados. Os objetos de serviço só são lidos da configuração global na inicialização do objeto.

Para obter mais informações sobre o objeto `CognitoIdentityCredentials`, consulte `AWS.CognitoIdentityCredentials` na Referência de API.

Usar identidade federada da web para autenticar usuários

Você pode configurar diretamente os provedores de identidade individuais para acessar recursos da AWS usando a federação de identidades da web. Atualmente, a AWS é compatível com a autenticação de usuários usando a federação de identidades da web por meio de vários provedores de identidade:

- Login da Amazon
- Login do Facebook
- Login do Google

Você deve primeiramente registrar o aplicativo com os provedores compatíveis com o aplicativo. Crie um perfil do IAM e configure permissões para ele. A função do IAM criada acaba sendo usada para conceder as permissões configuradas para ela por meio do respectivo provedor de identidade. Por exemplo, é possível configurar uma função que permita a usuários conectados por meio do Facebook ter acesso de leitura a um bucket do Amazon S3 específico controlado por você.

Depois que tiver um perfil do IAM com privilégios configurados e um aplicativo registrado com os provedores de identidade escolhido, você poderá configurar o SDK para receber credenciais para o perfil do IAM usando código auxiliar da seguinte maneira:

```
AWS.config.credentials = new AWS.WebIdentityCredentials({
   RoleArn: 'arn:aws:iam::<AWS_ACCOUNT_ID>/:role/<WEB_IDENTITY_ROLE_NAME>',
   ProviderId: 'graph.facebook.com|www.amazon.com', // this is null for Google
   WebIdentityToken: ACCESS_TOKEN
});
```

O valor no parâmetro `ProviderId` depende do provedor de identidade especificado. O valor do parâmetro `WebIdentityToken` é o token de acesso recuperado de um login bem-sucedido com o provedor de identidade. Para obter mais informações sobre como configurar e recuperar tokens de acesso de cada provedor de identidade, consulte a documentação do provedor de identidade.

Etapa 1: Registrar com provedores de identidade

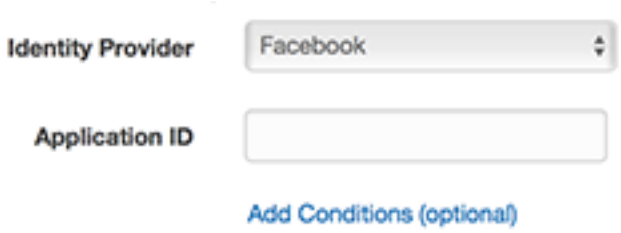
Para começar, registre um aplicativo com os provedores de identidade escolhidos para dar suporte. Você será solicitado a fornecer informações que identifiquem o aplicativo e possivelmente o autor. Isso garante que os provedores de identidade saibam quem está recebendo as informações do usuário. Em cada caso, o provedor de identidade emitirá um ID do aplicativo usado para configurar funções do usuário.

Etapa 2: Criar um perfil do IAM para um provedor de identidade

Depois de obter o ID do aplicativo do provedor de identidade, acesse o console do IAM no https://console.aws.amazon.com/iam/ para criar um novo perfil do IAM.

Para criar uma função do perfil do IAM para um provedor de identidade

1. Acesse a seção Roles (Funções) do console e escolha Create New Role (Criar nova função).
2. Digite um nome para a nova função que ajude a controlar o uso, como **`facebookIdentity`**, e escolha Next Step (Próxima etapa).
3. Em Select Role Type (Selecionar tipo de função), escolha Role for Identity Provider Access (Função de acesso do provedor de identidade).
4. Em Grant access to web identity providers (Conceder acesso a provedores de identidade da web), escolha Select (Selecionar).
5. Na lista Identity Provider (Provedor de identidade), escolha o provedor de identidade que você deseja usar nesse perfil do IAM.



6. Digite o ID do aplicativo fornecido pelo provedor de identidade em Application ID (ID do aplicativo) e escolha Next Step (Próxima etapa).
7. Configurar permissões para os recursos que você deseja expor, permitindo acesso a operações específicas em recursos específicos. Para obter mais informações sobre as permissões do IAM, consulte Visão geral das permissões do IAM do AWS no Manual do usuário do IAM. Analise e, se necessário, personalize a função a relação de confiança da função e escolha Next Step (Próxima etapa).
8. Anexe políticas adicionais de que você precisa e escolha Next Step (Próxima etapa). Para obter mais informações sobre políticas do IAM, consulte Visão geral das políticas do IAM no Guia do usuário do IAM.
9. Analise a nova função e escolha Create Role (Criar função).

Forneça outras restrições à função, como o escopo para IDs de usuário específicos. Se a função conceder permissões de gravação aos recursos, verifique o escopo correto da função para usuários com os privilégios corretos. Do contrário, qualquer usuário com uma identidade Amazon, Facebook ou Google será capaz de modificar recursos no aplicativo.

Para obter mais informações sobre como usar a federação de identidades da web no IAM, consulte Sobre federação de identidades da web no Guia de usuário do IAM.

Etapa 3: Obter um token de acesso do provedor após o login

Configure a ação de login para o aplicativo usando o SDK do provedor de identidade. Faça download e instale um SDK JavaScript do provedor de identidade que permita o login do usuário usando OAuth ou OpenID. Para obter informações sobre como fazer download e configurar o código do SDK no aplicativo, consulte a documentação do SDK do provedor de identidade:

- Login da Amazon
- Login do Facebook
- Login do Google

Etapa 4: obter credenciais temporárias

Depois que o aplicativo, as funções e as permissões de recursos forem configurados, adicione o código ao aplicativo para obter credenciais temporárias. Essas credenciais são fornecidas por meio do AWS Security Token Service usando a federação de identidades da web. Os usuários fazem login no provedor de identidade, que retorna um token de acesso. Configure o objeto `AWS.WebIdentityCredentials` usando o ARN do perfil do IAM criado para este provedor de identidade:

```
AWS.config.credentials = new AWS.WebIdentityCredentials({
    RoleArn: 'arn:aws:iam::<AWS_ACCOUNT_ID>:role/<WEB_IDENTITY_ROLE_NAME>',
    ProviderId: 'graph.facebook.com|www.amazon.com', // Omit this for Google
    WebIdentityToken: ACCESS_TOKEN // Access token from identity provider
});
```

Os objetos de serviço criados posteriormente terão as credenciais apropriadas. Os objetos criados antes de definir a propriedade `AWS.config.credentials` não terão as credenciais atuais.

Também é possível criar `AWS.WebIdentityCredentials` antes de recuperar o token de acesso. Isso permite criar objetos de serviço que dependam de credenciais antes de carregar o token de acesso. Para isso, crie o objeto de credenciais sem o parâmetro `WebIdentityToken`:

```
AWS.config.credentials = new AWS.WebIdentityCredentials({
  RoleArn: 'arn:aws:iam::<AWS_ACCOUNT_ID>:role/<WEB_IDENTITY_ROLE_NAME>',
```

```
  ProviderId: 'graph.facebook.com|www.amazon.com' // Omit this for Google
});

// Create a service object
var s3 = new AWS.S3;
```

Defina `WebIdentityToken` no retorno de chamada do SDK do provedor de identidade que contenha o token de acesso:

```
AWS.config.credentials.params.WebIdentityToken = accessToken;
```

Exemplos de identidades federadas da Web

Aqui estão alguns exemplos de como usar a identidade federada da web para obter credenciais no JavaScript do navegador. Esses exemplos devem ser executados em um esquema http:// ou https:// para garantir que o provedor de identidade possa redirecionar para o aplicativo.

Exemplo de Login with Amazon

O código a seguir mostra como usar Login with Amazon como um provedor de identidade.

```
<a href="#" id="login">
  <img border="0" alt="Login with Amazon"
    src="https://images-na.ssl-images-amazon.com/images/G/01/lwa/
btnLWA_gold_156x32.png"
    width="156" height="32" />
</a>
<div id="amazon-root"></div>
<script type="text/javascript">
  var s3 = null;
  var clientId = 'amzn1.application-oa2-client.1234567890abcdef'; // client ID
  var roleArn = 'arn:aws:iam::<AWS_ACCOUNT_ID>:role/<WEB_IDENTITY_ROLE_NAME>';

  window.onAmazonLoginReady = function() {
    amazon.Login.setClientId(clientId); // set client ID

    document.getElementById('login').onclick = function() {
      amazon.Login.authorize({scope: 'profile'}, function(response) {
        if (!response.error) { // logged in
          AWS.config.credentials = new AWS.WebIdentityCredentials({
            RoleArn: roleArn,
```

```
            ProviderId: 'www.amazon.com',
            WebIdentityToken: response.access_token
          });

          s3 = new AWS.S3();

          console.log('You are now logged in.');
        } else {
          console.log('There was a problem logging you in.');
        }
      });
    };
  };

  (function(d) {
    var a = d.createElement('script'); a.type = 'text/javascript';
    a.async = true; a.id = 'amazon-login-sdk';
    a.src = 'https://api-cdn.amazon.com/sdk/login1.js';
    d.getElementById('amazon-root').appendChild(a);
  })(document);
</script>
```

Exemplo de login do Facebook

O código a seguir mostra como usar o login do Facebook como um provedor de identidade:

```
<button id="login">Login</button>
<div id="fb-root"></div>
<script type="text/javascript">
var s3 = null;
var appId = '1234567890'; // Facebook app ID
var roleArn = 'arn:aws:iam::<AWS_ACCOUNT_ID>:role/<WEB_IDENTITY_ROLE_NAME>';

window.fbAsyncInit = function() {
  // init the FB JS SDK
  FB.init({appId: appId});

  document.getElementById('login').onclick = function() {
    FB.login(function (response) {
      if (response.authResponse) { // logged in
        AWS.config.credentials = new AWS.WebIdentityCredentials({
          RoleArn: roleArn,
          ProviderId: 'graph.facebook.com',
          WebIdentityToken: response.authResponse.accessToken
```

```
        });

        s3 = new AWS.S3;

        console.log('You are now logged in.');
      } else {
        console.log('There was a problem logging you in.');
      }
    });
  };
};

// Load the FB JS SDK asynchronously
(function(d, s, id){
   var js, fjs = d.getElementsByTagName(s)[0];
   if (d.getElementById(id)) {return;}
   js = d.createElement(s); js.id = id;
   js.src = "//connect.facebook.net/en_US/all.js";
   fjs.parentNode.insertBefore(js, fjs);
 }(document, 'script', 'facebook-jssdk'));
</script>
```

Exemplo de login do Google+

O código a seguir mostra como usar o login do Google+ como um provedor de identidade. O token de acesso usado para federação de identidades da web do Google é armazenado em `response.id_token`, em vez de `access_token` como outros provedores de identidade.

```
<span
  id="login"
  class="g-signin"
  data-height="short"
  data-callback="loginToGoogle"
  data-cookiepolicy="single_host_origin"
  data-requestvisibleactions="http://schemas.google.com/AddActivity"
  data-scope="https://www.googleapis.com/auth/plus.login">
</span>
<script type="text/javascript">
  var s3 = null;
  var clientID = '1234567890.apps.googleusercontent.com'; // Google client ID
  var roleArn = 'arn:aws:iam::<AWS_ACCOUNT_ID>:role/<WEB_IDENTITY_ROLE_NAME>';

  document.getElementById('login').setAttribute('data-clientid', clientID);
```

```
 function loginToGoogle(response) {
   if (!response.error) {
     AWS.config.credentials = new AWS.WebIdentityCredentials({
       RoleArn: roleArn, WebIdentityToken: response.id_token
     });

     s3 = new AWS.S3();

     console.log('You are now logged in.');
   } else {
     console.log('There was a problem logging you in.');
   }
 }

 (function() {
   var po = document.createElement('script'); po.type = 'text/javascript'; po.async =
true;
   po.src = 'https://apis.google.com/js/client:plusone.js';
   var s = document.getElementsByTagName('script')[0]; s.parentNode.insertBefore(po,
s);
 })();
</script>
```

Bloquear versões de API

Os serviços da AWS têm números de versão da API para acompanhar a compatibilidade da API. As versões da API em serviços da AWS são identificadas por uma string de data formatada em `YYYY-mm-dd`. Por exemplo, a versão da API atual do Amazon S3 é `2006-03-01`.

Recomendamos bloquear a versão da API de um serviço caso você dependa dela no código de produção. Isso pode isolar os aplicativos de alterações feitas no serviço resultantes de atualizações feitas no SDK. Se você não especificar uma versão da API ao criar objetos de serviço, o SDK usará a versão da API mais recente por padrão. Isso pode fazer o aplicativo fazer referência a uma API atualizada com alterações que afetam negativamente o aplicativo.

Para bloquear a versão da API usada em um serviço, passe o parâmetro `apiVersion` ao criar o objeto de serviço. No exemplo a seguir, um objeto de serviço `AWS.DynamoDB` recém-criado é bloqueado para a versão da API `2011-12-05`:

```
var dynamodb = new AWS.DynamoDB({apiVersion: '2011-12-05'});
```

Configure globalmente um conjunto de versões da API de serviço especificando o parâmetro `apiVersions` em `AWS.Config`. Por exemplo, para definir versões específicas das APIs DynamoDB e Amazon EC2 com a API Amazon Redshift atual, defina `apiVersions` da seguinte maneira:

```
AWS.config.apiVersions = {
  dynamodb: '2011-12-05',
  ec2: '2013-02-01',
  redshift: 'latest'
};
```

Obter versões de API

Para obter a versão da API de um serviço, consulte a seção Bloquear a versão da API na página de referência do serviço, como https://docs.aws.amazon.com/AWSJavaScriptSDK/latest/AWS/S3.html para Amazon S3.

Considerações sobre Node.js

Embora o código Node.js seja JavaScript, usar o AWS SDK for JavaScript em Node.js pode ser diferente de usar o SDK em scripts de navegador. Alguns métodos de API funcionam em Node.js, mas não em scripts de navegador e vice-versa. E usar com êxito algumas APIs depende da familiaridade com padrões de codificação Node.js comuns, como importar e usar outros módulos Node.js como o módulo `File System (fs)`.

Usar módulos de Node.js integrados

Node.js oferece um conjunto de módulos integrados que é possível usar sem instalá-los. Para usar esses módulos, crie um objeto com o método `require` para especificar o nome do módulo. Por exemplo, para incluir o módulo HTTP integrado, use o seguinte.

```
var http = require('http');
```

Invoque métodos do módulo como se eles fossem métodos desse objeto. Por exemplo, aqui está o código que lê um arquivo HTML.

```
// include File System module
var fs = require('fs');
// Invoke readFile method
fs.readFile('index.html', function(err, data) {
```

```
  if (err) {
    throw err;
  } else {
    // Successful file read
  }
});
```

Para obter uma lista completa de todos os módulos integrados fornecidos por Node.js, consulte Documentação do Node.js v6.11.1 no site do Node.js.

Usar pacotes NPM

Além dos módulos integrados, também é possível incluir e incorporar um código de terceiros de npm, o gerenciador de pacotes Node.js. Este é um repositório de pacotes de Node.js de código-aberto e uma interface de linha de comando para instalar esses pacotes. Para obter mais informações sobre npm e uma lista de pacotes disponíveis no momento, consulte https://www.npmjs.com. Também é possível saber mais sobre pacotes de Node.js adicionais a serem usados aqui no GitHub.

Um exemplo de um pacote npm que é possível usar com o AWS SDK for JavaScript é `browserify`. Para obter detalhes, consulte Compilar o SDK como uma dependência com Browserify. Outro exemplo é `webpack`. Para obter detalhes, consulte Empacotamento de aplicativos com o Webpack.

Tópicos

- Configurar maxSockets em Node.js
- Reutilizar conexões com Keep-alive no Node.js
- Configurar proxies para Node.js
- Registrar pacotes de certificados em Node.js

Configurar maxSockets em Node.js

Em Node.js, defina o número máximo de conexões por origem. Se `maxSockets` estiver definido, o cliente HTTP de baixo nível adicionará solicitações HTTP à fila e as atribuirá a soquetes à medida que forem disponibilizados.

Isso permite definir um limite máximo para o número de solicitações simultâneas para uma determinada origem por vez. Reduzir esse valor pode reduzir o número de erros de tempo limite ou de limitação recebidos. No entanto, isso também pode aumentar o uso da memória porque as solicitações serão enfileiradas até um soquete ser disponibilizado.

O exemplo a seguir mostra como definir `maxSockets` para todos os objetos de serviço criados. Este exemplo permite até 25 conexões simultâneas para cada endpoint de serviço.

```
var AWS = require('aws-sdk');
var https = require('https');
var agent = new https.Agent({
   maxSockets: 25
});

AWS.config.update({
   httpOptions:{
      agent: agent
   }
});
```

O mesmo pode ser feito por serviço.

```
var AWS = require('aws-sdk');
var https = require('https');
var agent = new https.Agent({
   maxSockets: 25
});

var dynamodb = new AWS.DynamoDB({
   apiVersion: '2012-08-10'
   httpOptions:{
      agent: agent
   }
});
```

Ao usar o padrão `https`, o SDK utiliza o valor `maxSockets` do `globalAgent`. Se o valor `maxSockets` não estiver definido ou for `Infinity`, o SDK assumirá um valor `maxSockets` de 50.

Para obter mais informações sobre como definir `maxSockets` em Node.js, consulte a documentação online de Node.js.

Reutilizar conexões com Keep-alive no Node.js

Por padrão, o agente Node.js HTTP/HTTPS padrão cria uma nova conexão TCP para cada nova solicitação. Para evitar o custo de estabelecer uma nova conexão, você pode reutilizar uma conexão existente.

Para operações de curta duração, como consultas do DynamoDB, a sobrecarga de latência da configuração de uma conexão TCP pode ser maior do que a própria operação. Além disso, como a criptografia em repouso do DynamoDB é integrada com o AWS KMS, você pode experimentar latências do banco de dados tendo que restabelecer novas entradas de cache do AWS KMS para cada operação.

A maneira mais fácil de configurar o SDK para JavaScript para reutilizar conexões TCP é definir a variável de ambiente `AWS_NODEJS_CONNECTION_REUSE_ENABLED` como 1. Esse recurso foi adicionado na versão 2.463.0.

Como alternativa, você pode definir a propriedade `keepAlive` de um agente HTTP ou HTTPS como `true`, conforme exibido no exemplo a seguir.

```
const AWS = require('aws-sdk');
// http or https
const http = require('http');
const agent = new http.Agent({
  keepAlive: true,
// Infinity is read as 50 sockets
  maxSockets: Infinity
});

AWS.config.update({
  httpOptions: {
    agent
  }
});
```

O exemplo a seguir mostra como definir `keepAlive` para apenas um cliente do DynamoDB:

```
const AWS = require('aws-sdk')
// http or https
const https = require('https');
const agent = new https.Agent({
  keepAlive: true
});

const dynamodb = new AWS.DynamoDB({
  httpOptions: {
    agent
  }
```

```
});
```

Se o `keepAlive` estiver habilitado, você também poderá definir o atraso inicial para pacotes TCP Keep-alive com `keepAliveMsecs` que, por padrão, é 1000 ms. Consulte a documentação do Node.js para obter detalhes.

Configurar proxies para Node.js

Se você não conseguir se conectar à Internet, o SDK para JavaScript oferecerá suporte ao uso de proxies HTTP ou HTTPS por meio de um agente HTTP de terceiros, como proxy-agent. Para instalar proxy-agent, digite o seguinte na linha de comando.

```
npm install proxy-agent --save
```

Se você optar por usar um proxy diferente, primeiro siga as instruções de instalação e configuração para esse proxy. Para usar esse ou outro proxy de terceiros no aplicativo, você deve definir a propriedade `httpOptions` de `AWS.Config` para especificar o proxy escolhido. Este exemplo mostra `proxy-agent`.

```
var AWS = require("aws-sdk");
var ProxyAgent = require('proxy-agent').ProxyAgent;
AWS.config.update({
  httpOptions: { agent: new ProxyAgent('http://internal.proxy.com') }
});
```

Para obter mais informações sobre outras bibliotecas de proxy, consulte npm, o gerenciador de pacotes Node.js.

Registrar pacotes de certificados em Node.js

Os armazenamentos de confiança padrão de Node.js incluem os certificados necessários para acessar os serviços da AWS. Em alguns casos, pode ser preferível incluir apenas um conjunto específico de certificados.

Neste exemplo, um certificado específico em disco é usado para criar um `https.Agent` que rejeita conexões, a menos que o certificado designado seja fornecido. O `https.Agent` recém-criado acaba sendo usado para atualizar a configuração do SDK.

```
var fs = require('fs');
```

```
var https = require('https');
var certs = [
  fs.readFileSync('/path/to/cert.pem')
];

AWS.config.update({
  httpOptions: {
    agent: new https.Agent({
      rejectUnauthorized: true,
      ca: certs
    })
  }
});
```

Considerações sobre o script de navegador

Os tópicos a seguir descrevem considerações especiais para usar o AWS SDK for JavaScript em scripts de navegador.

Tópicos

- Criar o SDK para navegadores
- Cross-Origin Resource Sharing (CORS, Compartilhamento de recursos de origem cruzada)

Criar o SDK para navegadores

O SDK para JavaScript é fornecido como um arquivo JavaScript com suporte incluído para um conjunto padrão de serviços. Esse arquivo costuma ser carregado em scripts de navegador que usam uma tag `<script>` que faz referência ao pacote do SDK hospedado. No entanto, talvez você precise de suporte para serviços diferentes do conjunto padrão ou que precisem personalizar o SDK.

Se trabalhar com o SDK fora de um ambiente que imponha CORS no navegador e quiser acesso a todos os serviços fornecidos pelo SDK para JavaScript, você poderá criar uma cópia personalizada do SDK localmente clonando o repositório e executando as mesmas ferramentas de criação que compilam a versão hospedada padrão do SDK. As seções a seguir descrevem as etapas para compilar o SDK com serviços extras e versões da API.

Tópicos

- Usar o SDK Builder para criar o SDK para JavaScript

- Usar a CLI para criar o SDK para JavaScript
- Criar serviços específicos e versões de API
- Compilar o SDK como uma dependência com Browserify

Usar o SDK Builder para criar o SDK para JavaScript

A maneira mais fácil de criar a própria compilação do AWS SDK for JavaScript é usar o aplicativo web de criador de SDKs em https://sdk.amazonaws.com/builder/js. Use o criador de SDKs para especificar serviços e as versões da API a serem incluídos na compilação.

Escolha Select all services (Selecionar todos os serviços) ou Select all services (Selecionar serviços padrão) como um ponto de partida de onde é possível adicionar ou remover serviços. Escolha Development (Desenvolvimento) para obter um código mais legível ou Minified (Minimizado) para criar uma compilação mini a ser implantada. Depois de escolher os serviços e as versões a serem incluídos, escolha Build (Criar) para criar e fazer download do SDK personalizado.

Usar a CLI para criar o SDK para JavaScript

Para compilar o SDK para JavaScript usando a AWS CLI, você primeiro precisa clonar o repositório Git que contém a fonte do SDK. Você deve ter Git e Node.js instalados no computador.

Primeiro, clone o repositório do GitHub e altere o diretório para o diretório:

```
git clone https://github.com/aws/aws-sdk-js.git
cd aws-sdk-js
```

Depois de clonar o repositório, faça download dos módulos de dependência do SDK e da ferramenta de compilação:

```
npm install
```

Já é possível criar uma versão em pacote do SDK.

Criar a partir da linha de comando

A ferramenta de construção está em `dist-tools/browser-builder.js`. Execute este script digitando:

```
node dist-tools/browser-builder.js > aws-sdk.js
```

Este comando cria o arquivo aws-sdk.js. Este arquivo está descompactado. Por padrão, esse pacote só inclui o conjunto padrão de serviços.

Minimizar saída de compilação

Para reduzir a quantidade de dados na rede, os arquivos JavaScript podem ser compactados por meio de um processo chamado minimização. A minimização remove comentários, espaços desnecessários e outros caracteres que auxiliam na legibilidade humana, mas que não afetam a execução do código. A ferramenta de criação pode produzir uma saída descompactada ou minimizada. Para minimizar a saída de criação, defina a variável de ambiente `MINIFY`:

```
MINIFY=1 node dist-tools/browser-builder.js > aws-sdk.js
```

Criar serviços específicos e versões de API

É possível selecionar quais serviços compilar no SDK. Para selecionar serviços, especifique os nomes de serviço, delimitados por vírgulas, como parâmetros. Por exemplo, para criar apenas Amazon S3 e Amazon EC2, use o seguinte comando:

```
node dist-tools/browser-builder.js s3,ec2 > aws-sdk-s3-ec2.js
```

Você também pode selecionar versões da API específicas da criação de serviços adicionando o nome da versão após o nome do serviço. Por exemplo, para criar ambas as versões da API do Amazon DynamoDB, use o seguinte comando:

```
node dist-tools/browser-builder.js dynamodb-2011-12-05,dynamodb-2012-08-10
```

Identificadores de serviço e versões de API estão disponíveis nos arquivos de configuração específicos do serviço em https://github.com/aws/aws-sdk-js/tree/master/apis.

Criar todos os serviços

Você pode criar todos os serviços e versões de API incluindo o parâmetro `all`:

```
node dist-tools/browser-builder.js all > aws-sdk-full.js
```

Criar serviços específicos

Para personalizar o conjunto selecionado de serviços incluídos na compilação, passe a variável de ambiente `AWS_SERVICES` para o comando Browserify que contém a lista de serviços que você deseja. O exemplo a seguir cria os serviços Amazon EC2, Amazon S3 e DynamoDB.

```
$ AWS_SERVICES=ec2,s3,dynamodb browserify index.js > browser-app.js
```

Compilar o SDK como uma dependência com Browserify

Node.js tem um mecanismo baseado no módulo para incluir código e funcionalidade de desenvolvedores de terceiros. Essa abordagem modular não tem suporte nativo do JavaScript em execução em navegadores da web. No entanto, com uma ferramenta chamada Browserify, é possível usar a abordagem do módulo Node.js e usar os módulos escritos para Node.js no navegador. Browserify cria as dependências de módulo para um script de navegador em um único arquivo JavaScript autocontido que é possível usar no navegador.

É possível compilar o SDK como uma dependência de biblioteca para qualquer script de navegador usando Browserify. Por exemplo, o seguinte código Node.js exige o SDK:

```
var AWS = require('aws-sdk');
var s3 = new AWS.S3();
s3.listBuckets(function(err, data) { console.log(err, data); });
```

Este código de exemplo pode ser compilado em uma versão compatível com navegador usando Browserify:

```
$ browserify index.js > browser-app.js
```

O aplicativo, inclusive as dependências do SDK, é disponibilizado no navegador por meio de `browser-app.js`.

Para obter mais informações sobre o Browserify, consulte o site do Browserify.

Cross-Origin Resource Sharing (CORS, Compartilhamento de recursos de origem cruzada)

O compartilhamento de recursos de origem cruzada, ou CORS, é um recurso de segurança de navegadores da web modernos. Isso permite que navegadores da web negociem quais domínios

podem fazer solicitações de sites ou serviços externos. CORS é uma consideração importante durante o desenvolvimento de aplicativos de navegador com o AWS SDK for JavaScript porque a maioria das solicitações de recursos é enviada para um domínio externo, como o endpoint de um serviço da web. Se o ambiente do JavaScript impuser segurança CORS, você deverá configurar CORS com o serviço.

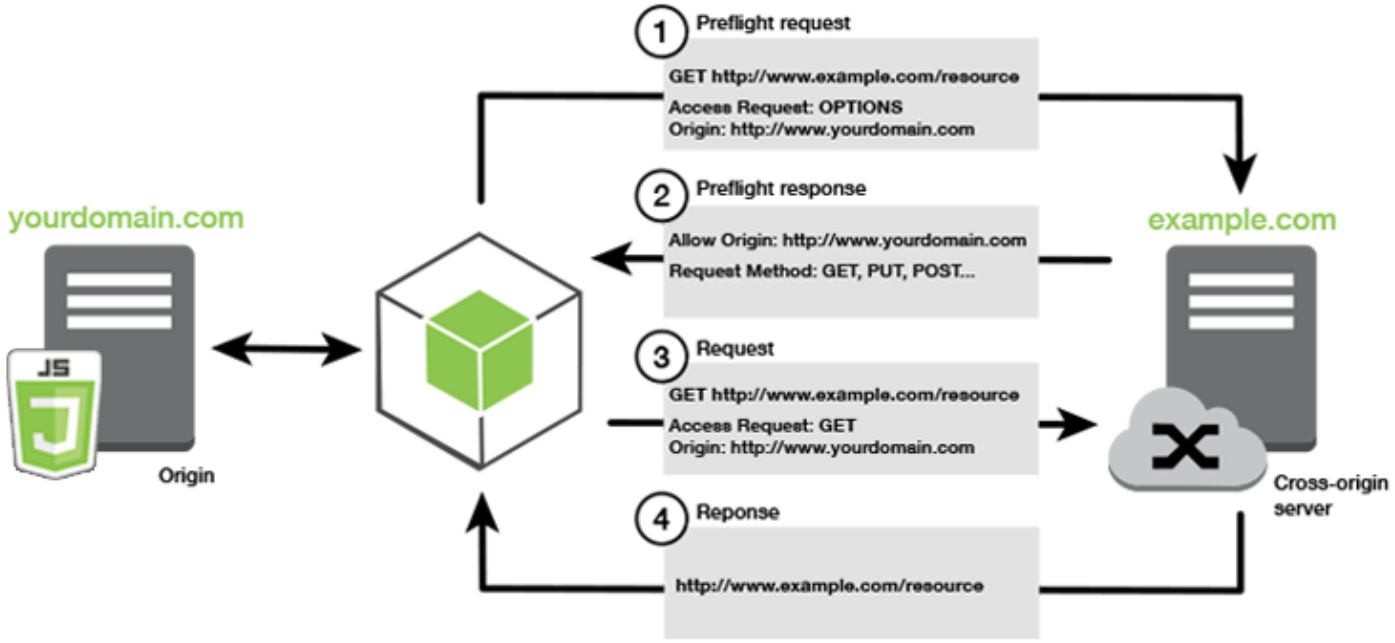
O CORS determina se é necessário permitir ou não o compartilhamento de recursos em uma solicitação entre origens com base em:

- O domínio específico que faz a solicitação
- O tipo de solicitação HTTP feita (GET, PUT, POST, DELETE etc.)

Como CORS funciona

No caso mais simples, o script de navegador faz uma solicitação GET para um recurso de um servidor em outro domínio. Dependendo da configuração CORS desse servidor, se a solicitação for de um domínio autorizado para enviar solicitações GET, o servidor de origem cruzada responderá retornando o recurso solicitado.

Se o domínio solicitante ou o tipo de solicitação HTTP não estiver autorizado, a solicitação será negada. No entanto, CORS possibilita simular a solicitação antes de enviá-la efetivamente. Neste caso, uma solicitação de simulação é feita em que a operação de solicitação de acesso `OPTIONS` é enviada. Se o servidor de origem cruzada da configuração CORS conceder acesso ao domínio solicitante, o servidor reenviará uma resposta de simulação que lista todos os tipos de solicitação HTTP que o domínio solicitante pode fazer no recurso solicitado.



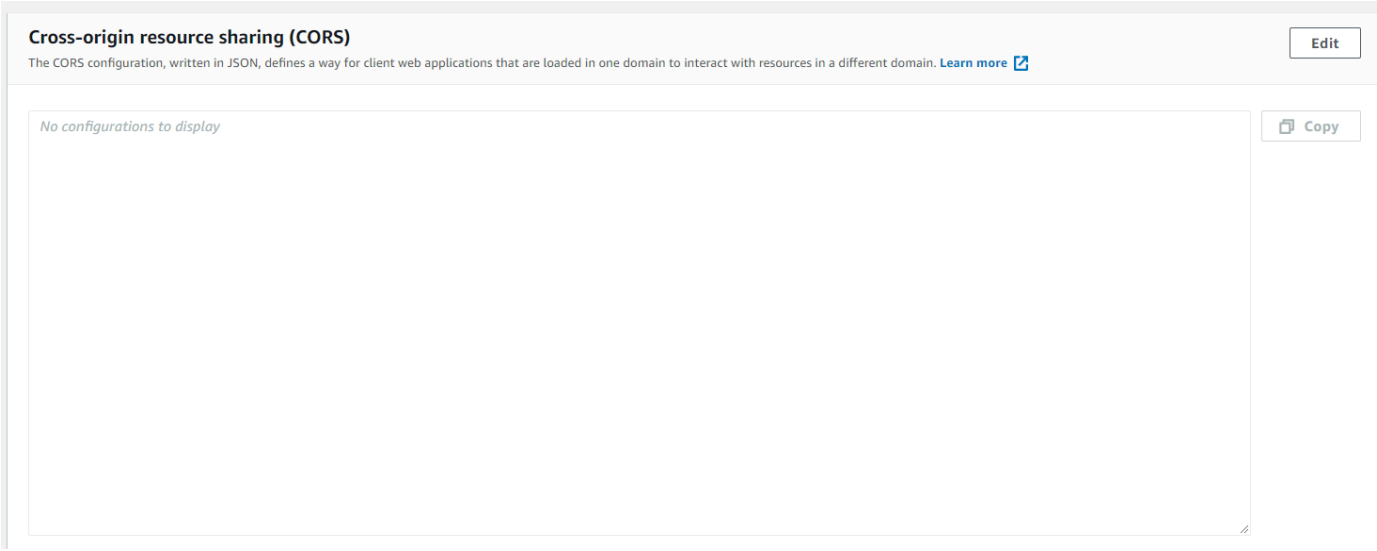
A configuração de CORS é obrigatória

Os buckets do Amazon S3 exigem a configuração de CORS para realizar operações neles. Em alguns ambientes JavaScript, o CORS talvez não seja imposto e, por isso, configurar o CORS é algo desnecessário. Por exemplo, se você hospedar o aplicativo de um bucket do e acessar recursos de `*.s3.amazonaws.com` ou algum outro endpoint específico, as solicitações não acessarão um domínio externo. Por isso, essa configuração não exige CORS. Nesse caso, o CORS continua sendo usado em serviços que não sejam o Amazon S3.

Configuração de CORS para um bucket do Amaznon S3

Você pode configurar um bucket do Amazon S3 para usar o CORS no console do Amazon S3.

1. No console do Amazon S3, selecione o bucket criado para ser editado.
2. Selecione a guia Permissões e role para baixo até o painel Compartilhamento de recursos de origem cruzada (CORS).



3. Escolha Edit e digite a configuração do CORS no CORS Configuration Editor (Editor de configuração do CORS), e escolha Save (Salvar).

A configuração do CORS é um arquivo XML que contém uma série de regras dentro de um `<CORSRule>`. Uma configuração pode ter até 100 regras. Uma regra é definida por uma das seguintes tags:

- `<AllowedOrigin>`, que especifica as origens de domínio permitidas para fazer solicitações entre domínios.
- `<AllowedMethod>`, que especifica um tipo de solicitação permitida (GET, PUT, POST, DELETE, HEAD) em solicitações entre domínios.
- `<AllowedHeader>`, que especifica os cabeçalhos permitidos em uma solicitação de simulação.

Para configurações de exemplo, consulte Como configuro o CORS no meu bucket? no Guia de usuário do Amazon Simple Storage Service.

Exemplo de configuração do CORS

O exemplo de configuração CORS a seguir permite a um usuário exibir, adicionar, remover ou atualizar objetos dentro de um bucket no domínio `example.org`. Entretanto, é recomendado o escopo `<AllowedOrigin>` para o domínio do site. Especifique `"*"` para permitir qualquer origem.

> **Important**
>
> No novo console do S3, a configuração CORS deve ser JSON.

XML

```
<?xml version="1.0" encoding="UTF-8"?>
<CORSConfiguration xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
  <CORSRule>
    <AllowedOrigin>https://example.org</AllowedOrigin>
    <AllowedMethod>HEAD</AllowedMethod>
    <AllowedMethod>GET</AllowedMethod>
    <AllowedMethod>PUT</AllowedMethod>
    <AllowedMethod>POST</AllowedMethod>
    <AllowedMethod>DELETE</AllowedMethod>
    <AllowedHeader>*</AllowedHeader>
    <ExposeHeader>ETag</ExposeHeader>
    <ExposeHeader>x-amz-meta-custom-header</ExposeHeader>
  </CORSRule>
</CORSConfiguration>
```

JSON

```
[
    {
        "AllowedHeaders": [
            "*"
        ],
        "AllowedMethods": [
            "HEAD",
            "GET",
            "PUT",
            "POST",
            "DELETE"
        ],
        "AllowedOrigins": [
            "https://www.example.org"
        ],
        "ExposeHeaders": [
             "ETag",
             "x-amz-meta-custom-header"]
```

```
    }
]
```

Essa configuração não autoriza o usuário para executar ações no bucket. Ela permite que o modelo de segurança do navegador faça uma solicitação ao Amazon S3. As permissões devem ser configuradas por meio de permissões do bucket ou permissões do perfil do IAM.

Use `ExposeHeader` para permitir que os cabeçalhos de resposta de leitura do SDK sejam retornados do Amazon S3. Por exemplo, se quiser ler o cabeçalho `ETag` de um PUT ou de um multipart upload, você precisará incluir a tag `ExposeHeader` na configuração, conforme mostrado no exemplo anterior. O SDK só pode acessar cabeçalhos expostos por meio da configuração do CORS. Se você definir metadados no objeto, os valores serão retornados como cabeçalhos com o prefixo `x-amz-meta-`, como `x-amz-meta-my-custom-header`, e também deverão ser expostos da mesma maneira.

Empacotamento de aplicativos com o Webpack

Os aplicativos web nos scripts do navegador ou o uso pelo Node.js dos módulos do código criam dependências. Esses módulos de código podem ter dependências próprias, o que resulta em uma coleção de módulos interligados que seu aplicativo exige para funcionar. Para gerenciar dependências, você pode usar um agrupador de módulos, como Webpack.

O agrupador de módulos Webpack analisa o código do seu aplicativo, procurando por instruções `import` ou `require`, para criar pacotes que contenham todos os ativos de que o seu aplicativo precisa para que os ativos sejam facilmente apresentados em uma página da web. O SDK para JavaScript pode ser incluído no Webpack como uma das dependências para incluir no pacote de saída.

Para obter mais informações sobre o Webpack, consulte o módulo do agrupador de módulos Webpack no GitHub.

Instalar o Webpack

Para instalar o agrupador de módulos Webpack, primeiro você deve instalar o npm, o gerenciador de pacotes do Node.js. Digite o comando a seguir para instalar a CLI do Webpack e o módulo do JavaScript.

```
npm install webpack
```

Pode ser necessário também instalar um plugin do Webpack que permita o carregamento de arquivos JSON. Digite o comando a seguir para instalar o plug-in do carregador JSON.

```
npm install json-loader
```

Configurar o Webpack

Por padrão, o Webpack busca por um arquivo JavaScript chamado `webpack.config.js` no diretório raiz do projeto. Esse arquivo especifica as opções de configuração. Eis um exemplo de um arquivo de configuração do `webpack.config.js`.

```
// Import path for resolving file paths
var path = require('path');
module.exports = {
  // Specify the entry point for our app.
  entry: [
    path.join(__dirname, 'browser.js')
  ],
  // Specify the output file containing our bundled code
  output: {
    path: __dirname,
    filename: 'bundle.js'
  },
  module: {
    /**
      * Tell webpack how to load 'json' files.
      * When webpack encounters a 'require()' statement
      * where a 'json' file is being imported, it will use
      * the json-loader.
      */
    loaders: [
      {
        test: /\.json$/,
        loaders: ['json']
      }
    ]
  }
}
```

Neste exemplo, `browser.js` é especificado como ponto de entrada. O ponto de entrada é o arquivo que o webpack usa para iniciar a pesquisa por módulos importados. O nome do arquivo da saída

é especificado como `bundle.js`. Esse arquivo de saída conterá tudo do JavaScript de que o aplicativo precisa para ser executado. Se o código especificado no ponto de entrada importar ou exigir outros módulos, como o SDK para JavaScript, esse código será incluído sem a necessidade de especificá-lo na configuração.

A configuração do plugin `json-loader` que foi instalado anteriormente especifica ao Webpack como importar arquivos JSON. Por padrão, o Webpack só é compatível com JavaScript, mas usa carregadores para adicionar suporte à importação de outros tipos de arquivo. Como o SDK para JavaScript faz amplo uso de arquivos JSON, o Webpack lançará um erro ao gerar o pacote se o `json-loader` não estiver incluído.

Executar o Webpack

Para criar um aplicativo para usar o Webpack, adicione o seguinte ao objeto `scripts` no seu arquivo `package.json`.

```
"build": "webpack"
```

Veja a seguir um exemplo de `package.json` que demonstra a adição do Webpack.

```
{
  "name": "aws-webpack",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "build": "webpack"
  },
  "author": "",
  "license": "ISC",
  "dependencies": {
    "aws-sdk": "^2.6.1"
  },
  "devDependencies": {
    "json-loader": "^0.5.4",
    "webpack": "^1.13.2"
  }
}
```

Para criar seu aplicativo, digite o comando a seguir.

```
npm run build
```

O agrupador de módulos Webpack gera o arquivo JavaScript que você especificou no diretório raiz do projeto.

Usar o pacote Webpack

Para usar o pacote no script de um navegador, você pode incorporar o pacote usando uma tag `<script>`, conforme mostrado no exemplo a seguir.

```
<!DOCTYPE html>
<html>
    <head>
        <title>AWS SDK with webpack</title>
    </head>
    <body>
        <div id="list"></div>
        <script src="bundle.js"></script>
    </body>
</html>
```

Importar serviços individuais

Um dos benefícios do Webpack é que ele analisa as dependências do seu código e agrupa somente o código de que seu aplicativo precisa. Se você estiver usando o SDK para JavaScript, agrupar apenas as partes do SDK de fato usadas pelo seu aplicativo pode reduzir consideravelmente o tamanho da saída do Webpack.

Considere o seguinte exemplo do código usado para criar um objeto de serviço do Amazon S3.

```
// Import the AWS SDK
var AWS = require('aws-sdk');

// Set credentials and Region
// This can also be done directly on the service client
AWS.config.update({region: 'us-west-1', credentials: {YOUR_CREDENTIALS}});

var s3 = new AWS.S3({apiVersion: '2006-03-01'});
```

A função `require()` especifica o SDK inteiro. Um pacote do Webpack gerado com esse código incluiria todo o SDK, mas nem todo o SDK é necessário quando somente a classe de cliente Amazon

S3 é usada. O tamanho do pacote seria substancialmente menor se apenas a parte do SDK de que você precisa para o serviço do Amazon S3 fosse incluída. Nem a configuração exige todo o SDK, pois você pode definir os dados de configuração no objeto de serviço do Amazon S3.

Aqui está o mesmo código quando inclui somente a parte do Amazon S3 do SDK.

```
// Import the Amazon S3 service client
var S3 = require('aws-sdk/clients/s3');

// Set credentials and Region
var s3 = new S3({
    apiVersion: '2006-03-01',
    region: 'us-west-1',
    credentials: {YOUR_CREDENTIALS}
  });
```

Empacotamento para o Node.js

Você pode usar o Webpack para gerar pacotes executados no Node.js ao especificá-los como destino na configuração.

```
target: "node"
```

Isso é útil ao executar um aplicativo do Node.js em um ambiente no qual o espaço em disco é limitado. Aqui está um exemplo de configuração do `webpack.config.js` com o Node.js especificado como o destino de saída.

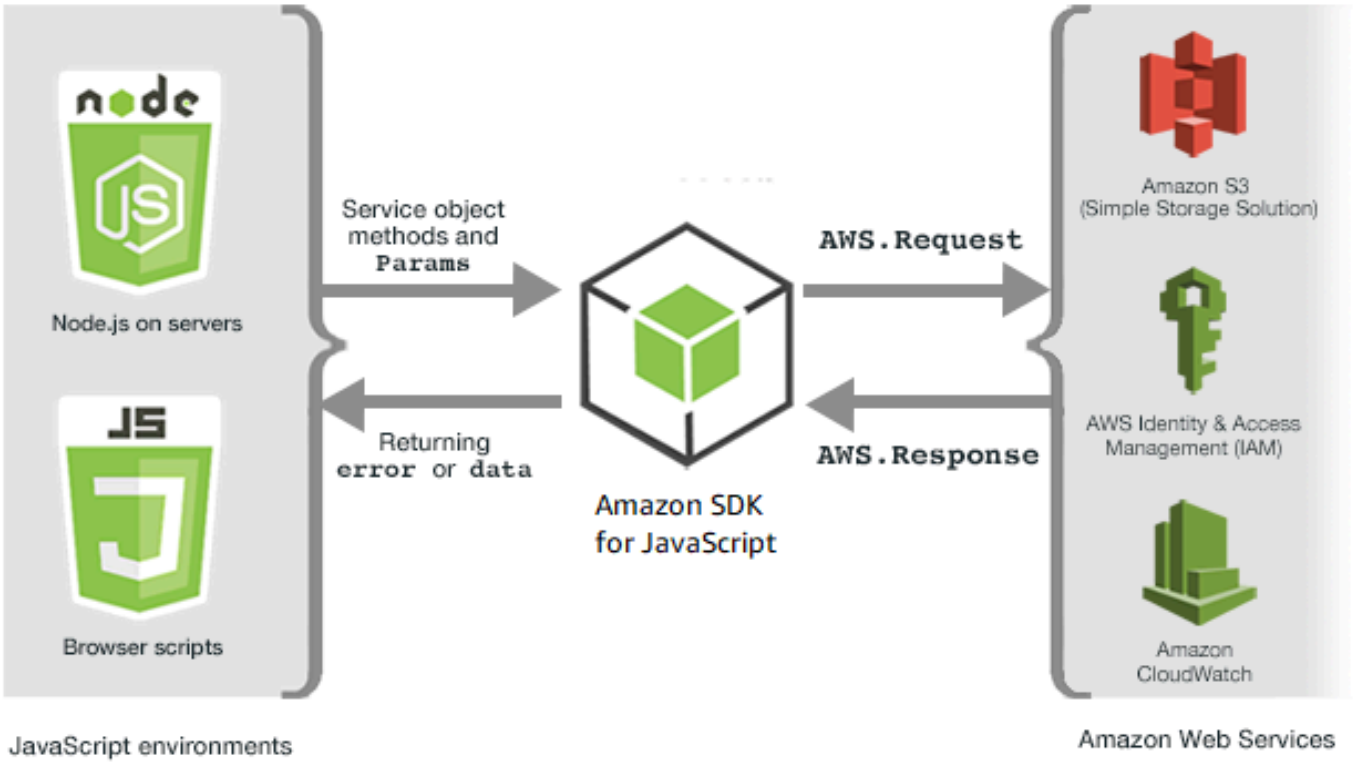
```
// Import path for resolving file paths
var path = require('path');
module.exports = {
  // Specify the entry point for our app
  entry: [
    path.join(__dirname, 'node.js')
  ],
  // Specify the output file containing our bundled code
  output: {
    path: __dirname,
    filename: 'bundle.js'
  },
  // Let webpack know to generate a Node.js bundle
  target: "node",
```

```
  module: {
    /**
      * Tell webpack how to load JSON files.
      * When webpack encounters a 'require()' statement
      * where a JSON file is being imported, it will use
      * the json-loader
      */
    loaders: [
      {
        test: /\.json$/,
        loaders: ['json']
      }
    ]
  }
}
```

Usar serviços no SDK da para JavaScript

O AWS SDK for JavaScript dá acesso aos serviços a que oferece suporte por meio de uma série de classes de clientes. Com base nessas classes de clientes, você cria objetos de interface de serviço, comumente chamados de objetos de serviço. Cada serviço compatível da AWS tem uma ou mais classes de clientes que oferecem APIs de baixo nível para uso dos recursos de serviço. Por exemplo: as APIs do Amazon DynamoDB estão disponíveis por meio da classe `AWS.DynamoDB`.

Os serviços expostos por meio do SDK para JavaScript seguem o padrão solicitação-resposta para trocar mensagens com aplicativos de chamada. Neste padrão, o código que invoca um serviço envia uma solicitação HTTP/HTTPS a um endpoint para o serviço. A solicitação contém os parâmetros necessários para invocar com sucesso o recurso específico que está sendo chamado. O serviço que é invocado gera uma resposta, que é enviada de volta ao solicitante. A resposta contém dados, caso a operação tenha tido sucesso, ou informações de erro, caso a operação não tenha tido sucesso.



Invocar um serviço da AWS inclui todo o ciclo de vida de solicitação e resposta de uma operação em um objeto de serviço, incluindo quaisquer novas tentativas. A solicitação é encapsulada no SDK pelo objeto `AWS.Request`. A resposta está encapsulada no SDK pelo objeto `AWS.Response`, que é fornecido ao solicitante por meio de uma das várias técnicas, como uma função de retorno de chamada ou uma promessa do JavaScript.

Tópicos

- Criar e chamar objetos de serviço
- Registrar em log as chamadas a AWS SDK for JavaScript
- Chamar serviços assincronamente
- Usar o objeto de resposta
- Trabalhar com o JSON

Criar e chamar objetos de serviço

A API JavaScript oferece suporte para a maioria dos serviços da AWS disponíveis. Cada classe de serviço na API JavaScript fornece acesso a cada chamada de API em seu serviço. Para obter mais informações sobre classes de serviço, operações e parâmetros na API JavaScript, consulte a referência da API.

Ao usar o SDK em Node.js, você adiciona o pacote do SDK para seu aplicativo usando `require`, que fornece suporte a todos os serviços atuais.

```
var AWS = require('aws-sdk');
```

Ao usar o SDK com o JavaScript do navegador, você carrega o pacote do SDK para os scripts do navegador usando o pacote do SDK hospedado pela AWS. Para carregar o pacote do SDK, adicione o seguinte elemento `<script>`:

```
<script src="https://sdk.amazonaws.com/js/aws-sdk-SDK_VERSION_NUMBER.min.js"></script>
```

Para encontrar o SDK_VERSION_NUMBER atual, consulte a Referência da API para o SDK para JavaScript no Guia de referência da API. AWS SDK for JavaScript

O pacote do SDK hospedado padrão oferece suporte a um subconjunto dos serviços da AWS disponíveis. Para obter uma lista dos serviços padrão no pacote do SDK hospedado para o navegador, consulte Serviços compatíveis na Referência de API. Você poderá usar o SDK com outros serviços se a verificação de segurança do CORS estiver desabilitada. Nesse caso, você pode criar uma versão personalizada do SDK para incluir os serviços adicionais de que precisa. Para obter mais informações sobre a criação de uma versão personalizada do SDK, consulte Criar o SDK para navegadores.

Exigir serviços individuais

Exigir o SDK para JavaScript, como mostrado anteriormente, inclui todo o SDK no seu código. Como alternativa, você pode optar por solicitar apenas os serviços individuais usados pelo seu código. Considere o código a seguir usado para criar um objeto de serviço do Amazon S3.

```
// Import the AWS SDK
var AWS = require('aws-sdk');

// Set credentials and Region
// This can also be done directly on the service client
AWS.config.update({region: 'us-west-1', credentials: {YOUR_CREDENTIALS}});

var s3 = new AWS.S3({apiVersion: '2006-03-01'});
```

No exemplo anterior, a função `require` especifica todo o SDK. A quantidade de código para transporte pela rede, bem como a sobrecarga de memória do seu código, seriam substancialmente menores se for incluída apenas a parte do SDK de que você precisa para o serviço do Amazon S3. Para exigir um serviço individual, chame a função `require`, conforme mostrado, incluindo o construtor do serviço, com todas as letras minúsculas.

```
require('aws-sdk/clients/SERVICE');
```

Veja como é o código para criar o objeto de serviço do Amazon S3 quando inclui apenas a parte do SDK referente ao Amazon S3.

```
// Import the Amazon S3 service client
var S3 = require('aws-sdk/clients/s3');

// Set credentials and Region
var s3 = new S3({
    apiVersion: '2006-03-01',
    region: 'us-west-1',
    credentials: {YOUR_CREDENTIALS}
  });
```

Você ainda pode acessar o namespace global da AWS sem todos os serviços anexados a ele.

```
require('aws-sdk/global');
```

Essa é uma técnica útil ao aplicar a mesma configuração em vários serviços individuais para, por exemplo, fornecer as mesmas credenciais a todos os serviços. Exigir serviços individuais deve reduzir o tempo de carregamento e o consumo de memória em Node.js. Quando feito juntamente com uma ferramenta de empacotamento, como Browserify ou webpack, exigir serviços individuais faz com que o SDK tenha uma fração do tamanho total. Isso ajuda com ambientes com restrição de memória ou espaço em disco, como um dispositivo com IoT ou uma função do Lambda.

Criar objetos de serviço

Para acessar recursos de serviços por meio da API JavaScript, primeiro você deve criar um objeto de serviço e, por meio dele, acessar uma série de recursos fornecidos pela classe de cliente subjacente. Em geral, há uma classe de cliente para cada serviço. No entanto, alguns serviços dividem o acesso a seus recursos entre várias classes de cliente.

Para usar um recurso, você deve criar uma instância da classe que fornece acesso a ele. O exemplo a seguir mostra como criar um objeto de serviço para o DynamoDB a partir da classe de cliente do `AWS.DynamoDB`.

```
var dynamodb = new AWS.DynamoDB({apiVersion: '2012-08-10'});
```

Por padrão, o objeto de serviço vem configurado com as configurações globais também usadas para configurar o SDK. No entanto, você pode configurar o objeto de serviço com dados de configuração do tempo de execução específicos desse objeto de serviço. Dados de configuração específicos do serviço são aplicados após a aplicação das configurações globais.

No exemplo a seguir, um objeto de serviço do Amazon EC2 é criado com a configuração de uma região específica, mas usando a configuração global.

```
var ec2 = new AWS.EC2({region: 'us-west-2', apiVersion: '2014-10-01'});
```

Além de oferecer suporte a configurações específicas do serviço aplicadas a um objeto de serviço individual, você também pode aplicar configurações específicas do serviço a todos os objetos de serviço recém-criados de uma determinada classe. Por exemplo, para configurar todos os objetos de serviço criados com base na classe Amazon EC2 para usar a região (`us-west-2`) Oeste dos EUA (Oregon), adicione o seguinte ao objeto de configuração global `AWS.config`.

```
AWS.config.ec2 = {region: 'us-west-2', apiVersion: '2016-04-01'};
```

Bloquear a versão da API de um objeto de serviço

Você pode bloquear um objeto de serviço para usar uma determinada versão da API de um serviço ao especificar a opção `apiVersion` quando for criar o objeto. No exemplo a seguir, é criado um objeto de serviço do DynamoDB e ele está bloqueado para uma determinada versão da API.

```
var dynamodb = new AWS.DynamoDB({apiVersion: '2011-12-05'});
```

Para obter mais informações sobre como bloquear a versão da API de um objeto de serviço, consulte Bloquear versões de API.

Especificar os parâmetros do objeto de serviço

Ao chamar um método de um objeto de serviço, passe os parâmetros em JSON, conforme exigido pela API. Por exemplo, no Amazon S3, para obter um objeto para bucket e chave especificados, passe os seguintes parâmetros para o método . Para obter mais informações sobre como passar os parâmetros JSON, consulte Trabalhar com o JSON.

```
s3.getObject({Bucket: 'bucketName', Key: 'keyName'});
```

Para obter mais informações sobre parâmetros do Amazon S3, consulte `Class: AWS.S3` na Referência a APIs.

Além disso, você pode vincular valores a parâmetros individuais ao criar um objeto de serviço usando o parâmetro `params`. O valor do parâmetro `params` de objetos de serviço é um mapa que especifica um ou mais dos valores de parâmetro definidos pelo objeto de serviço. O exemplo a seguir mostra o parâmetro `Bucket` de um objeto de serviço do que está sendo vinculado a um bucket chamado `myBucket`.

```
var s3bucket = new AWS.S3({params: {Bucket: 'myBucket'}, apiVersion: '2006-03-01' });
```

Ao vincular o objeto de serviço a um bucket, o objeto de serviço `s3bucket` trata o valor do parâmetro `myBucket` como padrão, que não precisa mais ser especificado para operações subsequentes. Quaisquer valores de parâmetro vinculados são ignorados ao usar o objeto para operações em que o valor do parâmetro não é aplicável. Você pode substituir esse parâmetro vinculado ao fazer chamadas no objeto de serviço, para tanto especificando um novo valor.

```
var s3bucket = new AWS.S3({ params: {Bucket: 'myBucket'}, apiVersion: '2006-03-01' });
```

```
s3bucket.getObject({Key: 'keyName'});
// ...
s3bucket.getObject({Bucket: 'myOtherBucket', Key: 'keyOtherName'});
```

Detalhes sobre os parâmetros disponíveis para cada método são encontrados na referência da API.

Registrar em log as chamadas a AWS SDK for JavaScript

O AWS SDK for JavaScript é instrumentado com um registrador integrado para que você possa registrar chamadas de API feitas com o SDK para JavaScript.

Para ativar o registrador e imprimir entradas de log no console, adicione a instrução a seguir ao seu código.

```
AWS.config.logger = console;
```

Aqui está um exemplo da saída do log.

```
[AWS s3 200 0.185s 0 retries] createMultipartUpload({ Bucket: 'js-sdk-test-bucket',
 Key: 'issues_1704' })
```

Usar um registrador de terceiros

Você também pode usar um registrador de terceiros, desde que tenha as operações `log()` ou `write()` para gravar em um arquivo de log ou servidor. Você deve instalar e configurar o registrador personalizado de acordo com as instruções, para que possa usá-lo com o JavaScript.

Um registrador que você pode usar em scripts do navegador ou em Node.js é o logplease. No Node.js, você pode configurar o logplease para gravar entradas em um arquivo de log. Você também pode usá-lo com o webpack.

Ao usar um registrador de terceiros, defina todas as opções antes de atribuir o logger ao `AWS.Config.logger`. Por exemplo, a seguir está especificado um arquivo de log externo e definido o nível de log para logplease

```
// Require AWS Node.js SDK
const AWS = require('aws-sdk')
// Require logplease
const logplease = require('logplease');
```

```
// Set external log file option
logplease.setLogfile('debug.log');
// Set log level
logplease.setLogLevel('DEBUG');
// Create logger
const logger = logplease.create('logger name');
// Assign logger to SDK
AWS.config.logger = logger;
```

Para obter mais informações sobre o logplease, consulte registrador logplease simples do logplease no GitHub.

Chamar serviços assincronamente

Todas as solicitações feitas por meio do SDK são assíncronas. É importante ter isso em mente ao gravar scripts do navegador. O JavaScript executado em um navegador normalmente tem apenas um único thread de execução. Depois de fazer uma chamada assíncrona para um serviço da AWS, o script do navegador continua em execução e, nesse processo, pode tentar executar um código que depende do resultado assíncrono antes que tenha retorno.

Fazer chamadas assíncronas para gerenciar um serviço da AWS inclui gerenciar essas chamadas para que seu código não tente usar dados antes de serem disponibilizados. Os tópicos desta seção explicam a necessidade de gerenciar chamadas assíncronas e detalha diferentes técnicas que você pode usar para gerenciá-las.

Tópicos

- Gerenciar chamadas assíncronas
- Usar uma função de retorno de chamada anônimo
- Usar um listener de evento do objeto de solicitação
- Usar async/await
- Usar as promessas do JavaScript

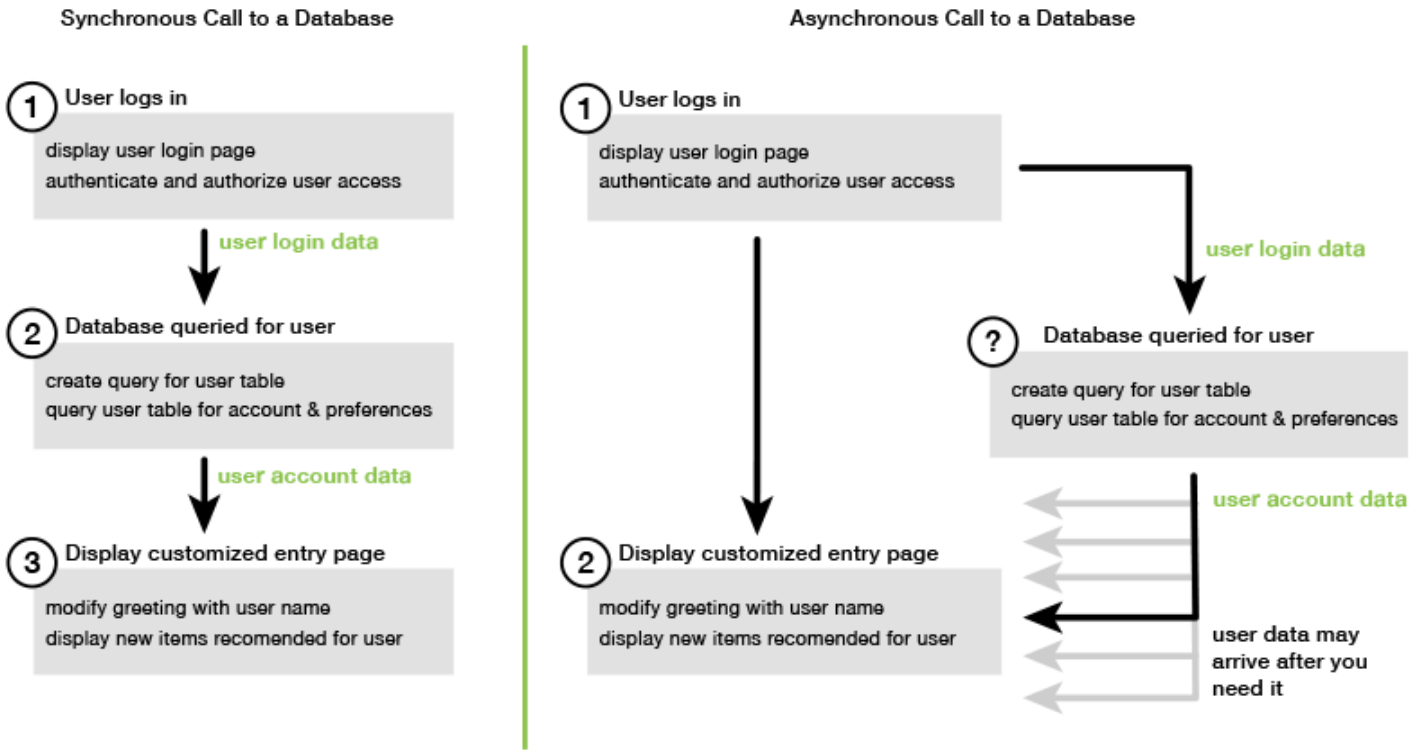
Gerenciar chamadas assíncronas

Por exemplo, a página inicial de um site de comércio eletrônico permite que os clientes que retornam façam login. Parte do benefício para os clientes que fazem login é que, depois de fazerem-no, o site se personaliza de acordo com suas preferências específicas. Para fazer isso acontecer:

1. O cliente deve fazer login e ser validado com suas credenciais de login.
2. As preferências do cliente são solicitadas a banco de dados de clientes.
3. O banco de dados fornece as preferências do cliente, que são usadas para personalizar o site antes que a página carregue.

Se essas tarefas forem executadas de forma síncrona, cada uma delas deverá terminar antes de a seguinte começar. A página da web não terminaria de carregar até que as preferências do cliente fossem apresentadas pelo banco de dados. No entanto, após a consulta de banco de dados ser enviada ao servidor, o recebimento dos dados do cliente pode ser atrasado ou até mesmo falhar devido a gargalos da rede, tráfego excepcionalmente alto no banco de dados ou conexão ruim nos dispositivos móveis.

Para evitar que o site congele sob essas condições, chame o banco de dados de forma assíncrona. Depois de a chamada do banco de dados ser executada, enviando sua solicitação assíncrona, o código continuará a ser executado conforme o esperado. Se você não gerir adequadamente a resposta de uma chamada assíncrona, o código poderá tentar usar informações que espera de volta do banco de dados quando esses dados ainda não estiverem disponíveis.



Usar uma função de retorno de chamada anônimo

Cada método de objeto de serviço que cria um objeto `AWS.Request` pode aceitar uma função de retorno de chamada anônimo como último parâmetro. A assinatura dessa função de retorno de chamada é:

```
function(error, data) {
    // callback handling code
}
```

Essa função de retorno de chamada é executada ao se retornar uma resposta bem-sucedida ou dados de erro. Se a chamada do método for bem-sucedida, o conteúdo da resposta estará disponível para a função de retorno de chamada no parâmetro `data`. Se a chamada não for bem-sucedida, os detalhes sobre a falha são fornecidos no parâmetro `error`.

Normalmente o código dentro da função de retorno de chamada testa um erro, que ele processa, caso seja retornado um erro. Se o erro não for retornado, o código recuperará os dados na resposta pelo parâmetro `data`. O formato básico da função de retorno de chamada é semelhante a este exemplo.

```
function(error, data) {
    if (error) {
        // error handling code
        console.log(error);
    } else {
        // data handling code
        console.log(data);
    }
}
```

No exemplo anterior, os detalhes do erro ou dos dados retornados são registrados no console. Veja a seguir um exemplo que mostra uma função de retorno de chamada passada como parte da chamada de um método em um objeto de serviço.

```
new AWS.EC2({apiVersion: '2014-10-01'}).describeInstances(function(error, data) {
  if (error) {
    console.log(error); // an error occurred
  } else {
    console.log(data); // request succeeded
  }
```

```
});
```

Acessar os objetos de solicitação e resposta

Dentro da função de retorno de chamada, a palavra-chave `this` do JavaScript refere-se ao objeto `AWS.Response` subjacente à maioria dos serviços. No exemplo a seguir, a propriedade `httpResponse` de um objeto `AWS.Response` é usada dentro de uma função de retorno de chamada para registrar os dados de resposta brutos e os cabeçalhos para ajudar com a depuração.

```
new AWS.EC2({apiVersion: '2014-10-01'}).describeInstances(function(error, data) {
  if (error) {
    console.log(error); // an error occurred
    // Using this keyword to access AWS.Response object and properties
    console.log("Response data and headers: " + JSON.stringify(this.httpResponse));
  } else {
    console.log(data); // request succeeded
  }
});
```

Além disso, como o objeto `AWS.Response` tem uma propriedade `Request` que contém o `AWS.Request` enviado pela chamada do método original, você também pode acessar os detalhes da solicitação que foi feita.

Usar um listener de evento do objeto de solicitação

Se você não criar e passar uma função de retorno de chamada anônimo como parâmetro quando chamar o método de objeto de um serviço, a chamada do método gera um objeto `AWS.Request`, que deve ser enviado manualmente usando o método `send`.

Para processar a resposta, você deve criar um listener de evento para o objeto `AWS.Request` para registrar uma função de retorno de chamada para a chamada do método. O exemplo a seguir mostra como criar o objeto `AWS.Request` para chamar um método de objeto de serviço e o listener do evento para uma devolução bem-sucedida.

```
// create the AWS.Request object
var request = new AWS.EC2({apiVersion: '2014-10-01'}).describeInstances();

// register a callback event handler
request.on('success', function(response) {
```

```
  // log the successful data response
  console.log(response.data);
});

// send the request
request.send();
```

Depois de chamado o método `send` no objeto `AWS.Request`, o manipulador de eventos é executado quando o objeto de serviço recebe um objeto `AWS.Response`.

Para obter mais informações sobre o objeto `AWS.Request`, consulte `Class: AWS.Request` na Referência de API. Para obter mais informações sobre o objeto `AWS.Response`, consulte Usar o objeto de resposta ou `Class: AWS.Response` na Referência de API.

Encadear vários retornos de chamada

Você pode registrar vários retornos de chamada em qualquer objeto de solicitação. É possível registrar vários retornos de chamada para diferentes eventos ou para o mesmo evento. Além disso, você pode encadear retornos de chamada conforme exibido no exemplo a seguir.

```
request.
  on('success', function(response) {
    console.log("Success!");
  }).
  on('error', function(response) {
    console.log("Error!");
  }).
  on('complete', function() {
    console.log("Always!");
  }).
  send();
```

Eventos de conclusão do objeto de solicitação

O objeto `AWS.Request` gera esses eventos de conclusão com base na resposta de cada método de operação de serviço:

- `success`
- `error`
- `complete`

Você pode registrar uma função de retorno de chamada em resposta a qualquer um desses eventos. Para obter uma lista completa de todos os eventos de objeto da solicitação, consulte `Class: AWS.Request` na Referência da API.

O evento "success"

O evento `success` é gerado após uma resposta bem-sucedida recebida do objeto de serviço. Veja aqui como registrar uma função de retorno de chamada para esse evento.

```
request.on('success', function(response) {
  // event handler code
});
```

A resposta fornece uma propriedade `data`, que contém os dados de resposta serializada do serviço. Por exemplo, a chamada a seguir para o método `listBuckets` do objeto de serviço do Amazon S3.

```
s3.listBuckets.on('success', function(response) {
  console.log(response.data);
}).send();
```

retorna a resposta e imprime os conteúdos da propriedade `data` a seguir no console.

```
{ Owner: { ID: '...', DisplayName: '...' },
  Buckets:
   [ { Name: 'someBucketName', CreationDate: someCreationDate },
     { Name: 'otherBucketName', CreationDate: otherCreationDate } ],
  RequestId: '...' }
```

O evento "error"

O evento `error` é gerado após uma resposta de erro recebida do objeto de serviço. Veja aqui como registrar uma função de retorno de chamada para esse evento.

```
request.on('error', function(error, response) {
  // event handling code
});
```

Quando o evento `error` é gerado, o valor da propriedade `data` da resposta é `null` e a propriedade `error` contém os dados do erro. O objeto `error` associado é passado como o primeiro parâmetro para a função de retorno de chamada registrada. Por exemplo, o seguinte código:

```
s3.config.credentials.accessKeyId = 'invalid';
s3.listBuckets().on('error', function(error, response) {
  console.log(error);
}).send();
```

retorna o erro e imprime os seguintes dados de erro no console.

```
{ code: 'Forbidden', message: null }
```

O evento "complete"

O evento `complete` é gerado quando uma chamada de objeto de serviço tiver sido concluída, independentemente de ela resultar em sucesso ou erro. Veja aqui como registrar uma função de retorno de chamada para esse evento.

```
request.on('complete', function(response) {
  // event handler code
});
```

Use o retorno de chamada do evento `complete` para lidar com qualquer limpeza de solicitação que precise executar, independentemente de sucesso ou erro. Se você usar dados de resposta dentro de um retorno de chamada para o evento `complete`, primeiro verifique as propriedades `response.data` ou `response.error` para depois tentar acessar qualquer uma delas, conforme mostrado no exemplo a seguir.

```
request.on('complete', function(response) {
  if (response.error) {
    // an error occurred, handle it
  } else {
    // we can use response.data here
  }
}).send();
```

Eventos HTTP do objeto da solicitação

O objeto `AWS.Request` gera esses eventos HTTP com base na resposta de cada método de operação de serviço:

- `httpHeaders`

- `httpData`
- `httpUploadProgress`
- `httpDownloadProgress`
- `httpError`
- `httpDone`

Você pode registrar uma função de retorno de chamada em resposta a qualquer um desses eventos. Para obter uma lista completa de todos os eventos de objeto da solicitação, consulte `Class: AWS.Request` na Referência da API.

O evento "httpHeaders"

O evento `httpHeaders` é gerado quando cabeçalhos são enviados pelo servidor remoto. Veja aqui como registrar uma função de retorno de chamada para esse evento.

```
request.on('httpHeaders', function(statusCode, headers, response) {
  // event handling code
});
```

O parâmetro `statusCode` para a função de retorno de chamada é o código de status HTTP. O parâmetro `headers` contém os cabeçalhos de resposta.

O evento "httpData"

O evento `httpData` é gerado para transmitir pacotes de dados de resposta do serviço. Veja aqui como registrar uma função de retorno de chamada para esse evento.

```
request.on('httpData', function(chunk, response) {
  // event handling code
});
```

Esse evento é normalmente usado para receber grandes respostas em blocos quando não for prático carregar toda a resposta na memória. Esse evento tem um parâmetro `chunk` adicional que contém uma parte dos dados reais do servidor.

Se você registrar um retorno de chamada para o evento `httpData`, a propriedade `data` da resposta contém toda a saída serializada para a solicitação. Você deve remover o listener `httpData` padrão se você não tiver custos indiretos de memória e análise sintática adicional para os handlers incorporados.

Os eventos "httpUploadProgress" e "httpDownloadProgress"

O evento `httpUploadProgress` é gerado quando a solicitação HTTP tiver carregado mais dados. Da mesma forma, o evento `httpDownloadProgress` é gerado quando a solicitação HTTP tiver baixado mais dados. Veja aqui como registrar uma função de retorno de chamada para esses eventos.

```
request.on('httpUploadProgress', function(progress, response) {
  // event handling code
})
.on('httpDownloadProgress', function(progress, response) {
  // event handling code
});
```

O parâmetro `progress` para a função de retorno de chamada contém um objeto com os bytes carregados e totais da solicitação.

O evento "httpError"

O evento `httpError` é gerado quando a solicitação HTTP falhar. Veja aqui como registrar uma função de retorno de chamada para esse evento.

```
request.on('httpError', function(error, response) {
  // event handling code
});
```

O parâmetro `error` para a função de retorno de chamada contém o erro que foi lançado.

O evento "httpDone"

O evento `httpDone` é gerado quando o servidor terminar de enviar dados. Veja aqui como registrar uma função de retorno de chamada para esse evento.

```
request.on('httpDone', function(response) {
  // event handling code
});
```

Usar async/await

Você pode usar o `async/await` padrão em suas chamadas para o AWS SDK for JavaScript. A maioria das funções que recebem um retorno de chamada não retorna uma promessa. Como você

usa apenas funções do `await` que retornam uma promessa, para usar o padrão de `async/await`, você precisa encadear o método de `.promise()` até o final da chamada e remover o retorno de chamada.

O exemplo a seguir usa async/await para listar todas as tabelas do Amazon DynamoDB em `us-west-2`

```
var AWS = require("aws-sdk");
//Create an Amazon DynamoDB client service object.
dbClient = new AWS.DynamoDB({ region: "us-west-2" });
// Call DynamoDB to list existing tables
const run = async () => {
  try {
    const results = await dbClient.listTables({}).promise();
    console.log(results.TableNames.join("\n"));
  } catch (err) {
    console.error(err);
  }
};
run();
```

> **Note**
>
> Nem todos os navegadores oferecem suporte para async/await. Consulte Funções assíncronas para obter uma lista de navegadores com suporte para async/await.

Usar as promessas do JavaScript

O método `AWS.Request.promise` fornece uma maneira de chamar uma operação de serviço assíncrona e gerenciar o fluxo em vez de usar retornos de chamada. No Node.js e nos scripts do navegador, um objeto `AWS.Request` é retornado quando a operação de serviço é chamada sem uma função de retorno de chamada. Você pode chamar o método `send` da solicitação para fazer a chamada de serviço.

No entanto, o `AWS.Request.promise` imediatamente começa a chamada de serviço e retorna uma promessa que é cumprida com a propriedade `data` da resposta ou rejeitada com a propriedade `error` da resposta.

```
var request = new AWS.EC2({apiVersion: '2014-10-01'}).describeInstances();
```

```
// create the promise object
var promise = request.promise();

// handle promise's fulfilled/rejected states
promise.then(
  function(data) {
    /* process the data */
  },
  function(error) {
    /* handle the error */
  }
);
```

O exemplo a seguir retorna uma promessa que é atendida com um objeto `data` ou rejeitada com um objeto `error`. Usando promessas, um único retorno de chamada não é responsável por detectar erros. Em vez disso, o retorno de chamada correto é chamado com base no sucesso ou na falha de uma solicitação.

```
var s3 = new AWS.S3({apiVersion: '2006-03-01', region: 'us-west-2'});
var params = {
  Bucket: 'bucket',
  Key: 'example2.txt',
  Body: 'Uploaded text using the promise-based method!'
};
var putObjectPromise = s3.putObject(params).promise();
putObjectPromise.then(function(data) {
  console.log('Success');
}).catch(function(err) {
  console.log(err);
});
```

Coordenar várias promessas

Em algumas situações, seu código deve fazer várias chamadas assíncronas que exigem ação somente quando todos tiverem sido retornados com êxito. Se você gerenciar as chamadas individuais do método assíncrono com promessas, pode criar uma promessa adicional que usa o método `all`. Esse método cumpre essa promessa generalista se, e quando, a série de promessas que você passar para o método forem cumpridas. A função de retorno de chamada é transmitida a um array dos valores das promessas passadas para o método `all`.

No exemplo a seguir, uma função do AWS Lambda deve fazer três chamadas assíncronas para Amazon DynamoDB, mas só pode ser concluída após as promessas para cada chamada serem cumpridas.

```
Promise.all([firstPromise, secondPromise, thirdPromise]).then(function(values) {

  console.log("Value 0 is " + values[0].toString);
  console.log("Value 1 is " + values[1].toString);
  console.log("Value 2 is " + values[2].toString);

  // return the result to the caller of the Lambda function
  callback(null, values);
});
```

Suporte de navegador e Node.js para promessas

O suporte para promessas de JavaScript nativo (ECMAScript 2015) depende do mecanismo JavaScript e da versão em que seu código é executado. Para ajudar a determinar o suporte para promessas de JavaScript em cada ambiente onde seu código precisa rodar, consulte a Tabela Compatibilidade de ECMAScript no GitHub.

Usando outras implementações de promessa

Além da implementação de promessa nativa no ECMAScript 2015, você também pode usar bibliotecas de promessa de terceiros, incluindo:

- bluebird
- RSVP
- Q

Essas bibliotecas de promessa opcionais podem ser úteis se você precisar que seu código rode em ambientes que não são compatíveis com a implementação de promessa nativa no ECMAScript 5 e no ECMAScript 2015.

Para usar uma biblioteca de promessa de terceiros, defina uma dependência de promessas no SDK chamando o método `setPromisesDependency` do objeto de configuração global. No navegador de scripts, carregue a biblioteca de promessas de terceiros antes de carregar o SDK. No exemplo a seguir, o SDK é configurado para usar a implementação na biblioteca de promessas do bluebird.

```
AWS.config.setPromisesDependency(require('bluebird'));
```

Para voltar a usar a implementação de promessa nativa do mecanismo JavaScript, chame `setPromisesDependency` novamente, passando um `null` em vez do nome de uma biblioteca.

Usar o objeto de resposta

Depois de um método de objeto de serviço ser chamado, ele retornará um objeto `AWS.Response` passando para sua função de retorno de chamada. Você pode acessar o conteúdo da resposta usando as propriedades do objeto `AWS.Response`. Existem duas propriedades do objeto `AWS.Response` que você usa para acessar o conteúdo da resposta:

- a propriedade `data`
- a propriedade `error`

Ao usar o mecanismo de retorno de chamada padrão, essas duas propriedades serão fornecidas como parâmetros na função de retorno de chamada anônimo, conforme mostrado no exemplo a seguir.

```
function(error, data) {
    if (error) {
        // error handling code
        console.log(error);
    } else {
        // data handling code
        console.log(data);
    }
}
```

Acessar dados retornados no objeto de resposta

A propriedade `data` do objeto `AWS.Response` contém os dados serializados retornados pela solicitação de serviço. Quando a solicitação for bem-sucedida, a propriedade `data` conterá um objeto com um mapa para os dados retornados. A propriedade `data` pode ser nula, caso ocorra um erro.

Aqui está um exemplo de como chamar o método `getItem` de uma tabela do DynamoDB para recuperar o nome do arquivo de um arquivo de imagem para usar como parte de um jogo.

```
// Initialize parameters needed to call DynamoDB
var slotParams = {
    Key : {'slotPosition' : {N: '0'}},
    TableName : 'slotWheels',
    ProjectionExpression: 'imageFile'
};

// prepare request object for call to DynamoDB
var request = new AWS.DynamoDB({region: 'us-west-2', apiVersion:
 '2012-08-10'}).getItem(slotParams);
// log the name of the image file to load in the slot machine
request.on('success', function(response) {
    // logs a value like "cherries.jpg" returned from DynamoDB
    console.log(response.data.Item.imageFile.S);
});
// submit DynamoDB request
request.send();
```

Para este exemplo, a tabela do DynamoDB é uma pesquisa de imagens que mostram os resultados de uma máquina caça-níqueis conforme especificado pelos parâmetros em `slotParams`.

Após uma chamada bem-sucedida do método `getItem`, a propriedade `data` do objeto `AWS.Response` conterá um objeto `Item` retornado por DynamoDB. Os dados retornados são acessados de acordo com o parâmetro `ProjectionExpression` da solicitação, que neste caso significa o membro `imageFile` do objeto `Item`. Como o membro `imageFile` contém um valor de string, você acessa o nome do arquivo da imagem em si por meio do valor do membro-filho de `S` de `imageFile`.

Paginar pelos dados retornados

Às vezes, o conteúdo da propriedade `data` retornada por uma solicitação de serviço abrange várias páginas. Você pode acessar a próxima página de dados chamando o método `response.nextPage`. Esse método envia uma nova solicitação. A resposta da solicitação pode ser capturada com um retorno de chamada ou com listeners de sucesso e erro.

Você pode verificar se os dados retornados por uma solicitação de serviço têm páginas adicionais de dados chamando o método `response.hasNextPage`. Esse método retorna um valor booleano para indicar se a chamada `response.nextPage` retorna dados adicionais.

```
s3.listObjects({Bucket: 'bucket'}).on('success', function handlePage(response) {
```

```
    // do something with response.data
    if (response.hasNextPage()) {
        response.nextPage().on('success', handlePage).send();
    }
}).send();
```

Acessar informações de erro de um objeto de resposta

A propriedade `error` do objeto `AWS.Response` contém os dados de erro disponíveis no caso de um erro de serviço ou de transferência. O erro retornado assume a forma a seguir.

```
{ code: 'SHORT_UNIQUE_ERROR_CODE', message: 'a descriptive error message' }
```

No caso de um erro, o valor da propriedade `data` é `null`. Se você lidar com eventos que estejam em estado de falha, verifique sempre se a propriedade `error` foi definida antes de tentar acessar o valor da propriedade `data`.

Acessar o objeto de solicitação de origem

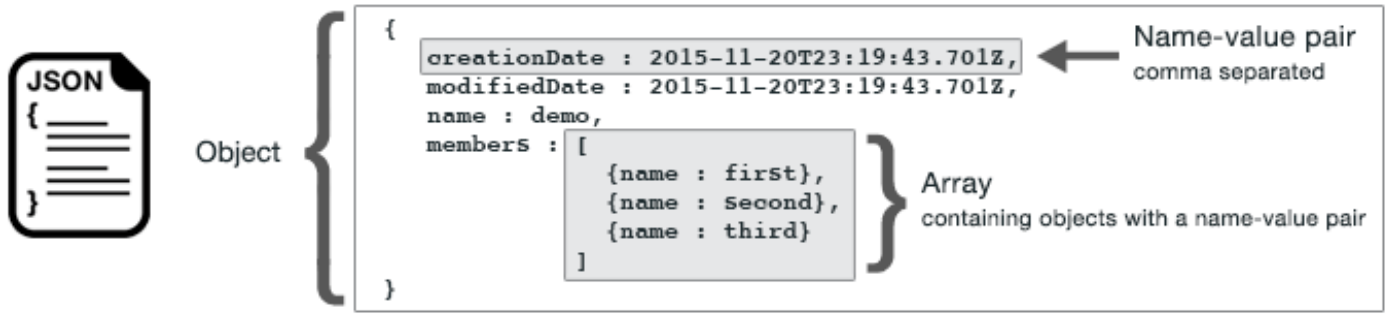
A propriedade `request` fornece acesso ao objeto `AWS.Request` de origem. Ela pode ser útil para fazer referência ao objeto `AWS.Request` original para acessar os parâmetros originais que enviou. No exemplo a seguir, a propriedade `request` é usada para acessar o parâmetro `Key` da solicitação de serviço original.

```
s3.getObject({Bucket: 'bucket', Key: 'key'}).on('success', function(response) {
   console.log("Key was", response.request.params.Key);
}).send();
```

Trabalhar com o JSON

JSON é um formato de intercâmbio de dados capaz de ser lido por humanos e por máquina. Embora o nome JSON seja acrônimo de JavaScript Object Notation, o formato do JSON independe de qualquer linguagem de programação.

O SDK usa JSON para enviar dados para objetos de serviço ao fazer solicitações e recebe dados de objetos de serviço como JSON. Para mais informações sobre JSON, consulte json.org.



JSON representa dados de duas formas:

- Um objeto, que é uma coleção não ordenada de pares de nome/valor. Um objeto é definido dentro das chaves esquerda ({) e direita (}). Cada par de nome e valor começa com o nome seguido por uma vírgula seguido pelo valor. O pares de nome/valor são separados por vírgulas.
- Uma matriz, que é uma coleção ordenada de valores. Um array é definido dentro dos colchetes esquerdo ([) e direito (]). Os itens no array são separados por vírgulas.

Aqui está um exemplo de um objeto JSON que contém um array de objetos em que os objetos representam as cartas de um baralho. Cada carta é definida por dois pares de nome/valor: um que especifica um valor exclusivo para identificar que essa carta e outro que especifica um URL que aponta para a imagem da carta correspondente.

```
var cards = [{"CardID":"defaultname", "Image":"defaulturl"},
  {"CardID":"defaultname", "Image":"defaulturl"},
  {"CardID":"defaultname", "Image":"defaulturl"},
  {"CardID":"defaultname", "Image":"defaulturl"},
  {"CardID":"defaultname", "Image":"defaulturl"}];
```

JSON como parâmetros do objeto de serviço

Veja a seguir um exemplo de JSON simples usado para definir os parâmetros de uma chamada para um objeto de serviço do Lambda.

```
var pullParams = {
   FunctionName : 'slotPull',
   InvocationType : 'RequestResponse',
   LogType : 'None'
};
```

O objeto `pullParams` é definido por três pares de nome/valor, separados por vírgulas, dentro das chaves esquerda e direita. Ao fornecer parâmetros para uma chamada do método do objeto de serviço, os nomes são determinados pelos nomes de parâmetro do método do objeto de serviço que você pretende chamar. Ao invocar uma função do Lambda, `FunctionName`, `InvocationType` e `LogType` são os parâmetros usados para chamar o método `invoke` em um objeto de serviço do Lambda.

Ao passar os parâmetros para uma chamada do método do objeto de serviço, forneça o objeto JSON para a chamada de método, conforme mostrado no exemplo a seguir de como invocar uma função do Lambda.

```
lambda = new AWS.Lambda({region: 'us-west-2', apiVersion: '2015-03-31'});
// create JSON object for service call parameters
var pullParams = {
   FunctionName : 'slotPull',
   InvocationType : 'RequestResponse',
   LogType : 'None'
};
// invoke Lambda function, passing JSON object
lambda.invoke(pullParams, function(err, data) {
   if (err) {
      console.log(err);
   } else {
      console.log(data);
   }
});
```

Retornar dados como JSON

O JSON fornece uma maneira padrão para passar dados entre partes de um aplicativo que precisa enviar vários valores ao mesmo tempo. Os métodos das classes de clientes na API comumente retornam JSON no parâmetro `data` passado para suas funções de retorno de chamada. Por exemplo, aqui está uma chamada para o método `getBucketCors` da classe de cliente Amazon S3.

```
// call S3 to retrieve CORS configuration for selected bucket
s3.getBucketCors(bucketParams, function(err, data) {
  if (err) {
    console.log(err);
  } else if (data) {
    console.log(JSON.stringify(data));
  }
```

```
});
```

O valor de `data` é um objeto JSON. Neste exemplo, JSON que descreve a configuração atual de CORS para um bucket especificado do Amazon S3.

```
{
   "CORSRules": [
      {
          "AllowedHeaders":["*"],
          "AllowedMethods":["POST","GET","PUT","DELETE","HEAD"],
          "AllowedOrigins":["*"],
          "ExposeHeaders":[],
          "MaxAgeSeconds":3000
      }
   ]
}
```

Exemplos de código do SDK para JavaScript

Os tópicos desta seção contêm exemplos de como usar o AWS SDK for JavaScript com as APIs de vários serviços para executar tarefas comuns.

Encontre o código-fonte para esses exemplos e outros no repositório de exemplos de código no GitHub da documentação da AWS. Para sugerir um novo exemplo de código para a equipe de documentação da AWS considerar a produção, crie uma solicitação. A equipe está buscando produzir exemplos de código que abrangem cenários e casos de uso mais amplos, em vez de trechos de código simples que abrangem apenas chamadas de API individuais. Para obter instruções, consulte a seção Código de autoria nas diretrizes de contribuição.

Tópicos

- Exemplos do Amazon CloudWatch
- Exemplos do Amazon DynamoDB
- Exemplos do Amazon EC2
- Exemplos do AWS Elemental MediaConvert
- Exemplos de Amazon S3 Glacier
- Exemplos do IAM AWS
- Exemplos do Amazon Kinesis
- Exemplos do Amazon S3
- Exemplos do Amazon Simple Email Service
- Exemplos do Amazon Simple Notification Service
- Exemplos do Amazon SQS

Exemplos do Amazon CloudWatch

O Amazon CloudWatch (CloudWatch) é um serviço web que monitora os recursos e os aplicativos da Amazon Web Services executados na AWS em tempo real. Você pode usar o CloudWatch para coletar e monitorar métricas, que são as variáveis mensuráveis que ajudam você a avaliar seus recursos e aplicativos. Os alarmes do CloudWatch enviam notificações ou fazem alterações automaticamente nos recursos que você está monitorando com base nas regras definidas.

A API JavaScript do CloudWatch é exposta por meio das classes cliente `AWS.CloudWatch`, `AWS.CloudWatchEvents` e `AWS.CloudWatchLogs`. Para obter mais informações sobre como usar as classes de cliente do CloudWatch, consulte `Class: AWS.CloudWatch`, `Class: AWS.CloudWatchEvents` e `Class: AWS.CloudWatchLogs` na referência da API.

Tópicos

- Criação de alarmes no Amazon CloudWatch
- Usar ações de alarmes no Amazon CloudWatch
- Obter métricas do Amazon CloudWatch
- Enviar eventos para o Amazon CloudWatch Events
- Uso de filtros de assinatura do Amazon CloudWatch Logs

Criação de alarmes no Amazon CloudWatch

Este exemplo de código Node.js mostra:

- Como recuperar informações básicas sobre os alarmes do CloudWatch.
- Como criar e excluir um alarme do CloudWatch.

O cenário

Um alarme observa uma única métrica ao longo de um período especificado por você e realiza uma ou mais ações com base no valor da métrica relativo a um determinado limite ao longo de vários períodos.

Neste exemplo, uma série de módulos Node.js é usada para criar alarmes no CloudWatch. Os módulos Node.js usam o SDK para JavaScript para criar alarmes usando esses métodos da classe de cliente do `AWS.CloudWatch`:

- `describeAlarms`
- `putMetricAlarm`
- `deleteAlarms`

Para obter mais informações sobre os alarmes do CloudWatch e as alterações de estado, consulte Criar alarmes do Amazon CloudWatch no Manual do usuário do Amazon CloudWatch.

Tarefas de pré-requisito

Para configurar e executar este exemplo, você deve primeiro concluir estas tarefas:

- Instale o Node.js. Para obter mais informações sobre como instalar Node.js, consulte o website de Node.js.
- Crie um arquivo de configurações compartilhado com as credenciais de usuário. Para obter mais informações sobre como fornecer um arquivo de credenciais compartilhadas, consulte Carregar credenciais em Node.js do arquivo de credenciais compartilhado.

Descrever alarmes

Crie um módulo do Node.js com o nome de arquivo `cw_describealarms.js`. Não se esqueça de configurar o SDK conforme mostrado anteriormente. Para acessar o CloudWatch, crie um objeto de serviço do `AWS.CloudWatch`. Crie um objeto JSON para armazenar os parâmetros para recuperar as descrições de alarme, limitando os alarmes retornados àqueles com um estado de `INSUFFICIENT_DATA`. Depois, chame o método `describeAlarms` do objeto de serviço do `AWS.CloudWatch`.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
```

```
// Set the region
AWS.config.update({ region: "REGION" });

// Create CloudWatch service object
var cw = new AWS.CloudWatch({ apiVersion: "2010-08-01" });

cw.describeAlarms({ StateValue: "INSUFFICIENT_DATA" }, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    // List the names of all current alarms in the console
    data.MetricAlarms.forEach(function (item, index, array) {
      console.log(item.AlarmName);
    });
  }
});
```

Para executar o exemplo, digite o seguinte na linha de comando.

```
node cw_describealarms.js
```

Este código de exemplo pode ser encontrado aqui no GitHub.

Criando um alarme para o CloudWatch Metric

Crie um módulo do Node.js com o nome de arquivo `cw_putmetricalarm.js`. Não se esqueça de configurar o SDK conforme mostrado anteriormente. Para acessar o CloudWatch, crie um objeto de serviço do `AWS.CloudWatch`. Crie um objeto JSON para os parâmetros necessários a fim de criar um alarme baseado em uma métrica. Neste caso, a utilização de CPU de uma instância do Amazon EC2. Os demais parâmetros são definidos, de maneira que o alarme seja acionado quando a métrica exceder o limite de 70%. Depois, chame o método `describeAlarms` do objeto de serviço do `AWS.CloudWatch`.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create CloudWatch service object
var cw = new AWS.CloudWatch({ apiVersion: "2010-08-01" });
```

```
var params = {
  AlarmName: "Web_Server_CPU_Utilization",
  ComparisonOperator: "GreaterThanThreshold",
  EvaluationPeriods: 1,
  MetricName: "CPUUtilization",
  Namespace: "AWS/EC2",
  Period: 60,
  Statistic: "Average",
  Threshold: 70.0,
  ActionsEnabled: false,
  AlarmDescription: "Alarm when server CPU exceeds 70%",
  Dimensions: [
    {
      Name: "InstanceId",
      Value: "INSTANCE_ID",
    },
  ],
  Unit: "Percent",
};

cw.putMetricAlarm(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

Para executar o exemplo, digite o seguinte na linha de comando.

```
node cw_putmetricalarm.js
```

Este código de exemplo pode ser encontrado aqui no GitHub.

Exclusão de um alarme

Crie um módulo do Node.js com o nome de arquivo `cw_deletealarms.js`. Não se esqueça de configurar o SDK conforme mostrado anteriormente. Para acessar o CloudWatch, crie um objeto de serviço do `AWS.CloudWatch`. Crie um objeto JSON para manter os nomes dos alarmes que você deseja excluir. Depois, chame o método `deleteAlarms` do objeto de serviço do `AWS.CloudWatch`.

```
// Load the AWS SDK for Node.js
```

```
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create CloudWatch service object
var cw = new AWS.CloudWatch({ apiVersion: "2010-08-01" });

var params = {
  AlarmNames: ["Web_Server_CPU_Utilization"],
};

cw.deleteAlarms(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

Para executar o exemplo, digite o seguinte na linha de comando.

```
node cw_deletealarms.js
```

Este código de exemplo pode ser encontrado aqui no GitHub.

Usar ações de alarmes no Amazon CloudWatch

Este exemplo de código Node.js mostra:

- Como alterar automaticamente o estado das instâncias do Amazon EC2 com base em um alarme do CloudWatch.

O cenário

Ao usar ações de alarme do , é possível criar alarmes que param, encerram, reiniciam ou recuperam as instâncias do Amazon EC2 automaticamente. Use as ações parar ou encerrar quando não for

mais necessário que uma instância seja executada. Use as ações reiniciar e recuperar para reiniciar essas instâncias automaticamente.

Neste exemplo, uma série de módulos de Node.js é usada para definir uma ação de alarme no CloudWatch que acionará a reinicialização de uma instância do Amazon EC2. Os módulos do Node.js usam o SDK para JavaScript para gerenciar instâncias do Amazon EC2 usando os métodos da classe de cliente do `CloudWatch`:

- `enableAlarmActions`
- `disableAlarmActions`

Para obter mais informações sobre as ações de alarme do CloudWatch, consulte Criar alarmes que param, encerram, reinicializam ou recuperam uma instância, no Guia do usuário do Amazon CloudWatch.

Tarefas de pré-requisito

Para configurar e executar este exemplo, você deve primeiro concluir estas tarefas:

- Instale o Node.js. Para obter mais informações sobre como instalar Node.js, consulte o website de Node.js.
- Crie um arquivo de configurações compartilhado com as credenciais de usuário. Para obter mais informações sobre como fornecer um arquivo de credenciais compartilhadas, consulte Carregar credenciais em Node.js do arquivo de credenciais compartilhado.
- Crie uma função do IAM cuja política conceda permissão para descrever, reinicializar, interromper ou encerrar uma instância do Amazon EC2. Para obter mais informações sobre como criar um perfil do IAM, consulte Criação de uma função para delegar permissões a um serviço da AWS no Guia do usuário do IAM.

Use a política de função a seguir ao criar a função do IAM.

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Effect": "Allow",
         "Action": [
            "cloudwatch:Describe*",
            "ec2:Describe*",
```

```
            "ec2:RebootInstances",
            "ec2:StopInstances*",
            "ec2:TerminateInstances"
         ],
         "Resource": [
            "*"
         ]
      }
   ]
}
```

Configure o SDK para JavaScript criando um objeto de configuração global e definindo a região do código. Neste exemplo, a região é definida como `us-west-2`.

```
// Load the SDK for JavaScript
var AWS = require('aws-sdk');
// Set the Region
AWS.config.update({region: 'us-west-2'});
```

Criar e ativar ações em um alarme

Crie um módulo do Node.js com o nome de arquivo `cw_enablealarmactions.js`. Não se esqueça de configurar o SDK conforme mostrado anteriormente. Para acessar o CloudWatch, crie um objeto de serviço do `AWS.CloudWatch`.

Crie um objeto JSON para manter os parâmetros da criação de um alarme especificando `ActionsEnabled` como `true` e uma matriz de ARNs para as ações que o alarme acionará. Chame o método `putMetricAlarm` do objeto de serviço `AWS.CloudWatch`. Isso criará o alarme se ele não existir ou o atualizará se ele já existir.

Na função de retorno de chamada do `putMetricAlarm`, mediante a conclusão bem-sucedida, crie um objeto JSON que contém o nome do alarme do CloudWatch. Chame o método `enableAlarmActions` para ativar a ação do alarme.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create CloudWatch service object
var cw = new AWS.CloudWatch({ apiVersion: "2010-08-01" });
```

```
var params = {
  AlarmName: "Web_Server_CPU_Utilization",
  ComparisonOperator: "GreaterThanThreshold",
  EvaluationPeriods: 1,
  MetricName: "CPUUtilization",
  Namespace: "AWS/EC2",
  Period: 60,
  Statistic: "Average",
  Threshold: 70.0,
  ActionsEnabled: true,
  AlarmActions: ["ACTION_ARN"],
  AlarmDescription: "Alarm when server CPU exceeds 70%",
  Dimensions: [
    {
      Name: "InstanceId",
      Value: "INSTANCE_ID",
    },
  ],
  Unit: "Percent",
};

cw.putMetricAlarm(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Alarm action added", data);
    var paramsEnableAlarmAction = {
      AlarmNames: [params.AlarmName],
    };
    cw.enableAlarmActions(paramsEnableAlarmAction, function (err, data) {
      if (err) {
        console.log("Error", err);
      } else {
        console.log("Alarm action enabled", data);
      }
    });
  }
});
```

Para executar o exemplo, digite o seguinte na linha de comando.

```
node cw_enablealarmactions.js
```

Este código de exemplo pode ser encontrado aqui no GitHub.

Desabilitar ações em um alarme

Crie um módulo do Node.js com o nome de arquivo `cw_disablealarmactions.js`. Não se esqueça de configurar o SDK conforme mostrado anteriormente. Para acessar o CloudWatch, crie um objeto de serviço do `AWS.CloudWatch`. Crie um objeto JSON que contenha o nome do alarme do CloudWatch. Chame o método `disableAlarmActions` para desativar as ações desse alarme.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create CloudWatch service object
var cw = new AWS.CloudWatch({ apiVersion: "2010-08-01" });

cw.disableAlarmActions(
  { AlarmNames: ["Web_Server_CPU_Utilization"] },
  function (err, data) {
    if (err) {
      console.log("Error", err);
    } else {
      console.log("Success", data);
    }
  }
);
```

Para executar o exemplo, digite o seguinte na linha de comando.

```
node cw_disablealarmactions.js
```

Este código de exemplo pode ser encontrado aqui no GitHub.

Obter métricas do Amazon CloudWatch

Este exemplo de código Node.js mostra:

- Como recuperar uma lista de métricas do do CloudWatch publicadas.
- Como publicar pontos de dados em métricas do CloudWatch.

O cenário

Métricas são dados sobre a performance de seus sistemas. É possível habilitar o monitoramento detalhado de alguns recursos, como instâncias do Amazon EC2 ou as próprias métricas do aplicativo.

Neste exemplo, uma série de módulos Node.js é usada para obter métricas do CloudWatch e para enviar eventos ao Amazon CloudWatch Events. Os módulos Node.js usam o SDK para JavaScript para obter métricas do CloudWatch usando esses métodos da classe de cliente `CloudWatch`:

- `listMetrics`
- `putMetricData`

Para obter mais informações sobre métricas do CloudWatch, consulte Como usar métricas do Amazon CloudWatch no Manual do usuário do Amazon CloudWatch.

Tarefas de pré-requisito

Para configurar e executar este exemplo, você deve primeiro concluir estas tarefas:

- Instale o Node.js. Para obter mais informações sobre como instalar Node.js, consulte o website de Node.js.
- Crie um arquivo de configurações compartilhado com as credenciais de usuário. Para obter mais informações sobre como fornecer um arquivo de credenciais compartilhadas, consulte Carregar credenciais em Node.js do arquivo de credenciais compartilhado.

Listar métricas

Crie um módulo do Node.js com o nome de arquivo `cw_listmetrics.js`. Não se esqueça de configurar o SDK conforme mostrado anteriormente. Para acessar o CloudWatch, crie um objeto de serviço do `AWS.CloudWatch`. Crie um objeto JSON que contenha os parâmetros necessários para listar métricas dentro do namespace `AWS/Logs`. Chame o método `listMetrics` para listar a métrica `IncomingLogEvents`.

```
// Load the AWS SDK for Node.js
```

```
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create CloudWatch service object
var cw = new AWS.CloudWatch({ apiVersion: "2010-08-01" });

var params = {
  Dimensions: [
    {
      Name: "LogGroupName" /* required */,
    },
  ],
  MetricName: "IncomingLogEvents",
  Namespace: "AWS/Logs",
};

cw.listMetrics(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Metrics", JSON.stringify(data.Metrics));
  }
});
```

Para executar o exemplo, digite o seguinte na linha de comando.

```
node cw_listmetrics.js
```

Este código de exemplo pode ser encontrado aqui no GitHub.

Enviar métricas personalizadas

Crie um módulo do Node.js com o nome de arquivo `cw_putmetricdata.js`. Não se esqueça de configurar o SDK conforme mostrado anteriormente. Para acessar o CloudWatch, crie um objeto de serviço do `AWS.CloudWatch`. Crie um objeto JSON que contenha os parâmetros necessários a fim de enviar um ponto de dados para a métrica personalizada `PAGES_VISITED`. Chame o método `putMetricData`.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
```

```
AWS.config.update({ region: "REGION" });

// Create CloudWatch service object
var cw = new AWS.CloudWatch({ apiVersion: "2010-08-01" });

// Create parameters JSON for putMetricData
var params = {
  MetricData: [
    {
      MetricName: "PAGES_VISITED",
      Dimensions: [
        {
          Name: "UNIQUE_PAGES",
          Value: "URLS",
        },
      ],
      Unit: "None",
      Value: 1.0,
    },
  ],
  Namespace: "SITE/TRAFFIC",
};

cw.putMetricData(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", JSON.stringify(data));
  }
});
```

Para executar o exemplo, digite o seguinte na linha de comando.

```
node cw_putmetricdata.js
```

Este código de exemplo pode ser encontrado aqui no GitHub.

Enviar eventos para o Amazon CloudWatch Events

Este exemplo de código Node.js mostra:

- Como criar e atualizar uma regra usada para acionar um evento.
- Como definir um ou mais destinos para responder a um evento.
- Como enviar eventos correspondentes aos destinos para processamento.

O cenário

O CloudWatch Events oferece um fluxo quase em tempo real de eventos do sistema que descrevem as mudanças nos recursos da Amazon Web Services para vários destinos. Com regras simples, é possível corresponder eventos e roteá-los para um ou mais fluxos ou funções de destino.

Neste exemplo, uma série de módulos Node.js é usada para enviar eventos ao CloudWatch Events. Os módulos Node.js usam o SDK para JavaScript para gerenciar instâncias usando esses métodos da classe de cliente :

- `putRule`
- `putTargets`
- `putEvents`

Para obter mais informações sobre o CloudWatch Events, consulte Adicionar eventos com o PutEvents no Guia do usuário do Amazon CloudWatch Events.

Tarefas de pré-requisito

Para configurar e executar este exemplo, você deve primeiro concluir estas tarefas:

- Instale o Node.js. Para obter mais informações sobre como instalar Node.js, consulte o website de Node.js.
- Crie um arquivo de configurações compartilhado com as credenciais de usuário. Para obter mais informações sobre como fornecer um arquivo de credenciais compartilhadas, consulte Carregar credenciais em Node.js do arquivo de credenciais compartilhado.
- Crie uma função Lambda usando o modelo hello-world para servir como destino para eventos. Para saber como, consulte Etapa 1: Criar uma função do AWS Lambda no Guia do usuário do Amazon CloudWatch Events.
- Crie um perfil do IAM cuja política conceda permissão para o CloudWatch Events e que inclua `events.amazonaws.com` como uma entidade confiável. Para obter mais informações sobre

como criar um perfil do IAM, consulte Criação de uma função para delegar permissões a um serviço da AWS no Guia do usuário do IAM.

Use a política de função a seguir ao criar a função do IAM.

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Sid": "CloudWatchEventsFullAccess",
         "Effect": "Allow",
         "Action": "events:*",
         "Resource": "*"
      },
      {
         "Sid": "IAMPassRoleForCloudWatchEvents",
         "Effect": "Allow",
         "Action": "iam:PassRole",
         "Resource": "arn:aws:iam::*:role/AWS_Events_Invoke_Targets"
      }
   ]
}
```

Use o relacionamento de confiança a seguir ao criar a função do IAM.

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

Criar uma regra programada

Crie um módulo do Node.js com o nome de arquivo `cwe_putrule.js`. Não se esqueça de configurar o SDK conforme mostrado anteriormente. Para acessar o CloudWatch Events, crie um objeto de serviço de `AWS.CloudWatchEvents`. Crie um objeto JSON que contenha os parâmetros necessários para especificar a nova regra programada, que inclua o seguinte:

- Um nome para a regra
- O ARN da função do &IAM; que você criou anteriormente
- Uma expressão para programar o acionamento da regra a cada cinco minutos

Chame o método `putRule` para criar a regra. O retorno de chamada retorna o ARN da regra nova ou atualizada.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create CloudWatchEvents service object
var cwevents = new AWS.CloudWatchEvents({ apiVersion: "2015-10-07" });

var params = {
  Name: "DEMO_EVENT",
  RoleArn: "IAM_ROLE_ARN",
  ScheduleExpression: "rate(5 minutes)",
  State: "ENABLED",
};

cwevents.putRule(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.RuleArn);
  }
});
```

Para executar o exemplo, digite o seguinte na linha de comando.

```
node cwe_putrule.js
```

Este código de exemplo pode ser encontrado aqui no GitHub.

Adicionar o destino da função AWS Lambda

Crie um módulo do Node.js com o nome de arquivo `cwe_puttargets.js`. Não se esqueça de configurar o SDK conforme mostrado anteriormente. Para acessar o CloudWatch Events, crie um objeto de serviço de `AWS.CloudWatchEvents`. Crie um objeto JSON que contém os parâmetros necessários para especificar a regra à qual você deseja anexar o destino, inclusive o ARN da função do Lambda criada. Chame o método `putTargets` do objeto de serviço do `AWS.CloudWatchEvents`.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create CloudWatchEvents service object
var cwevents = new AWS.CloudWatchEvents({ apiVersion: "2015-10-07" });

var params = {
  Rule: "DEMO_EVENT",
  Targets: [
    {
      Arn: "LAMBDA_FUNCTION_ARN",
      Id: "myCloudWatchEventsTarget",
    },
  ],
};

cwevents.putTargets(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

Para executar o exemplo, digite o seguinte na linha de comando.

```
node cwe_puttargets.js
```

Este código de exemplo pode ser encontrado aqui no GitHub.

Enviar eventos

Crie um módulo do Node.js com o nome de arquivo `cwe_putevents.js`. Não se esqueça de configurar o SDK conforme mostrado anteriormente. Para acessar o CloudWatch Events, crie um objeto de serviço de `AWS.CloudWatchEvents`. Crie um objeto JSON que contenha os parâmetros necessários para enviar eventos. Para cada caso, inclua a fonte do evento, os ARNs de todos os recursos afetados pelo evento e os detalhes do evento. Chame o método `putEvents` do objeto de serviço do `AWS.CloudWatchEvents`.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create CloudWatchEvents service object
var cwevents = new AWS.CloudWatchEvents({ apiVersion: "2015-10-07" });

var params = {
  Entries: [
    {
      Detail: '{ "key1": "value1", "key2": "value2" }',
      DetailType: "appRequestSubmitted",
      Resources: ["RESOURCE_ARN"],
      Source: "com.company.app",
    },
  ],
};

cwevents.putEvents(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.Entries);
  }
});
```

Para executar o exemplo, digite o seguinte na linha de comando.

```
node cwe_putevents.js
```

Este código de exemplo pode ser encontrado aqui no GitHub.

Uso de filtros de assinatura do Amazon CloudWatch Logs

Este exemplo de código Node.js mostra:

- Como criar e excluir filtros para eventos de log no CloudWatch Logs.

O cenário

As assinaturas dão acesso a um feed em tempo real de eventos de log do CloudWatch Logs e entrega esse feed a outros serviços, como um stream do Amazon Kinesis ou AWS Lambda, para processamento personalizado, análise ou carregamento em outros sistemas. Um filtro de assinatura define o padrão a ser usado para filtrar quais eventos de log são entregues ao seu recurso da AWS.

Neste exemplo, uma série de módulos Node.js é usada para listar, criar e excluir um filtro de assinatura no CloudWatch Logs. O destino dos eventos de log é uma função Lambda. Os módulos Node.js usam o SDK para JavaScript para gerenciar filtros de assinatura usando esses métodos da classe de clientedo `CloudWatchLogs`:

- `putSubscriptionFilters`
- `describeSubscriptionFilters`
- `deleteSubscriptionFilter`

Para obter mais informações sobre as assinaturas do CloudWatch Logs, consulte Processamento em tempo real dos dados de log com assinaturas no Guia de usuário do Amazon CloudWatch Logs.

Tarefas de pré-requisito

Para configurar e executar este exemplo, você deve primeiro concluir estas tarefas:

- Instale o Node.js. Para obter mais informações sobre como instalar Node.js, consulte o website de Node.js.
- Crie um arquivo de configurações compartilhado com as credenciais de usuário. Para obter mais informações sobre como fornecer um arquivo de credenciais compartilhadas, consulte Carregar credenciais em Node.js do arquivo de credenciais compartilhado.

- Crie uma função Lambda como destino para os eventos de log. Você precisará usar o ARN dessa função. Para obter mais informações sobre a sondagem longa como configurar uma função do Lambda, consulte Filtros de assinatura com o AWS Lambda no Guia de usuário do Amazon CloudWatch Logs.
- Crie uma função do IAM cuja política conceda permissão para invocar a função Lambda que você criou e que conceda acesso total ao CloudWatch Logs ou aplique a política a seguir na função de execução que você cria para a função Lambda. Para obter mais informações sobre como criar um perfil do IAM, consulte Criação de uma função para delegar permissões a um serviço da AWS no Guia do usuário do IAM.

Use a política de função a seguir ao criar a função do IAM.

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Effect": "Allow",
         "Action": [
            "logs:CreateLogGroup",
            "logs:CreateLogStream",
            "logs:PutLogEvents"
         ],
         "Resource": "arn:aws:logs:*:*:*"
      },
      {
         "Effect": "Allow",
         "Action": [
            "lambda:InvokeFunction"
         ],
         "Resource": [
            "*"
         ]
      }
   ]
}
```

Descrever os filtros de assinatura existentes

Crie um módulo do Node.js com o nome de arquivo `cwl_describesubscriptionfilters.js`. Não se esqueça de configurar o SDK conforme mostrado anteriormente. Para acessar os logs

do CloudWatch, crie um objeto de serviço de `AWS.CloudWatchLogs`. Crie um objeto JSON que contenha os parâmetros necessários para descrever os filtros existentes, inclusive o nome do grupo de logs e o número máximo de filtros que você deseja descrever. Chame o método `describeSubscriptionFilters`.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the CloudWatchLogs service object
var cwl = new AWS.CloudWatchLogs({ apiVersion: "2014-03-28" });

var params = {
  logGroupName: "GROUP_NAME",
  limit: 5,
};

cwl.describeSubscriptionFilters(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.subscriptionFilters);
  }
});
```

Para executar o exemplo, digite o seguinte na linha de comando.

```
node cwl_describesubscriptionfilters.js
```

Este código de exemplo pode ser encontrado aqui no GitHub.

Criar um filtro de assinatura

Crie um módulo do Node.js com o nome de arquivo `cwl_putsubscriptionfilter.js`. Não se esqueça de configurar o SDK conforme mostrado anteriormente. Para acessar os logs do CloudWatch, crie um objeto de serviço de `AWS.CloudWatchLogs`. Crie um objeto JSON que contenha os parâmetros necessários para criar um filtro, inclusive o ARN da função de destino do Lambda, o nome do filtro, padrão de string para filtrar e o nome do grupo de logs. Chame o método `putSubscriptionFilters`.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the CloudWatchLogs service object
var cwl = new AWS.CloudWatchLogs({ apiVersion: "2014-03-28" });

var params = {
  destinationArn: "LAMBDA_FUNCTION_ARN",
  filterName: "FILTER_NAME",
  filterPattern: "ERROR",
  logGroupName: "LOG_GROUP",
};

cwl.putSubscriptionFilter(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

Para executar o exemplo, digite o seguinte na linha de comando.

```
node cwl_putsubscriptionfilter.js
```

Este código de exemplo pode ser encontrado aqui no GitHub.

Excluir um filtro de assinatura

Crie um módulo do Node.js com o nome de arquivo `cwl_deletesubscriptionfilters.js`. Não se esqueça de configurar o SDK conforme mostrado anteriormente. Para acessar os logs do CloudWatch, crie um objeto de serviço de `AWS.CloudWatchLogs`. Crie um objeto JSON que contenha os parâmetros necessários para excluir um filtro, inclusive os nomes do filtro e o grupo de logs. Chame o método `deleteSubscriptionFilters`.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });
```

```
// Create the CloudWatchLogs service object
var cwl = new AWS.CloudWatchLogs({ apiVersion: "2014-03-28" });

var params = {
  filterName: "FILTER",
  logGroupName: "LOG_GROUP",
};

cwl.deleteSubscriptionFilter(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

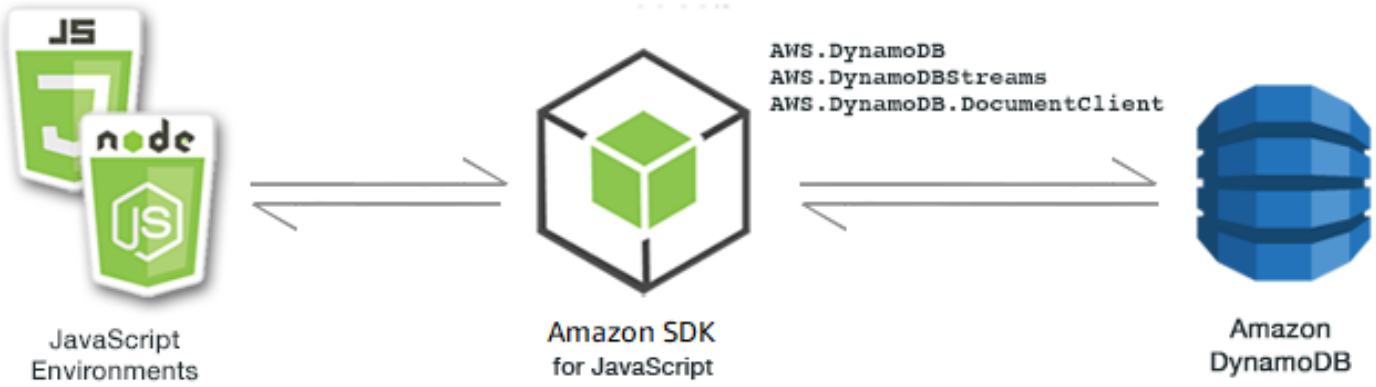
Para executar o exemplo, digite o seguinte na linha de comando.

```
node cwl_deletesubscriptionfilter.js
```

Este código de exemplo pode ser encontrado aqui no GitHub.

Exemplos do Amazon DynamoDB

Amazon DynamoDB é um banco de dados em nuvem NoSQL totalmente gerenciado que oferece suporte a modelos de armazenamento de documentos e chave-valor. Você cria tabelas sem esquema de dados sem a necessidade de provisionar ou manter servidores de banco de dados dedicados.



A API JavaScript do DynamoDB é exposta por meio das classes cliente `AWS.DynamoDB`, `AWS.DynamoDBStreams` e `AWS.DynamoDB.DocumentClient`. Para obter mais informações

sobre como usar as classes de cliente do DynamoDB, consulte `Class: AWS.DynamoDB`, `Class: AWS.DynamoDBStreams` e `Class: AWS.DynamoDB.DocumentClient` na referência da API.

Tópicos

- Criação e uso de tabelas no DynamoDB
- Ler e gravar um único item no DynamoDB
- Ler e gravar itens em lote no DynamoDB
- Consultar e verificar uma tabela do DynamoDB
- Uso do cliente de documentos do DynamoDB

Criação e uso de tabelas no DynamoDB

Este exemplo de código Node.js mostra:

- Como criar e gerenciar tabelas usadas para armazenar e recuperar dados do DynamoDB.

O cenário

De forma semelhante a outros sistemas de banco de dados, o DynamoDB armazena dados em tabelas. Uma tabela do DynamoDB é uma coleção de dados organizada em itens semelhantes às linhas. Para armazenar ou acessar dados no DynamoDB, você cria e trabalha com tabelas.

Neste exemplo, você usa uma série de módulos de Node.js para realizar operações básicas com uma tabela do DynamoDB. O código usa o SDK para JavaScript para criar e trabalhar com tabelas usando esses métodos da classe de cliente `AWS.DynamoDB`:

- createTable
- listTables
- describeTable
- deleteTable

Tarefas de pré-requisito

Para configurar e executar este exemplo, primeiro conclua estas tarefas:

- Instale o Node.js. Para obter mais informações, consulte o website Node.js.
- Crie um arquivo de configurações compartilhado com as credenciais de usuário. Para obter mais informações sobre como fornecer um arquivo de credenciais compartilhadas, consulte Carregar credenciais em Node.js do arquivo de credenciais compartilhado.

Criar uma tabela

Crie um módulo do Node.js com o nome de arquivo `ddb_createtable.js`. Não se esqueça de configurar o SDK conforme mostrado anteriormente. Para acessar o DynamoDB, crie um objeto de serviço do `AWS.DynamoDB`. Crie um objeto JSON que contenha os parâmetros necessários para criar uma tabela, que, neste exemplo, inclui o nome e o tipo de dados de cada atributo, o esquema de chave, o nome da tabela e as unidades de taxa de transferência para provisionar. Chame o método `createTable` do objeto de serviço do DynamoDB.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the DynamoDB service object
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

var params = {
  AttributeDefinitions: [
    {
      AttributeName: "CUSTOMER_ID",
      AttributeType: "N",
    },
    {
      AttributeName: "CUSTOMER_NAME",
      AttributeType: "S",
    },
  ],
  KeySchema: [
    {
      AttributeName: "CUSTOMER_ID",
      KeyType: "HASH",
```

```
    },
    {
      AttributeName: "CUSTOMER_NAME",
      KeyType: "RANGE",
    },
  ],
  ProvisionedThroughput: {
    ReadCapacityUnits: 1,
    WriteCapacityUnits: 1,
  },
  TableName: "CUSTOMER_LIST",
  StreamSpecification: {
    StreamEnabled: false,
  },
};

// Call DynamoDB to create the table
ddb.createTable(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Table Created", data);
  }
});
```

Para executar o exemplo, digite o seguinte na linha de comando.

```
node ddb_createtable.js
```

Este código de exemplo pode ser encontrado aqui no GitHub.

Listar as tabelas

Crie um módulo do Node.js com o nome de arquivo `ddb_listtables.js`. Não se esqueça de configurar o SDK conforme mostrado anteriormente. Para acessar o DynamoDB, crie um objeto de serviço do `AWS.DynamoDB`. Crie um objeto JSON que contém os parâmetros necessários para listar as tabelas, que, neste exemplo, limita o número de tabelas listadas a 10. Chame o método `listTables` do objeto de serviço do DynamoDB.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
```

```
AWS.config.update({ region: "REGION" });

// Create the DynamoDB service object
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

// Call DynamoDB to retrieve the list of tables
ddb.listTables({ Limit: 10 }, function (err, data) {
  if (err) {
    console.log("Error", err.code);
  } else {
    console.log("Table names are ", data.TableNames);
  }
});
```

Para executar o exemplo, digite o seguinte na linha de comando.

```
node ddb_listtables.js
```

Este código de exemplo pode ser encontrado aqui no GitHub.

Descrição de uma tabela

Crie um módulo do Node.js com o nome de arquivo `ddb_describetable.js`. Não se esqueça de configurar o SDK conforme mostrado anteriormente. Para acessar o DynamoDB, crie um objeto de serviço do `AWS.DynamoDB`. Crie um objeto JSON que contém os parâmetros necessários para descrever uma tabela, que, neste exemplo, inclui o nome da tabela fornecido como um parâmetro de linha de comando. Chame o método `describeTable` do objeto de serviço do DynamoDB.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the DynamoDB service object
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

var params = {
  TableName: process.argv[2],
};

// Call DynamoDB to retrieve the selected table descriptions
ddb.describeTable(params, function (err, data) {
```

```
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.Table.KeySchema);
  }
});
```

Para executar o exemplo, digite o seguinte na linha de comando.

```
node ddb_describetable.js TABLE_NAME
```

Este código de exemplo pode ser encontrado aqui no GitHub.

Exclusão de uma tabela

Crie um módulo do Node.js com o nome de arquivo `ddb_deletetable.js`. Não se esqueça de configurar o SDK conforme mostrado anteriormente. Para acessar o DynamoDB, crie um objeto de serviço do `AWS.DynamoDB`. Crie um objeto JSON com os parâmetros necessários para excluir uma tabela, que, neste exemplo, inclui o nome da tabela fornecido como um parâmetro de linha de comando. Chame o método `deleteTable` do objeto de serviço do DynamoDB.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the DynamoDB service object
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

var params = {
  TableName: process.argv[2],
};

// Call DynamoDB to delete the specified table
ddb.deleteTable(params, function (err, data) {
  if (err && err.code === "ResourceNotFoundException") {
    console.log("Error: Table not found");
  } else if (err && err.code === "ResourceInUseException") {
    console.log("Error: Table in use");
  } else {
    console.log("Success", data);
  }
```

```
});
```

Para executar o exemplo, digite o seguinte na linha de comando.

```
node ddb_deletetable.js TABLE_NAME
```

Este código de exemplo pode ser encontrado aqui no GitHub.

Ler e gravar um único item no DynamoDB

Este exemplo de código Node.js mostra:

- Como adicionar um item em uma tabela do DynamoDB.
- Como recuperar um item em uma tabela do DynamoDB.
- Como excluir um item de uma tabela do DynamoDB.

O cenário

Neste exemplo, você usa uma série de módulos de Node.js para ler e gravar um item em uma tabela do DynamoDB usando esses métodos da classe de cliente `AWS.DynamoDB`:

- putItem
- getItem
- deleteItem

Tarefas de pré-requisito

Para configurar e executar este exemplo, primeiro conclua estas tarefas:

- Instale o Node.js. Para obter mais informações, consulte o website Node.js.
- Crie um arquivo de configurações compartilhado com as credenciais de usuário. Para obter mais informações sobre como fornecer um arquivo de credenciais compartilhadas, consulte Carregar credenciais em Node.js do arquivo de credenciais compartilhado.

- Crie uma tabela do DynamoDB cujos itens você possa acessar. Para obter mais informações sobre como criar uma tabela do DynamoDB, consulte Criação e uso de tabelas no DynamoDB.

Gravação de um item

Crie um módulo do Node.js com o nome de arquivo `ddb_putitem.js`. Não se esqueça de configurar o SDK conforme mostrado anteriormente. Para acessar o DynamoDB, crie um objeto de serviço do `AWS.DynamoDB`. Crie um objeto JSON que contém os parâmetros necessários para adicionar um item, que, neste exemplo, inclui o nome da tabela e um mapa que define os atributos a serem configurados e os valores de cada atributo. Chame o método `putItem` do objeto de serviço do DynamoDB.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the DynamoDB service object
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

var params = {
  TableName: "CUSTOMER_LIST",
  Item: {
    CUSTOMER_ID: { N: "001" },
    CUSTOMER_NAME: { S: "Richard Roe" },
  },
};

// Call DynamoDB to add the item to the table
ddb.putItem(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

Para executar o exemplo, digite o seguinte na linha de comando.

```
node ddb_putitem.js
```

Este código de exemplo pode ser encontrado aqui no GitHub.

Obter um item

Crie um módulo do Node.js com o nome de arquivo `ddb_getitem.js`. Não se esqueça de configurar o SDK conforme mostrado anteriormente. Para acessar o DynamoDB, crie um objeto de serviço do `AWS.DynamoDB`. Para identificar o item a ser obtido, você deve fornecer o valor da chave primária desse item na tabela. Por padrão, o método `getItem` retorna todos os valores de atributo definidos para o item. Para obter apenas um subconjunto de todos os valores de atributo possíveis, especifique uma expressão de projeção.

Crie um objeto JSON que contenha os parâmetros necessários para obter um item, que, neste exemplo, inclui o nome da tabela, o nome e o valor da chave do item que você está recebendo, e uma expressão de projeção que identifica o atributo do item que deseja recuperar. Chame o método `getItem` do objeto de serviço do DynamoDB.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the DynamoDB service object
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

var params = {
  TableName: "TABLE",
  Key: {
    KEY_NAME: { N: "001" },
  },
  ProjectionExpression: "ATTRIBUTE_NAME",
};

// Call DynamoDB to read the item from the table
ddb.getItem(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.Item);
  }
});
```

Para executar o exemplo, digite o seguinte na linha de comando.

```
node ddb_getitem.js
```

Este código de exemplo pode ser encontrado aqui no GitHub.

Excluir um item

Crie um módulo do Node.js com o nome de arquivo `ddb_deleteitem.js`. Não se esqueça de configurar o SDK conforme mostrado anteriormente. Para acessar o DynamoDB, crie um objeto de serviço do `AWS.DynamoDB`. Crie um objeto JSON com os parâmetros necessários para excluir um item, que, neste exemplo, inclui o nome da tabela, além do nome da chave e o valor do item que você está excluindo. Chame o método `deleteItem` do objeto de serviço do DynamoDB.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the DynamoDB service object
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

var params = {
  TableName: "TABLE",
  Key: {
    KEY_NAME: { N: "VALUE" },
  },
};

// Call DynamoDB to delete the item from the table
ddb.deleteItem(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

Para executar o exemplo, digite o seguinte na linha de comando.

```
node ddb_deleteitem.js
```

Este código de exemplo pode ser encontrado aqui no GitHub.

Ler e gravar itens em lote no DynamoDB

Este exemplo de código Node.js mostra:

- Como ler e gravar lotes de itens em uma tabela do DynamoDB.

O cenário

Neste exemplo, você usa uma série de módulos de Node.js para colocar um lote de itens em uma tabela do DynamoDB, bem como ler um lote de itens. O código usa o SDK para JavaScript para realizar operações de leitura e gravação em lote usando esses métodos da classe de cliente DynamoDB:

- batchGetItem
- batchWriteItem

Tarefas de pré-requisito

Para configurar e executar este exemplo, primeiro conclua estas tarefas:

- Instale o Node.js. Para obter mais informações, consulte o website Node.js.
- Crie um arquivo de configurações compartilhado com as credenciais de usuário. Para obter mais informações sobre como fornecer um arquivo de credenciais compartilhadas, consulte Carregar credenciais em Node.js do arquivo de credenciais compartilhado.
- Crie uma tabela do DynamoDB cujos itens você possa acessar. Para obter mais informações sobre como criar uma tabela do DynamoDB, consulte Criação e uso de tabelas no DynamoDB.

Ler itens em lote

Crie um módulo do Node.js com o nome de arquivo `ddb_batchgetitem.js`. Não se esqueça de configurar o SDK conforme mostrado anteriormente. Para acessar o DynamoDB, crie um objeto de serviço do `AWS.DynamoDB`. Crie um objeto JSON que contenha os parâmetros necessários para

obter um lote de itens, que, neste exemplo, inclui o nome de uma ou mais tabelas para leitura, os valores de chaves para leitura em cada tabela, e a expressão de projeção que especifica os atributos a serem retornados. Chame o método `batchGetItem` do objeto de serviço do DynamoDB.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create DynamoDB service object
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

var params = {
  RequestItems: {
    TABLE_NAME: {
      Keys: [
        { KEY_NAME: { N: "KEY_VALUE_1" } },
        { KEY_NAME: { N: "KEY_VALUE_2" } },
        { KEY_NAME: { N: "KEY_VALUE_3" } },
      ],
      ProjectionExpression: "KEY_NAME, ATTRIBUTE",
    },
  },
};

ddb.batchGetItem(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    data.Responses.TABLE_NAME.forEach(function (element, index, array) {
      console.log(element);
    });
  }
});
```

Para executar o exemplo, digite o seguinte na linha de comando.

```
node ddb_batchgetitem.js
```

Este código de exemplo pode ser encontrado aqui no GitHub.

Gravar itens em lote

Crie um módulo do Node.js com o nome de arquivo `ddb_batchwriteitem.js`. Não se esqueça de configurar o SDK conforme mostrado anteriormente. Para acessar o DynamoDB, crie um objeto de serviço do `AWS.DynamoDB`. Crie um objeto JSON que contenha os parâmetros necessários para obter um lote de itens, que, neste exemplo, inclui a tabela na qual você deseja gravar itens, a(s) chave(s) que você deseja gravar para cada item, e os atributos com os valores. Chame o método `batchWriteItem` do objeto de serviço do DynamoDB.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create DynamoDB service object
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

var params = {
  RequestItems: {
    TABLE_NAME: [
      {
        PutRequest: {
          Item: {
            KEY: { N: "KEY_VALUE" },
            ATTRIBUTE_1: { S: "ATTRIBUTE_1_VALUE" },
            ATTRIBUTE_2: { N: "ATTRIBUTE_2_VALUE" },
          },
        },
      },
      {
        PutRequest: {
          Item: {
            KEY: { N: "KEY_VALUE" },
            ATTRIBUTE_1: { S: "ATTRIBUTE_1_VALUE" },
            ATTRIBUTE_2: { N: "ATTRIBUTE_2_VALUE" },
          },
        },
      },
    ],
  },
};

ddb.batchWriteItem(params, function (err, data) {
```

```
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

Para executar o exemplo, digite o seguinte na linha de comando.

```
node ddb_batchwriteitem.js
```

Este código de exemplo pode ser encontrado aqui no GitHub.

Consultar e verificar uma tabela do DynamoDB

Este exemplo de código Node.js mostra:

- Como consultar e verificar uma tabela do DynamoDB em busca de itens.

O cenário

Consultar encontra itens em uma tabela ou um índice secundário usando apenas valores de atributo de chave primária. Você deve fornecer um nome da chave de partição e um valor a serem procurados. Você pode fornecer um nome da chave e um valor de classificação, além de usar um operador de comparação para refinar os resultados da pesquisa. Pesquisar encontra itens ao verificar todos os itens na tabela especificada.

Neste exemplo, você usa uma série de módulos Node.js para identificar um ou mais itens que queira recuperar de uma tabela do DynamoDB. O código usa o SDK para JavaScript para consultar e verificar tabelas usando esses métodos da classe de cliente do DynamoDB:

- query
- scan

Tarefas de pré-requisito

Para configurar e executar este exemplo, primeiro conclua estas tarefas:

- Instale o Node.js. Para obter mais informações, consulte o website Node.js.
- Crie um arquivo de configurações compartilhado com as credenciais de usuário. Para obter mais informações sobre como fornecer um arquivo de credenciais compartilhadas, consulte Carregar credenciais em Node.js do arquivo de credenciais compartilhado.
- Crie uma tabela do DynamoDB cujos itens você possa acessar. Para obter mais informações sobre como criar uma tabela do DynamoDB, consulte Criação e uso de tabelas no DynamoDB.

Consultar uma tabela

Este exemplo consulta uma tabela que contém informações do episódio sobre uma série de vídeos, retornando os títulos e as legendas do episódio da segunda temporada anteriores ao episódio 9 que contenham uma frase especificada na legenda.

Crie um módulo do Node.js com o nome de arquivo `ddb_query.js`. Não se esqueça de configurar o SDK conforme mostrado anteriormente. Para acessar o DynamoDB, crie um objeto de serviço do `AWS.DynamoDB`. Crie um objeto JSON que contenha os parâmetros necessários para consultar a tabela, que, neste exemplo, inclui o nome da tabela, o `ExpressionAttributeValues` necessário pela consulta, um `KeyConditionExpression` que usa esses valores para definir quais itens a consulta retorna, e os nomes dos valores de atributo a serem retornados para cada item. Chame o método `query` do objeto de serviço do DynamoDB.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create DynamoDB service object
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

var params = {
  ExpressionAttributeValues: {
    ":s": { N: "2" },
    ":e": { N: "09" },
    ":topic": { S: "PHRASE" },
  },
```

```
  KeyConditionExpression: "Season = :s and Episode > :e",
  ProjectionExpression: "Episode, Title, Subtitle",
  FilterExpression: "contains (Subtitle, :topic)",
  TableName: "EPISODES_TABLE",
};

ddb.query(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    //console.log("Success", data.Items);
    data.Items.forEach(function (element, index, array) {
      console.log(element.Title.S + " (" + element.Subtitle.S + ")");
    });
  }
});
```

Para executar o exemplo, digite o seguinte na linha de comando.

```
node ddb_query.js
```

Este código de exemplo pode ser encontrado aqui no GitHub.

Verificação de uma tabela

Crie um módulo do Node.js com o nome de arquivo `ddb_scan.js`. Não se esqueça de configurar o SDK conforme mostrado anteriormente. Para acessar o DynamoDB, crie um objeto de serviço do `AWS.DynamoDB`. Crie um objeto JSON que contenha os parâmetros necessários para verificar a tabela em busca de itens, que, neste exemplo, inclui o nome da tabela, a lista de valores de atributos a serem retornados para cada item correspondente e uma expressão para filtrar o conjunto de resultados a fim de encontrar itens que contenham uma frase especificada. Chame o método `scan` do objeto de serviço do DynamoDB.

```
// Load the AWS SDK for Node.js.
var AWS = require("aws-sdk");
// Set the AWS Region.
AWS.config.update({ region: "REGION" });

// Create DynamoDB service object.
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });
```

```
const params = {
  // Specify which items in the results are returned.
  FilterExpression: "Subtitle = :topic AND Season = :s AND Episode = :e",
  // Define the expression attribute value, which are substitutes for the values you
 want to compare.
  ExpressionAttributeValues: {
    ":topic": { S: "SubTitle2" },
    ":s": { N: 1 },
    ":e": { N: 2 },
  },
  // Set the projection expression, which are the attributes that you want.
  ProjectionExpression: "Season, Episode, Title, Subtitle",
  TableName: "EPISODES_TABLE",
};

ddb.scan(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
    data.Items.forEach(function (element, index, array) {
      console.log(
        "printing",
        element.Title.S + " (" + element.Subtitle.S + ")"
      );
    });
  }
});
```

Para executar o exemplo, digite o seguinte na linha de comando.

```
node ddb_scan.js
```

Este código de exemplo pode ser encontrado aqui no GitHub.

Uso do cliente de documentos do DynamoDB

Este exemplo de código Node.js mostra:

- Como acessar uma tabela do usando o cliente de documento.

O cenário

O cliente de documento do DynamoDB simplifica o trabalho com itens abstraindo a noção de valores de atributo. Essa abstração anota tipos JavaScript nativos fornecidos como parâmetros de entrada, bem como converte dados de resposta anotados em tipos JavaScript nativos.

Para obter mais informações sobre a classe de cliente de documento do DynamoDB, consulte `AWS.DynamoDB.DocumentClient` na Referência de API. Para obter mais informações sobre programação com o Amazon DynamoDB, consulte Programação com o DynamoDB no Guia do Desenvovledor Amazon DynamoDB.

Neste exemplo, você usa uma série de módulos de Node.js para realizar operações básicas em uma tabela do DynamoDB usando o cliente de documento. O código usa o SDK para JavaScript para consultar e verificar tabelas usando esses métodos da classe de cliente de documento do DynamoDB:

- get
- put
- update
- query
- delete

Tarefas de pré-requisito

Para configurar e executar este exemplo, primeiro conclua estas tarefas:

- Instale o Node.js. Para obter mais informações, consulte o website Node.js.
- Crie um arquivo de configurações compartilhado com as credenciais de usuário. Para obter mais informações sobre como fornecer um arquivo de credenciais compartilhadas, consulte Carregar credenciais em Node.js do arquivo de credenciais compartilhado.
- Crie uma tabela do DynamoDB cujos itens você possa acessar. Para obter mais informações sobre como criar uma tabela do DynamoDB usando o SDK para JavaScript, consulte Criação e uso de tabelas no DynamoDB. Também é possível usar o console do para criar uma tabela.

Obtenção de um item de uma tabela

Crie um módulo do Node.js com o nome de arquivo `ddbdoc_get.js`. Não se esqueça de configurar o SDK conforme mostrado anteriormente. Para acessar o DynamoDB, crie um objeto do `AWS.DynamoDB.DocumentClient`. Crie um objeto JSON que contenha os parâmetros necessários para obter um item da tabela, que, neste exemplo, inclui o nome da tabela, o nome da chave hash nessa tabela e o valor da chave hash do item que você deseja receber. Chame o método `get` do cliente de documento do DynamoDB.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create DynamoDB document client
var docClient = new AWS.DynamoDB.DocumentClient({ apiVersion: "2012-08-10" });

var params = {
  TableName: "EPISODES_TABLE",
  Key: { KEY_NAME: VALUE },
};

docClient.get(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.Item);
  }
});
```

Para executar o exemplo, digite o seguinte na linha de comando.

```
node ddbdoc_get.js
```

Este código de exemplo pode ser encontrado aqui no GitHub.

Colocar um item em uma tabela

Crie um módulo do Node.js com o nome de arquivo `ddbdoc_put.js`. Não se esqueça de configurar o SDK conforme mostrado anteriormente. Para acessar o DynamoDB, crie um objeto do `AWS.DynamoDB.DocumentClient`. Crie um objeto JSON que contenha os parâmetros necessários

para gravar um item na tabela, que, neste exemplo, inclui o nome da tabela e uma descrição do item a ser adicionado ou atualizado que inclua a chave hash e o valor, bem como nomes e valores de atributos a serem definidos no item. Chame o método `put` do cliente de documento do DynamoDB.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create DynamoDB document client
var docClient = new AWS.DynamoDB.DocumentClient({ apiVersion: "2012-08-10" });

var params = {
  TableName: "TABLE",
  Item: {
    HASHKEY: VALUE,
    ATTRIBUTE_1: "STRING_VALUE",
    ATTRIBUTE_2: VALUE_2,
  },
};

docClient.put(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

Para executar o exemplo, digite o seguinte na linha de comando.

```
node ddbdoc_put.js
```

Este código de exemplo pode ser encontrado aqui no GitHub.

Atualizar um item em uma tabela

Crie um módulo do Node.js com o nome de arquivo `ddbdoc_update.js`. Não se esqueça de configurar o SDK conforme mostrado anteriormente. Para acessar o DynamoDB, crie um objeto do `AWS.DynamoDB.DocumentClient`. Crie um objeto JSON que contenha os parâmetros necessários para gravar um item na tabela, que, neste exemplo, inclui o nome da tabela, a chave do item a ser

atualizada, um conjunto de `UpdateExpressions` que definem os atributos do item a ser atualizado com tokens com valores atribuídos nos parâmetros `ExpressionAttributeValues`. Chame o método `update` do cliente de documento do DynamoDB.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create DynamoDB document client
var docClient = new AWS.DynamoDB.DocumentClient({ apiVersion: "2012-08-10" });

// Create variables to hold numeric key values
var season = SEASON_NUMBER;
var episode = EPISODES_NUMBER;

var params = {
  TableName: "EPISODES_TABLE",
  Key: {
    Season: season,
    Episode: episode,
  },
  UpdateExpression: "set Title = :t, Subtitle = :s",
  ExpressionAttributeValues: {
    ":t": "NEW_TITLE",
    ":s": "NEW_SUBTITLE",
  },
};

docClient.update(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

Para executar o exemplo, digite o seguinte na linha de comando.

```
node ddbdoc_update.js
```

Este código de exemplo pode ser encontrado aqui no GitHub.

Consultar uma tabela

Este exemplo consulta uma tabela que contém informações do episódio sobre uma série de vídeos, retornando os títulos e as legendas do episódio da segunda temporada anteriores ao episódio 9 que contenham uma frase especificada na legenda.

Crie um módulo do Node.js com o nome de arquivo `ddbdoc_query.js`. Não se esqueça de configurar o SDK conforme mostrado anteriormente. Para acessar o DynamoDB, crie um objeto do `AWS.DynamoDB.DocumentClient`. Crie um objeto JSON que contenha os parâmetros necessários para consultar a tabela, que, neste exemplo, inclui o nome da tabela, o `ExpressionAttributeValues` necessário pela consulta e um `KeyConditionExpression` que usa esses valores para definir quais itens a consulta retorna. Chame o método `query` do cliente de documento do DynamoDB.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create DynamoDB document client
var docClient = new AWS.DynamoDB.DocumentClient({ apiVersion: "2012-08-10" });

var params = {
  ExpressionAttributeValues: {
    ":s": 2,
    ":e": 9,
    ":topic": "PHRASE",
  },
  KeyConditionExpression: "Season = :s and Episode > :e",
  FilterExpression: "contains (Subtitle, :topic)",
  TableName: "EPISODES_TABLE",
};

docClient.query(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.Items);
  }
});
```

Para executar o exemplo, digite o seguinte na linha de comando.

```
node ddbdoc_query.js
```

Este código de exemplo pode ser encontrado aqui no GitHub.

Excluir um item de uma tabela

Crie um módulo do Node.js com o nome de arquivo `ddbdoc_delete.js`. Não se esqueça de configurar o SDK conforme mostrado anteriormente. Para acessar o DynamoDB, crie um objeto do `AWS.DynamoDB.DocumentClient`. Crie um objeto JSON que contenha os parâmetros necessários para excluir um item na tabela, que, neste exemplo, inclui o nome da tabela, bem como o nome e o valor da chave hash do item que você deseja excluir. Chame o método `delete` do cliente de documento do DynamoDB.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create DynamoDB document client
var docClient = new AWS.DynamoDB.DocumentClient({ apiVersion: "2012-08-10" });

var params = {
  Key: {
    HASH_KEY: VALUE,
  },
  TableName: "TABLE",
};

docClient.delete(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

Para executar o exemplo, digite o seguinte na linha de comando.

```
node ddbdoc_delete.js
```

Este código de exemplo pode ser encontrado aqui no GitHub.

Exemplos do Amazon EC2

O Amazon Elastic Compute Cloud (Amazon EC2) é um serviço Web que oferece hospedagem de servidor virtual na nuvem. Ele foi projetado para facilitar a computação em nuvem em escala da web para desenvolvedores fornecendo capacidade computacional redimensionável.

A API JavaScript do Amazon EC2 é exposta por meio da classe de cliente `AWS.EC2`. Para obter mais informações sobre como usar a classe de cliente do Amazon EC2, consulte `Class: AWS.EC2` na referência da API.

Tópicos

- Criar uma instância do Amazon EC2
- Gerenciar instâncias do Amazon EC2
- Trabalhar com pares de chaves do Amazon EC2
- Uso de regiões e zonas de disponibilidade para o Amazon EC2
- Trabalhar com grupos de segurança no Amazon EC2
- Usar endereços IP elásticos no Amazon EC2

Criar uma instância do Amazon EC2

Este exemplo de código Node.js mostra:

- Como criar uma instância do Amazon EC2 com base em uma Imagem de máquina da Amazon (AMI) pública.
- Como criar e atribuir tags à nova instância do Amazon EC2.

Sobre o exemplo

Neste exemplo, você usa um módulo Node.js para criar uma instância do Amazon EC2 e atribuir um par de chaves e tags a ela. O código usa o SDK para JavaScript para criar e marcar uma instância usando esses métodos da classe de cliente do Amazon EC2:

- `runInstances`
- `createTags`

Tarefas de pré-requisito

Para configurar e executar este exemplo, primeiro conclua estas tarefas.

- Instale o Node.js. Para obter mais informações, consulte o website Node.js.
- Crie um arquivo de configurações compartilhado com as credenciais de usuário. Para obter mais informações sobre como fornecer um arquivo de credenciais compartilhadas, consulte Carregar credenciais em Node.js do arquivo de credenciais compartilhado.
- Criar um par de chaves. Para obter detalhes, consulte Trabalhar com pares de chaves do Amazon EC2. Você usa o nome do par de chaves neste exemplo.

Criar e identificar uma instância

Crie um módulo do Node.js com o nome de arquivo `ec2_createinstances.js`. Não se esqueça de configurar o SDK conforme mostrado anteriormente.

Crie um objeto para passar os parâmetros do método `runInstances` da classe de cliente AWS.EC2, inclusive o nome do par de chaves a ser atribuído e o ID da AMI a ser executada. Para chamar o método `runInstances`, crie uma promessa para invocar um objeto de serviço do Amazon EC2 passando os parâmetros. Depois, lide com a resposta no retorno de chamada da promessa.

O código a seguir adiciona uma tag `Name` a uma nova instância, que o console do Amazon EC2 reconhece e exibe no campo Name (Nome) da lista de instâncias. Adicione até 50 tags a uma instância, todas podendo ser adicionadas em uma única chamada ao método `createTags`.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Load credentials and set region from JSON file
AWS.config.update({ region: "REGION" });

// Create EC2 service object
var ec2 = new AWS.EC2({ apiVersion: "2016-11-15" });

// AMI is amzn-ami-2011.09.1.x86_64-ebs
var instanceParams = {
  ImageId: "AMI_ID",
  InstanceType: "t2.micro",
  KeyName: "KEY_PAIR_NAME",
  MinCount: 1,
  MaxCount: 1,
};

// Create a promise on an EC2 service object
var instancePromise = new AWS.EC2({ apiVersion: "2016-11-15" })
  .runInstances(instanceParams)
  .promise();

// Handle promise's fulfilled/rejected states
instancePromise
  .then(function (data) {
    console.log(data);
    var instanceId = data.Instances[0].InstanceId;
    console.log("Created instance", instanceId);
    // Add tags to the instance
    tagParams = {
      Resources: [instanceId],
      Tags: [
        {
          Key: "Name",
          Value: "SDK Sample",
        },
      ],
    };
    // Create a promise on an EC2 service object
    var tagPromise = new AWS.EC2({ apiVersion: "2016-11-15" })
      .createTags(tagParams)
      .promise();
    // Handle promise's fulfilled/rejected states
```

```
    tagPromise
      .then(function (data) {
        console.log("Instance tagged");
      })
      .catch(function (err) {
        console.error(err, err.stack);
      });
  })
  .catch(function (err) {
    console.error(err, err.stack);
  });
```

Para executar o exemplo, digite o seguinte na linha de comando.

```
node ec2_createinstances.js
```

Este código de exemplo pode ser encontrado aqui no GitHub.

Gerenciar instâncias do Amazon EC2

Este exemplo de código Node.js mostra:

- Como recuperar informações básicas sobre as instâncias do Amazon EC2.
- Como iniciar e parar o monitoramento detalhado de uma instância do Amazon EC2.
- Como iniciar e parar uma instância do Amazon EC2.
- Como reinicializar uma instância do Amazon EC2

O cenário

Neste exemplo, você usa uma série de módulos de Node.js para realizar várias operações de gerenciamento de instâncias básicas. Os módulos Node.js usam o SDK para JavaScript para gerenciar instâncias usando os métodos da classe de cliente do Amazon EC2:

- `describeInstances`

- `monitorInstances`
- `unmonitorInstances`
- `startInstances`
- `stopInstances`
- `rebootInstances`

Para obter mais informações sobre o ciclo de vida de instâncias do Amazon EC2, consulte Instance Lifecycle no Guia do usuário do Amazon EC2.

Tarefas de pré-requisito

Para configurar e executar este exemplo, primeiro conclua estas tarefas:

- Instale o Node.js. Para obter mais informações sobre como instalar Node.js, consulte o website de Node.js.
- Crie um arquivo de configurações compartilhado com as credenciais de usuário. Para obter mais informações sobre como fornecer um arquivo de credenciais compartilhadas, consulte Carregar credenciais em Node.js do arquivo de credenciais compartilhado.
- Crie uma instância do Amazon EC2. Para obter mais informações sobre como criar instâncias do Amazon EC2, consulte Amazon EC2 Instances no Guia do usuário do Amazon EC2 ou Amazon EC2 Instances no Guia do usuário do Amazon EC2.

Descrever as instâncias

Crie um módulo do Node.js com o nome de arquivo `ec2_describeinstances.js`. Não se esqueça de configurar o SDK conforme mostrado anteriormente. Para acessar o Amazon EC2, crie um objeto de serviço do `AWS.EC2`. Chame o método `describeInstances` do objeto de serviço do Amazon EC2 para recuperar uma descrição detalhada das instâncias.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create EC2 service object
var ec2 = new AWS.EC2({ apiVersion: "2016-11-15" });
```

```
var params = {
  DryRun: false,
};

// Call EC2 to retrieve policy for selected bucket
ec2.describeInstances(params, function (err, data) {
  if (err) {
    console.log("Error", err.stack);
  } else {
    console.log("Success", JSON.stringify(data));
  }
});
```

Para executar o exemplo, digite o seguinte na linha de comando.

```
node ec2_describeinstances.js
```

Este código de exemplo pode ser encontrado aqui no GitHub.

Gerenciar monitoramento de instâncias

Crie um módulo do Node.js com o nome de arquivo `ec2_monitorinstances.js`. Não se esqueça de configurar o SDK conforme mostrado anteriormente. Para acessar o Amazon EC2, crie um objeto de serviço do `AWS.EC2`. Adicione os IDs às instâncias cujo monitoramento você deseja controlar.

Com base no valor de um argumento de linha de comando (`ON` ou `OFF`), chame o método `monitorInstances` do objeto de serviço do Amazon EC2 para iniciar o monitoramento detalhado das instâncias especificadas ou chame o método un`monitorInstances`. Use o parâmetro `DryRun` a fim de testar se você tem permissão para alterar o monitoramento de instâncias antes de tentar alterar o monitoramento dessas instâncias.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create EC2 service object
var ec2 = new AWS.EC2({ apiVersion: "2016-11-15" });

var params = {
  InstanceIds: ["INSTANCE_ID"],
```

```
  DryRun: true,
};

if (process.argv[2].toUpperCase() === "ON") {
  // Call EC2 to start monitoring the selected instances
  ec2.monitorInstances(params, function (err, data) {
    if (err && err.code === "DryRunOperation") {
      params.DryRun = false;
      ec2.monitorInstances(params, function (err, data) {
        if (err) {
          console.log("Error", err);
        } else if (data) {
          console.log("Success", data.InstanceMonitorings);
        }
      });
    } else {
      console.log("You don't have permission to change instance monitoring.");
    }
  });
} else if (process.argv[2].toUpperCase() === "OFF") {
  // Call EC2 to stop monitoring the selected instances
  ec2.unmonitorInstances(params, function (err, data) {
    if (err && err.code === "DryRunOperation") {
      params.DryRun = false;
      ec2.unmonitorInstances(params, function (err, data) {
        if (err) {
          console.log("Error", err);
        } else if (data) {
          console.log("Success", data.InstanceMonitorings);
        }
      });
    } else {
      console.log("You don't have permission to change instance monitoring.");
    }
  });
}
```

Para executar o exemplo, digite o seguinte na linha de comando, especificando `ON` para iniciar o monitoramento detalhado ou `OFF` para interrompê-lo.

```
node ec2_monitorinstances.js ON
```

Este código de exemplo pode ser encontrado aqui no GitHub.

Iniciar e parar instâncias

Crie um módulo do Node.js com o nome de arquivo `ec2_startstopinstances.js`. Não se esqueça de configurar o SDK conforme mostrado anteriormente. Para acessar o Amazon EC2, crie um objeto de serviço do `AWS.EC2`. Adicione os IDs às instâncias que você deseja iniciar ou parar.

Com base no valor de um argumento de linha de comando (`START` ou `STOP`), chame o `startInstances` método do objeto de serviço do Amazon EC2 para iniciar as instâncias especificadas ou chame o método `stopInstances` a fim de pará-las. Use o parâmetro `DryRun` para testar se você tem permissão antes de realmente tentar iniciar ou parar instâncias selecionadas.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create EC2 service object
var ec2 = new AWS.EC2({ apiVersion: "2016-11-15" });

var params = {
  InstanceIds: [process.argv[3]],
  DryRun: true,
};

if (process.argv[2].toUpperCase() === "START") {
  // Call EC2 to start the selected instances
  ec2.startInstances(params, function (err, data) {
    if (err && err.code === "DryRunOperation") {
      params.DryRun = false;
      ec2.startInstances(params, function (err, data) {
        if (err) {
          console.log("Error", err);
        } else if (data) {
          console.log("Success", data.StartingInstances);
        }
      });
    } else {
      console.log("You don't have permission to start instances.");
    }
  });
} else if (process.argv[2].toUpperCase() === "STOP") {
  // Call EC2 to stop the selected instances
  ec2.stopInstances(params, function (err, data) {
```

```
    if (err && err.code === "DryRunOperation") {
      params.DryRun = false;
      ec2.stopInstances(params, function (err, data) {
        if (err) {
          console.log("Error", err);
        } else if (data) {
          console.log("Success", data.StoppingInstances);
        }
      });
    } else {
      console.log("You don't have permission to stop instances");
    }
  });
}
```

Para executar o exemplo, digite o seguinte na linha de comando, especificando `START` para iniciar as instâncias ou `STOP` a fim de pará-las.

```
node ec2_startstopinstances.js START INSTANCE_ID
```

Este código de exemplo pode ser encontrado aqui no GitHub.

Reinicializar instâncias

Crie um módulo do Node.js com o nome de arquivo `ec2_rebootinstances.js`. Não se esqueça de configurar o SDK conforme mostrado anteriormente. Para acessar o Amazon EC2, crie um objeto de serviço do Amazon EC2. Adicione os IDs às instâncias que você deseja reinicializar. Chame o método `rebootInstances` do objeto de serviço `AWS.EC2` para reinicializar as instâncias especificadas. Use o parâmetro `DryRun` para testar se você tem permissão para reinicializar essas instâncias antes de realmente tentar reinicializá-las.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create EC2 service object
var ec2 = new AWS.EC2({ apiVersion: "2016-11-15" });

var params = {
  InstanceIds: ["INSTANCE_ID"],
  DryRun: true,
```

```
};

// Call EC2 to reboot instances
ec2.rebootInstances(params, function (err, data) {
  if (err && err.code === "DryRunOperation") {
    params.DryRun = false;
    ec2.rebootInstances(params, function (err, data) {
      if (err) {
        console.log("Error", err);
      } else if (data) {
        console.log("Success", data);
      }
    });
  } else {
    console.log("You don't have permission to reboot instances.");
  }
});
```

Para executar o exemplo, digite o seguinte na linha de comando.

```
node ec2_rebootinstances.js
```

Este código de exemplo pode ser encontrado aqui no GitHub.

Trabalhar com pares de chaves do Amazon EC2

Este exemplo de código Node.js mostra:

- Como recuperar informações sobre os pares de chaves.
- Como criar um par de chaves para acessar a instância do Amazon EC2.
- Como excluir um par de chaves existente.

O cenário

O Amazon EC2 utiliza criptografia de chave pública para criptografar e descriptografar as informações de login. A criptografia de chave pública utiliza uma chave pública para criptografar

dados, em seguida o destinatário usa a chave privada para descriptografar os dados. As chaves pública e privada são conhecidas como par de chaves.

Neste exemplo, você usa uma série de módulos de Node.js para realizar várias operações de gerenciamento de pares de chaves do Amazon EC2. Os módulos Node.js usam o SDK para JavaScript para gerenciar instâncias usando os métodos da classe de cliente do Amazon EC2:

- `createKeyPair`
- `deleteKeyPair`
- `describeKeyPairs`

Para obter mais informações sobre os pares de chaves do Amazon EC2, consulte Amazon EC2 Key Pairs no Guia do usuário do Amazon EC2 ou Amazon EC2 Key Pairs and Windows Instances no Guia do usuário do Amazon EC2.

Tarefas de pré-requisito

Para configurar e executar este exemplo, primeiro conclua estas tarefas:

- Instale o Node.js. Para obter mais informações sobre como instalar Node.js, consulte o website de Node.js.
- Crie um arquivo de configurações compartilhado com as credenciais de usuário. Para obter mais informações sobre como fornecer um arquivo de credenciais compartilhadas, consulte Carregar credenciais em Node.js do arquivo de credenciais compartilhado.

Descrever os pares de chaves

Crie um módulo do Node.js com o nome de arquivo `ec2_describekeypairs.js`. Não se esqueça de configurar o SDK conforme mostrado anteriormente. Para acessar o Amazon EC2, crie um objeto de serviço do `AWS.EC2`. Crie um objeto JSON vazio para manter os parâmetros exigidos pelo método `describeKeyPairs` para retornar descrições de todos os pares de chaves. Também é possível fornecer uma matriz de nomes de pares de chaves na parte `KeyName` dos parâmetros no arquivo JSON para o método `describeKeyPairs`.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });
```

```
// Create EC2 service object
var ec2 = new AWS.EC2({ apiVersion: "2016-11-15" });

// Retrieve key pair descriptions; no params needed
ec2.describeKeyPairs(function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", JSON.stringify(data.KeyPairs));
  }
});
```

Para executar o exemplo, digite o seguinte na linha de comando.

```
node ec2_describekeypairs.js
```

Este código de exemplo pode ser encontrado aqui no GitHub.

Criação de um par de chaves

Cada par de chaves exige um nome. O Amazon EC2 associa a chave pública ao nome especificado como o nome da chave. Crie um módulo do Node.js com o nome de arquivo `ec2_createkeypair.js`. Não se esqueça de configurar o SDK conforme mostrado anteriormente. Para acessar o Amazon EC2, crie um objeto de serviço do `AWS.EC2`. Crie os parâmetros JSON para especificar o nome do par de chaves e os passe para chamar o método `createKeyPair`.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create EC2 service object
var ec2 = new AWS.EC2({ apiVersion: "2016-11-15" });

var params = {
  KeyName: "KEY_PAIR_NAME",
};

// Create the key pair
ec2.createKeyPair(params, function (err, data) {
  if (err) {
```

```
    console.log("Error", err);
  } else {
    console.log(JSON.stringify(data));
  }
});
```

Para executar o exemplo, digite o seguinte na linha de comando.

```
node ec2_createkeypair.js
```

Este código de exemplo pode ser encontrado aqui no GitHub.

Excluir um par de chaves

Crie um módulo do Node.js com o nome de arquivo `ec2_deletekeypair.js`. Não se esqueça de configurar o SDK conforme mostrado anteriormente. Para acessar o Amazon EC2, crie um objeto de serviço do `AWS.EC2`. Crie os parâmetros JSON para especificar o nome do par de chaves que você deseja excluir. Depois, chame o método `deleteKeyPair`.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create EC2 service object
var ec2 = new AWS.EC2({ apiVersion: "2016-11-15" });

var params = {
  KeyName: "KEY_PAIR_NAME",
};

// Delete the key pair
ec2.deleteKeyPair(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Key Pair Deleted");
  }
});
```

Para executar o exemplo, digite o seguinte na linha de comando.

```
node ec2_deletekeypair.js
```

Este código de exemplo pode ser encontrado aqui no GitHub.

Uso de regiões e zonas de disponibilidade para o Amazon EC2

Este exemplo de código Node.js mostra:

- Como recuperar descrições de regiões e zonas de disponibilidade.

O cenário

O Amazon EC2 está hospedado em vários locais no mundo todo. Esses locais são compostos por regiões da e zonas de disponibilidade. Cada região da é uma área geográfica separada. Cada região contém vários locais isolados conhecidos como Zonas de Disponibilidade. O Amazon EC2 oferece a capacidade de alocar instâncias e dados em diversos locais.

Neste exemplo, você usa uma série de módulos do Node.js para recuperar detalhes sobre regiões e zonas de disponibilidade. Os módulos Node.js usam o SDK para JavaScript para gerenciar instâncias usando os seguintes métodos da classe de cliente do Amazon EC2:

- `describeAvailabilityZones`
- `describeRegions`

Para obter mais informações sobre as regiões e zonas de disponibilidade, consulte Regions and Availability Zones no Guia do usuário do Amazon EC2 ou Regions and Availability Zones no Guia do usuário do Amazon EC2.

Tarefas de pré-requisito

Para configurar e executar este exemplo, você deve primeiro concluir estas tarefas:

- Instale o Node.js. Para obter mais informações sobre como instalar Node.js, consulte o website de Node.js.

- Crie um arquivo de configurações compartilhado com as credenciais de usuário. Para obter mais informações sobre como fornecer um arquivo de credenciais compartilhadas, consulte Carregar credenciais em Node.js do arquivo de credenciais compartilhado.

Descrever regiões e zonas de disponibilidade

Crie um módulo do Node.js com o nome de arquivo `ec2_describeregionsandzones.js`. Não se esqueça de configurar o SDK conforme mostrado anteriormente. Para acessar o Amazon EC2, crie um objeto de serviço do `AWS.EC2`. Crie um objeto JSON vazio a ser passado como parâmetros, que retorna todas as descrições disponíveis. Depois, chame os métodos `describeRegions` e `describeAvailabilityZones`.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create EC2 service object
var ec2 = new AWS.EC2({ apiVersion: "2016-11-15" });

var params = {};

// Retrieves all regions/endpoints that work with EC2
ec2.describeRegions(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Regions: ", data.Regions);
  }
});

// Retrieves availability zones only for region of the ec2 service object
ec2.describeAvailabilityZones(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Availability Zones: ", data.AvailabilityZones);
  }
});
```

Para executar o exemplo, digite o seguinte na linha de comando.

```
node ec2_describeregionsandzones.js
```

Este código de exemplo pode ser encontrado aqui no GitHub.

Trabalhar com grupos de segurança no Amazon EC2

Este exemplo de código Node.js mostra:

- Como recuperar informações sobre os grupos de segurança.
- Como criar um grupo de segurança para acessar uma instância do Amazon EC2.
- Como excluir um grupo de segurança existente.

O cenário

Um grupo de segurança do Amazon EC2 atua como um firewall virtual que controla o tráfego para uma ou mais instâncias. Você adiciona regras a cada grupo de segurança que permite tráfego de entrada ou de saída das instâncias associadas. Você pode modificar as regras para um security group a qualquer momento. As novas regras são aplicadas automaticamente a todas as instâncias associadas ao security group.

Neste exemplo, você usa uma série de módulos de Node.js para realizar várias operações do Amazon EC2 envolvendo grupos de segurança. Os módulos Node.js usam o SDK para JavaScript para gerenciar instâncias usando os seguintes métodos da classe de cliente do Amazon EC2:

- `describeSecurityGroups`
- `authorizeSecurityGroupIngress`
- `createSecurityGroup`
- `describeVpcs`
- `deleteSecurityGroup`

Para obter mais informações sobre os grupos de segurança do Amazon EC2, consulte Amazon EC2 Amazon Security Groups for Linux Instances no Guia do usuário do Amazon EC2 ou Amazon EC2 Security Groups for Windows Instances no Guia do usuário do Amazon EC2.

Tarefas de pré-requisito

Para configurar e executar este exemplo, primeiro conclua estas tarefas:

- Instale o Node.js. Para obter mais informações sobre como instalar Node.js, consulte o website de Node.js.
- Crie um arquivo de configurações compartilhado com as credenciais de usuário. Para obter mais informações sobre como fornecer um arquivo de credenciais compartilhadas, consulte Carregar credenciais em Node.js do arquivo de credenciais compartilhado.

Como descrever security groups

Crie um módulo do Node.js com o nome de arquivo `ec2_describesecuritygroups.js`. Não se esqueça de configurar o SDK conforme mostrado anteriormente. Para acessar o Amazon EC2, crie um objeto de serviço do `AWS.EC2`. Crie um objeto JSON a ser passado como parâmetro, inclusive os IDs dos grupos de segurança que você deseja descrever. Depois, chame o método `describeSecurityGroups` do objeto de serviço do Amazon EC2.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create EC2 service object
var ec2 = new AWS.EC2({ apiVersion: "2016-11-15" });

var params = {
  GroupIds: ["SECURITY_GROUP_ID"],
};

// Retrieve security group descriptions
ec2.describeSecurityGroups(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", JSON.stringify(data.SecurityGroups));
```

```
  }
});
```

Para executar o exemplo, digite o seguinte na linha de comando.

```
node ec2_describesecuritygroups.js
```

Este código de exemplo pode ser encontrado aqui no GitHub.

Criar um grupo de segurança e regras

Crie um módulo do Node.js com o nome de arquivo `ec2_createsecuritygroup.js`. Não se esqueça de configurar o SDK conforme mostrado anteriormente. Para acessar o Amazon EC2, crie um objeto de serviço do `AWS.EC2`. Crie um objeto JSON para os parâmetros que especificam o nome do grupo de segurança, uma descrição e o ID da VPC. Passe os parâmetros para o método `createSecurityGroup`.

Depois de criar o grupo de segurança com êxito, você poderá definir regras para permitir o tráfego de entrada. Crie um objeto JSON para os parâmetros que especificam o protocolo IP e as portas de entrada em que a instância do Amazon EC2 receberá o tráfego. Passe os parâmetros para o método `authorizeSecurityGroupIngress`.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Load credentials and set region from JSON file
AWS.config.update({ region: "REGION" });

// Create EC2 service object
var ec2 = new AWS.EC2({ apiVersion: "2016-11-15" });

// Variable to hold a ID of a VPC
var vpc = null;

// Retrieve the ID of a VPC
ec2.describeVpcs(function (err, data) {
  if (err) {
    console.log("Cannot retrieve a VPC", err);
  } else {
    vpc = data.Vpcs[0].VpcId;
    var paramsSecurityGroup = {
      Description: "DESCRIPTION",
      GroupName: "SECURITY_GROUP_NAME",
```

```
      VpcId: vpc,
    };
    // Create the instance
    ec2.createSecurityGroup(paramsSecurityGroup, function (err, data) {
      if (err) {
        console.log("Error", err);
      } else {
        var SecurityGroupId = data.GroupId;
        console.log("Success", SecurityGroupId);
        var paramsIngress = {
          GroupId: "SECURITY_GROUP_ID",
          IpPermissions: [
            {
              IpProtocol: "tcp",
              FromPort: 80,
              ToPort: 80,
              IpRanges: [{ CidrIp: "0.0.0.0/0" }],
            },
            {
              IpProtocol: "tcp",
              FromPort: 22,
              ToPort: 22,
              IpRanges: [{ CidrIp: "0.0.0.0/0" }],
            },
          ],
        };
        ec2.authorizeSecurityGroupIngress(paramsIngress, function (err, data) {
          if (err) {
            console.log("Error", err);
          } else {
            console.log("Ingress Successfully Set", data);
          }
        });
      }
    });
  }
});
```

Para executar o exemplo, digite o seguinte na linha de comando.

```
node ec2_createsecuritygroup.js
```

Este código de exemplo pode ser encontrado aqui no GitHub.

Excluir um grupo de segurança

Crie um módulo do Node.js com o nome de arquivo `ec2_deletesecuritygroup.js`. Não se esqueça de configurar o SDK conforme mostrado anteriormente. Para acessar o Amazon EC2, crie um objeto de serviço do `AWS.EC2`. Crie os parâmetros JSON para especificar o nome do grupo de segurança a ser excluído. Depois, chame o método `deleteSecurityGroup`.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create EC2 service object
var ec2 = new AWS.EC2({ apiVersion: "2016-11-15" });

var params = {
  GroupId: "SECURITY_GROUP_ID",
};

// Delete the security group
ec2.deleteSecurityGroup(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Security Group Deleted");
  }
});
```

Para executar o exemplo, digite o seguinte na linha de comando.

```
node ec2_deletesecuritygroup.js
```

Este código de exemplo pode ser encontrado aqui no GitHub.

Usar endereços IP elásticos no Amazon EC2

Este exemplo de código Node.js mostra:

- Como recuperar descrições dos endereços IP elásticos.
- Como alocar e liberar um endereço IP elástico.
- Como associar um endereço IP elástico a uma instância do Amazon EC2.

O cenário

Um endereço IP elástico é um endereço IP estático projetado para computação em nuvem dinâmica. Um endereço IP elástico está associado à conta da AWS. Trata-se de um endereço IP público, que pode ser acessado na Internet. Se a instância não tiver um endereço IP público, você poderá associar um endereço IP elástico a ela para permitir a comunicação com a Internet.

Neste exemplo, você usa uma série de módulos de Node.js para realizar várias operações do Amazon EC2 envolvendo endereços IP elásticos. Os módulos Node.js usam o SDK para JavaScript para gerenciar endereços IP elásticos usando esses métodos da classe de cliente do Amazon EC2:

- `describeAddresses`
- `allocateAddress`
- `associateAddress`
- `releaseAddress`

Para obter mais informações sobre os endereços IP elásticos no Amazon EC2, consulte Elastic IP Addresses no Guia do usuário do Amazon EC2 ou Elastic IP Addresses no Guia do usuário do Amazon EC2.

Tarefas de pré-requisito

Para configurar e executar este exemplo, primeiro conclua estas tarefas:

- Instale o Node.js. Para obter mais informações sobre como instalar Node.js, consulte o website de Node.js.
- Crie um arquivo de configurações compartilhado com as credenciais de usuário. Para obter mais informações sobre como fornecer um arquivo de credenciais compartilhadas, consulte Carregar credenciais em Node.js do arquivo de credenciais compartilhado.
- Crie uma instância do Amazon EC2. Para obter mais informações sobre como criar instâncias do Amazon EC2, consulte Amazon EC2 Instances no Guia do usuário do Amazon EC2 ou Amazon EC2 Instances no Guia do usuário do Amazon EC2.

Descrever endereços IP elásticos

Crie um módulo do Node.js com o nome de arquivo `ec2_describeaddresses.js`. Não se esqueça de configurar o SDK conforme mostrado anteriormente. Para acessar o Amazon EC2, crie um objeto de serviço do `AWS.EC2`. Crie um objeto JSON a ser passado como parâmetro, filtrando os endereços retornados por aqueles na VPC. Para recuperar descrições de todos os endereços IP elásticos, omita um filtro do JSON dos parâmetros. Depois, chame o método `describeAddresses` do objeto de serviço do Amazon EC2.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create EC2 service object
var ec2 = new AWS.EC2({ apiVersion: "2016-11-15" });

var params = {
  Filters: [{ Name: "domain", Values: ["vpc"] }],
};

// Retrieve Elastic IP address descriptions
ec2.describeAddresses(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", JSON.stringify(data.Addresses));
  }
});
```

Para executar o exemplo, digite o seguinte na linha de comando.

```
node ec2_describeaddresses.js
```

Este código de exemplo pode ser encontrado aqui no GitHub.

Alocar e associar um endereço IP elástico a uma instância do Amazon EC2

Crie um módulo do Node.js com o nome de arquivo `ec2_allocateaddress.js`. Não se esqueça de configurar o SDK conforme mostrado anteriormente. Para acessar o Amazon EC2, crie um

objeto de serviço do `AWS.EC2`. Crie um objeto JSON para os parâmetros usados a fim de alocar um endereço IP elástico, que, neste caso, especifica que o `Domain` é uma VPC. Chame o método `allocateAddress` do objeto de serviço do Amazon EC2.

Se a chamada for bem-sucedida, o parâmetro `data` para a função de retorno de chamada terá uma propriedade `AllocationId` que identifica o endereço IP elástico alocado.

Crie um objeto JSON para os parâmetros usados na associação de um endereço IP elástico a uma instância do Amazon EC2, inclusive o `AllocationId` do endereço recém-alocado e o `InstanceId` da instância do Amazon EC2. Depois, chame o método `associateAddresses` do objeto de serviço do Amazon EC2.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create EC2 service object
var ec2 = new AWS.EC2({ apiVersion: "2016-11-15" });

var paramsAllocateAddress = {
  Domain: "vpc",
};

// Allocate the Elastic IP address
ec2.allocateAddress(paramsAllocateAddress, function (err, data) {
  if (err) {
    console.log("Address Not Allocated", err);
  } else {
    console.log("Address allocated:", data.AllocationId);
    var paramsAssociateAddress = {
      AllocationId: data.AllocationId,
      InstanceId: "INSTANCE_ID",
    };
    // Associate the new Elastic IP address with an EC2 instance
    ec2.associateAddress(paramsAssociateAddress, function (err, data) {
      if (err) {
        console.log("Address Not Associated", err);
      } else {
        console.log("Address associated:", data.AssociationId);
      }
    });
  }
```

```
});
```

Para executar o exemplo, digite o seguinte na linha de comando.

```
node ec2_allocateaddress.js
```

Este código de exemplo pode ser encontrado aqui no GitHub.

Como liberar um endereço IP elástico

Crie um módulo do Node.js com o nome de arquivo `ec2_releaseaddress.js`. Não se esqueça de configurar o SDK conforme mostrado anteriormente. Para acessar o Amazon EC2, crie um objeto de serviço do `AWS.EC2`. Crie um objeto JSON para os parâmetros usados na liberação de um endereço IP elástico, que, neste caso, especifica o `AllocationId` para o endereço IP elástico. Liberar um endereço IP elástico também o dissocia de qualquer instância do Amazon EC2. Chame o método `releaseAddress` do objeto de serviço do Amazon EC2.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create EC2 service object
var ec2 = new AWS.EC2({ apiVersion: "2016-11-15" });

var paramsReleaseAddress = {
  AllocationId: "ALLOCATION_ID",
};

// Disassociate the Elastic IP address from EC2 instance
ec2.releaseAddress(paramsReleaseAddress, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Address released");
  }
});
```

Para executar o exemplo, digite o seguinte na linha de comando.

```
node ec2_releaseaddress.js
```

Este código de exemplo pode ser encontrado aqui no GitHub.

Exemplos do AWS Elemental MediaConvert

O AWS Elemental MediaConvert é um serviço de transcodificação de vídeo baseado em arquivo com recursos de nível de transmissão. Você pode usá-lo para criar ativos para fornecer transmissão e vídeo sob demanda (VoD) na Internet. Para mais informações, consulte o Guia do usuário do AWS Elemental MediaConvert.

A API JavaScript do MediaConvert é exposta por meio da classe de cliente do `AWS.MediaConvert`. Para obter mais informações, consulte `Class: AWS.MediaConvert` na Referência da API.

Tópicos

- Obtendo seu endpoint específico da região para o MediaConvert
- Criar e gerenciar tarefas de transcodificação no MediaConvert
- Usando modelos de trabalho no MediaConvert

Obtendo seu endpoint específico da região para o MediaConvert

Este exemplo de código Node.js mostra:

- Como recuperar o endpoint específico da região do MediaConvert.

O cenário

Neste exemplo, você usa um módulo Node.js para chamar o MediaConvert e recuperar o endpoint específico da região. Você pode recuperar o URL do seu endpoint a partir do endpoint padrão de serviço e, dessa forma, o endpoint específico por região ainda não será necessário. O código usa o SDK para JavaScript para recuperar esse endpoint usando esse método da classe de cliente do MediaConvert:

- `describeEndpoints`

> **Important**
>
> O agente Node.js HTTP/HTTPS padrão cria uma nova conexão TCP para cada nova solicitação. Para evitar o custo de estabelecer uma nova conexão, o AWS SDK for JavaScript reutiliza conexões TCP existentes. Para ter mais informações, consulte Reutilizar conexões com Keep-alive no Node.js.

Tarefas de pré-requisito

Para configurar e executar este exemplo, primeiro conclua estas tarefas:

- Instale o Node.js. Para obter mais informações, consulte o website Node.js.
- Crie um arquivo de configurações compartilhado com as credenciais de usuário. Para obter mais informações sobre como fornecer um arquivo de credenciais compartilhadas, consulte Carregar credenciais em Node.js do arquivo de credenciais compartilhado.
- Crie um perfil do IAM que dê ao MediaConvert acesso aos arquivos de entrada e aos buckets do Amazon S3 onde os arquivos de saída são armazenados. Para obter detalhes, consulte Configurar permissões do IAM no Guia do usuário do AWS Elemental MediaConvert.

Obter o URL do endpoint

Crie um módulo do Node.js com o nome de arquivo `emc_getendpoint.js`. Não se esqueça de configurar o SDK conforme mostrado anteriormente.

Crie um objeto para passar os parâmetros de solicitação para o método `describeEndpoints` da classe de cliente `AWS.MediaConvert`. Para chamar o método `describeEndpoints`, crie uma promessa para invocar um objeto de serviço do MediaConvert passando os parâmetros. Lide com a resposta no retorno de chamada da promessa.

```
// Load the SDK for JavaScript.
const aws = require("aws-sdk");

// Set the AWS Region.
aws.config.update({ region: "us-west-2" });

// Create the client.
const mediaConvert = new aws.MediaConvert({ apiVersion: "2017-08-29" });
```

```
exports.handler = async (event, context) => {
  // Create empty request parameters
  const params = {
    MaxResults: 0,
  };

  try {
    const { Endpoints } = await mediaConvert
      .describeEndpoints(params)
      .promise();
    console.log("Your MediaConvert endpoint is ", Endpoints);
  } catch (err) {
    console.log("MediaConvert Error", err);
  }
};
```

Para executar o exemplo, digite o seguinte na linha de comando.

```
node emc_getendpoint.js
```

Este código de exemplo pode ser encontrado aqui no GitHub.

Criar e gerenciar tarefas de transcodificação no MediaConvert

Este exemplo de código Node.js mostra:

- Como especificar o endpoint específico por região a ser usado com o MediaConvert.
- Como criar tarefas de transcodificação no MediaConvert.
- Como cancelar uma tarefa de transcodificação.
- Como recuperar o JSON para concluir uma tarefa de transcodificação.
- Como recuperar uma matriz JSON para até 20 das tarefas criadas mais recentemente.

O cenário

Neste exemplo, você usa um módulo Node.js a fim de chamar o MediaConvert para criar e gerenciar tarefas de transcodificação. O código usa o SDK para JavaScript para fazer isso usando estes métodos da classe de cliente do MediaConvert:

- `createJob`
- `cancelJob`
- `getJob`
- `listJobs`

Tarefas de pré-requisito

Para configurar e executar este exemplo, primeiro conclua estas tarefas:

- Instale o Node.js. Para obter mais informações, consulte o website Node.js.
- Crie um arquivo de configurações compartilhado com as credenciais de usuário. Para obter mais informações sobre como fornecer um arquivo de credenciais compartilhadas, consulte Carregar credenciais em Node.js do arquivo de credenciais compartilhado.
- Crie e configure buckets do Amazon S3 que forneçam armazenamento para arquivos de entrada e de saída da tarefa. Para obter detalhes, consulte Criar armazenamento para arquivos no Guia do usuário do AWS Elemental MediaConvert.
- Faça upload do vídeo de entrada no bucket do Amazon S3 provisionado para armazenamento de entrada. Para obter uma lista dos codecs de vídeo de entrada e contêineres compatíveis, consulte Codecs e contêineres de entrada compatíveis no .
- Crie um perfil do IAM que dê ao MediaConvert acesso aos arquivos de entrada e aos buckets do Amazon S3 onde os arquivos de saída são armazenados. Para obter detalhes, consulte Configurar permissões do IAM no Guia do usuário do AWS Elemental MediaConvert.

Como configurar o SDK

Configure o criando um objeto de configuração global e definindo a região do código. Neste exemplo, a região é definida como `us-west-2`. Como o MediaConvert usa endpoints personalizados para cada conta, você também deve configurar a classe de cliente `AWS.MediaConvert` para usar o endpoint específico da conta. Para isso, defina o parâmetro `endpoint` em `AWS.config.mediaconvert`.

```
// Load the SDK for JavaScript
var AWS = require("aws-sdk");
// Set the Region
AWS.config.update({ region: "us-west-2" });
// Set the custom endpoint for your account
AWS.config.mediaconvert = { endpoint: "ACCOUNT_ENDPOINT" };
```

Definição de uma tarefa de transcodificação simples

Crie um módulo do Node.js com o nome de arquivo `emc_createjob.js`. Não se esqueça de configurar o SDK conforme mostrado anteriormente. Crie o JSON que define os parâmetros da tarefa de transcodificação.

Esses parâmetros são bem detalhados. Você pode usar o console do AWS Elemental MediaConvert para gerar os parâmetros de trabalho do JSON escolhendo as configurações de tarefa no console e depois selecionando Mostrar JSON de trabalho na parte inferior da seção Trabalho. Este exemplo mostra o JSON para uma tarefa simples.

```
var params = {
  Queue: "JOB_QUEUE_ARN",
  UserMetadata: {
    Customer: "Amazon",
  },
  Role: "IAM_ROLE_ARN",
  Settings: {
    OutputGroups: [
      {
        Name: "File Group",
        OutputGroupSettings: {
          Type: "FILE_GROUP_SETTINGS",
          FileGroupSettings: {
            Destination: "s3://OUTPUT_BUCKET_NAME/",
          },
        },
        Outputs: [
          {
            VideoDescription: {
              ScalingBehavior: "DEFAULT",
              TimecodeInsertion: "DISABLED",
              AntiAlias: "ENABLED",
              Sharpness: 50,
              CodecSettings: {
```

```
            Codec: "H_264",
            H264Settings: {
              InterlaceMode: "PROGRESSIVE",
              NumberReferenceFrames: 3,
              Syntax: "DEFAULT",
              Softness: 0,
              GopClosedCadence: 1,
              GopSize: 90,
              Slices: 1,
              GopBReference: "DISABLED",
              SlowPal: "DISABLED",
              SpatialAdaptiveQuantization: "ENABLED",
              TemporalAdaptiveQuantization: "ENABLED",
              FlickerAdaptiveQuantization: "DISABLED",
              EntropyEncoding: "CABAC",
              Bitrate: 5000000,
              FramerateControl: "SPECIFIED",
              RateControlMode: "CBR",
              CodecProfile: "MAIN",
              Telecine: "NONE",
              MinIInterval: 0,
              AdaptiveQuantization: "HIGH",
              CodecLevel: "AUTO",
              FieldEncoding: "PAFF",
              SceneChangeDetect: "ENABLED",
              QualityTuningLevel: "SINGLE_PASS",
              FramerateConversionAlgorithm: "DUPLICATE_DROP",
              UnregisteredSeiTimecode: "DISABLED",
              GopSizeUnits: "FRAMES",
              ParControl: "SPECIFIED",
              NumberBFramesBetweenReferenceFrames: 2,
              RepeatPps: "DISABLED",
              FramerateNumerator: 30,
              FramerateDenominator: 1,
              ParNumerator: 1,
              ParDenominator: 1,
            },
          },
          AfdSignaling: "NONE",
          DropFrameTimecode: "ENABLED",
          RespondToAfd: "NONE",
          ColorMetadata: "INSERT",
        },
        AudioDescriptions: [
```

```
            {
              AudioTypeControl: "FOLLOW_INPUT",
              CodecSettings: {
                Codec: "AAC",
                AacSettings: {
                  AudioDescriptionBroadcasterMix: "NORMAL",
                  RateControlMode: "CBR",
                  CodecProfile: "LC",
                  CodingMode: "CODING_MODE_2_0",
                  RawFormat: "NONE",
                  SampleRate: 48000,
                  Specification: "MPEG4",
                  Bitrate: 64000,
                },
              },
              LanguageCodeControl: "FOLLOW_INPUT",
              AudioSourceName: "Audio Selector 1",
            },
          ],
          ContainerSettings: {
            Container: "MP4",
            Mp4Settings: {
              CslgAtom: "INCLUDE",
              FreeSpaceBox: "EXCLUDE",
              MoovPlacement: "PROGRESSIVE_DOWNLOAD",
            },
          },
          NameModifier: "_1",
        },
      ],
    },
  ],
  AdAvailOffset: 0,
  Inputs: [
    {
      AudioSelectors: {
        "Audio Selector 1": {
          Offset: 0,
          DefaultSelection: "NOT_DEFAULT",
          ProgramSelection: 1,
          SelectorType: "TRACK",
          Tracks: [1],
        },
      },
```

```
        VideoSelector: {
          ColorSpace: "FOLLOW",
        },
        FilterEnable: "AUTO",
        PsiControl: "USE_PSI",
        FilterStrength: 0,
        DeblockFilter: "DISABLED",
        DenoiseFilter: "DISABLED",
        TimecodeSource: "EMBEDDED",
        FileInput: "s3://INPUT_BUCKET_AND_FILE_NAME",
      },
    ],
    TimecodeConfig: {
      Source: "EMBEDDED",
    },
  },
};
```

Criar uma tarefa de transcodificação

Depois de criar o JSON de parâmetros de tarefa, chame o método `createJob` criando uma promessa para invocar um objeto de serviço `AWS.MediaConvert` e passando os parâmetros. Depois, lide com a resposta no retorno de chamada da promessa. O ID da tarefa criado é retornado no `data` da resposta.

```
// Create a promise on a MediaConvert object
var endpointPromise = new AWS.MediaConvert({ apiVersion: "2017-08-29" })
  .createJob(params)
  .promise();

// Handle promise's fulfilled/rejected status
endpointPromise.then(
  function (data) {
    console.log("Job created! ", data);
  },
  function (err) {
    console.log("Error", err);
  }
);
```

Para executar o exemplo, digite o seguinte na linha de comando.

```
node emc_createjob.js
```

Este código de exemplo pode ser encontrado aqui no GitHub.

Cancelar uma tarefa de transcodificação

Crie um módulo do Node.js com o nome de arquivo `emc_canceljob.js`. Não se esqueça de configurar o SDK conforme mostrado anteriormente. Crie o JSON que inclua o ID da tarefa a ser cancelada. Depois, chame o método `cancelJob` criando uma promessa para invocar um objeto de serviço `AWS.MediaConvert` passando os parâmetros. Lide com a resposta no retorno de chamada da promessa.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the Region
AWS.config.update({ region: "us-west-2" });
// Set MediaConvert to customer endpoint
AWS.config.mediaconvert = { endpoint: "ACCOUNT_ENDPOINT" };

var params = {
  Id: "JOB_ID" /* required */,
};

// Create a promise on a MediaConvert object
var endpointPromise = new AWS.MediaConvert({ apiVersion: "2017-08-29" })
  .cancelJob(params)
  .promise();

// Handle promise's fulfilled/rejected status
endpointPromise.then(
  function (data) {
    console.log("Job  " + params.Id + " is canceled");
  },
  function (err) {
    console.log("Error", err);
  }
);
```

Para executar o exemplo, digite o seguinte na linha de comando.

```
node ec2_canceljob.js
```

Este código de exemplo pode ser encontrado aqui no GitHub.

Listagem de tarefas de transcodificação recentes

Crie um módulo do Node.js com o nome de arquivo `emc_listjobs.js`. Não se esqueça de configurar o SDK conforme mostrado anteriormente.

Crie o JSON de parâmetros, inclusive valores para especificar se você deseja classificar a lista em ordem `ASCENDING` ou `DESCENDING`, o ARN da fila de trabalhos a ser verificada e o status de tarefas a ser incluído. Depois, chame o método `listJobs` criando uma promessa para invocar um objeto de serviço `AWS.MediaConvert` passando os parâmetros. Lide com a resposta no retorno de chamada da promessa.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the Region
AWS.config.update({ region: "us-west-2" });
// Set the customer endpoint
AWS.config.mediaconvert = { endpoint: "ACCOUNT_ENDPOINT" };

var params = {
  MaxResults: 10,
  Order: "ASCENDING",
  Queue: "QUEUE_ARN",
  Status: "SUBMITTED",
};

// Create a promise on a MediaConvert object
var endpointPromise = new AWS.MediaConvert({ apiVersion: "2017-08-29" })
  .listJobs(params)
  .promise();

// Handle promise's fulfilled/rejected status
endpointPromise.then(
  function (data) {
    console.log("Jobs: ", data);
  },
  function (err) {
    console.log("Error", err);
  }
);
```

Para executar o exemplo, digite o seguinte na linha de comando.

```
node emc_listjobs.js
```

Este código de exemplo pode ser encontrado aqui no GitHub.

Usando modelos de trabalho no MediaConvert

Este exemplo de código Node.js mostra:

- Como criar modelos de trabalho do MediaConvert.
- Como usar um modelo de tarefa para criar uma tarefa de transcodificação.
- Como listar todos os modelos de tarefa.
- Como excluir modelos de tarefa.

O cenário

O JSON necessário para criar um trabalho de transcodificação no MediaConvert é detalhado, contendo um grande número de configurações. Simplifique muito a criação de tarefas salvando as configurações em boas condições em um modelo de tarefa que você possa usar para criar tarefas subsequentes. Neste exemplo, você usa um módulo Node.js a fim de chamar o MediaConvert para criar, usar e gerenciar modelos de trabalho. O código usa o SDK para JavaScript para fazer isso usando estes métodos da classe de cliente do MediaConvert:

- `createJobTemplate`
- `createJob`
- `deleteJobTemplate`
- `listJobTemplates`

Tarefas de pré-requisito

Para configurar e executar este exemplo, primeiro conclua estas tarefas:

- Instale o Node.js. Para obter mais informações, consulte o website Node.js.
- Crie um arquivo de configurações compartilhado com as credenciais de usuário. Para obter mais informações sobre como fornecer um arquivo de credenciais compartilhadas, consulte Carregar credenciais em Node.js do arquivo de credenciais compartilhado.
- Crie um perfil do IAM que dê ao MediaConvert acesso aos arquivos de entrada e aos buckets do Amazon S3 onde os arquivos de saída são armazenados. Para obter detalhes, consulte Configurar permissões do IAM no Guia do usuário do AWS Elemental MediaConvert.

Criar um modelo de tarefa

Crie um módulo do Node.js com o nome de arquivo `emc_create_jobtemplate.js`. Não se esqueça de configurar o SDK conforme mostrado anteriormente.

Especifique o JSON de parâmetros para a criação do modelo. Use a maioria dos parâmetros JSON de uma tarefa anterior bem-sucedida para especificar os valores `Settings` no modelo. Este exemplo usa as configurações de tarefa de Criar e gerenciar tarefas de transcodificação no MediaConvert.

Chame o método `createJobTemplate` criando uma promessa para invocar um objeto de serviço `AWS.MediaConvert` passando os parâmetros. Depois, lide com a resposta no retorno de chamada da promessa.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the Region
AWS.config.update({ region: "us-west-2" });
// Set the custom endpoint for your account
AWS.config.mediaconvert = { endpoint: "ACCOUNT_ENDPOINT" };

var params = {
  Category: "YouTube Jobs",
  Description: "Final production transcode",
  Name: "DemoTemplate",
  Queue: "JOB_QUEUE_ARN",
  Settings: {
    OutputGroups: [
      {
        Name: "File Group",
        OutputGroupSettings: {
          Type: "FILE_GROUP_SETTINGS",
```

```
          FileGroupSettings: {
            Destination: "s3://BUCKET_NAME/",
          },
        },
        Outputs: [
          {
            VideoDescription: {
              ScalingBehavior: "DEFAULT",
              TimecodeInsertion: "DISABLED",
              AntiAlias: "ENABLED",
              Sharpness: 50,
              CodecSettings: {
                Codec: "H_264",
                H264Settings: {
                  InterlaceMode: "PROGRESSIVE",
                  NumberReferenceFrames: 3,
                  Syntax: "DEFAULT",
                  Softness: 0,
                  GopClosedCadence: 1,
                  GopSize: 90,
                  Slices: 1,
                  GopBReference: "DISABLED",
                  SlowPal: "DISABLED",
                  SpatialAdaptiveQuantization: "ENABLED",
                  TemporalAdaptiveQuantization: "ENABLED",
                  FlickerAdaptiveQuantization: "DISABLED",
                  EntropyEncoding: "CABAC",
                  Bitrate: 5000000,
                  FramerateControl: "SPECIFIED",
                  RateControlMode: "CBR",
                  CodecProfile: "MAIN",
                  Telecine: "NONE",
                  MinIInterval: 0,
                  AdaptiveQuantization: "HIGH",
                  CodecLevel: "AUTO",
                  FieldEncoding: "PAFF",
                  SceneChangeDetect: "ENABLED",
                  QualityTuningLevel: "SINGLE_PASS",
                  FramerateConversionAlgorithm: "DUPLICATE_DROP",
                  UnregisteredSeiTimecode: "DISABLED",
                  GopSizeUnits: "FRAMES",
                  ParControl: "SPECIFIED",
                  NumberBFramesBetweenReferenceFrames: 2,
                  RepeatPps: "DISABLED",
```

```
                FramerateNumerator: 30,
                FramerateDenominator: 1,
                ParNumerator: 1,
                ParDenominator: 1,
              },
            },
            AfdSignaling: "NONE",
            DropFrameTimecode: "ENABLED",
            RespondToAfd: "NONE",
            ColorMetadata: "INSERT",
          },
          AudioDescriptions: [
            {
              AudioTypeControl: "FOLLOW_INPUT",
              CodecSettings: {
                Codec: "AAC",
                AacSettings: {
                  AudioDescriptionBroadcasterMix: "NORMAL",
                  RateControlMode: "CBR",
                  CodecProfile: "LC",
                  CodingMode: "CODING_MODE_2_0",
                  RawFormat: "NONE",
                  SampleRate: 48000,
                  Specification: "MPEG4",
                  Bitrate: 64000,
                },
              },
              LanguageCodeControl: "FOLLOW_INPUT",
              AudioSourceName: "Audio Selector 1",
            },
          ],
          ContainerSettings: {
            Container: "MP4",
            Mp4Settings: {
              CslgAtom: "INCLUDE",
              FreeSpaceBox: "EXCLUDE",
              MoovPlacement: "PROGRESSIVE_DOWNLOAD",
            },
          },
          NameModifier: "_1",
        },
      ],
    },
  ],
```

```
    AdAvailOffset: 0,
    Inputs: [
      {
        AudioSelectors: {
          "Audio Selector 1": {
            Offset: 0,
            DefaultSelection: "NOT_DEFAULT",
            ProgramSelection: 1,
            SelectorType: "TRACK",
            Tracks: [1],
          },
        },
        VideoSelector: {
          ColorSpace: "FOLLOW",
        },
        FilterEnable: "AUTO",
        PsiControl: "USE_PSI",
        FilterStrength: 0,
        DeblockFilter: "DISABLED",
        DenoiseFilter: "DISABLED",
        TimecodeSource: "EMBEDDED",
      },
    ],
    TimecodeConfig: {
      Source: "EMBEDDED",
    },
  },
};

// Create a promise on a MediaConvert object
var templatePromise = new AWS.MediaConvert({ apiVersion: "2017-08-29" })
  .createJobTemplate(params)
  .promise();

// Handle promise's fulfilled/rejected status
templatePromise.then(
  function (data) {
    console.log("Success!", data);
  },
  function (err) {
    console.log("Error", err);
  }
);
```

Para executar o exemplo, digite o seguinte na linha de comando.

```
node emc_create_jobtemplate.js
```

Este código de exemplo pode ser encontrado aqui no GitHub.

Criar uma tarefa de transcodificação com base em um modelo

Crie um módulo do Node.js com o nome de arquivo `emc_template_createjob.js`. Não se esqueça de configurar o SDK conforme mostrado anteriormente.

Crie o JSON de parâmetros da criação de tarefas, inclusive o nome do modelo de tarefa a ser usado, e o `Settings` para usar os específicos da tarefa que você está criando. Depois, chame o método `createJobs` criando uma promessa para invocar um objeto de serviço `AWS.MediaConvert` passando os parâmetros. Lide com a resposta no retorno de chamada da promessa.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the Region
AWS.config.update({ region: "us-west-2" });
// Set the custom endpoint for your account
AWS.config.mediaconvert = { endpoint: "ACCOUNT_ENDPOINT" };

var params = {
  Queue: "QUEUE_ARN",
  JobTemplate: "TEMPLATE_NAME",
  Role: "ROLE_ARN",
  Settings: {
    Inputs: [
      {
        AudioSelectors: {
          "Audio Selector 1": {
            Offset: 0,
            DefaultSelection: "NOT_DEFAULT",
            ProgramSelection: 1,
            SelectorType: "TRACK",
            Tracks: [1],
          },
        },
        VideoSelector: {
          ColorSpace: "FOLLOW",
        },
        FilterEnable: "AUTO",
```

```
        PsiControl: "USE_PSI",
        FilterStrength: 0,
        DeblockFilter: "DISABLED",
        DenoiseFilter: "DISABLED",
        TimecodeSource: "EMBEDDED",
        FileInput: "s3://BUCKET_NAME/FILE_NAME",
      },
    ],
  },
};

// Create a promise on a MediaConvert object
var templateJobPromise = new AWS.MediaConvert({ apiVersion: "2017-08-29" })
  .createJob(params)
  .promise();

// Handle promise's fulfilled/rejected status
templateJobPromise.then(
  function (data) {
    console.log("Success! ", data);
  },
  function (err) {
    console.log("Error", err);
  }
);
```

Para executar o exemplo, digite o seguinte na linha de comando.

```
node emc_template_createjob.js
```

Este código de exemplo pode ser encontrado aqui no GitHub.

Listar os modelos de tarefa

Crie um módulo do Node.js com o nome de arquivo `emc_listtemplates.js`. Não se esqueça de configurar o SDK conforme mostrado anteriormente.

Crie um objeto para passar os parâmetros de solicitação para o método `listTemplates` da classe de cliente `AWS.MediaConvert`. Inclua valores para determinar quais modelos listar (`NAME`, `CREATION_DATE`, `SYSTEM`), quantas listar e a ordem de classificação. Para chamar o método `listTemplates`, crie uma promessa para invocar um objeto de serviço do MediaConvert passando os parâmetros. Depois, lide com a resposta no retorno de chamada da promessa.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the Region
AWS.config.update({ region: "us-west-2" });
// Set the customer endpoint
AWS.config.mediaconvert = { endpoint: "ACCOUNT_ENDPOINT" };

var params = {
  ListBy: "NAME",
  MaxResults: 10,
  Order: "ASCENDING",
};

// Create a promise on a MediaConvert object
var listTemplatesPromise = new AWS.MediaConvert({ apiVersion: "2017-08-29" })
  .listJobTemplates(params)
  .promise();

// Handle promise's fulfilled/rejected status
listTemplatesPromise.then(
  function (data) {
    console.log("Success ", data);
  },
  function (err) {
    console.log("Error", err);
  }
);
```

Para executar o exemplo, digite o seguinte na linha de comando.

```
node emc_listtemplates.js
```

Este código de exemplo pode ser encontrado aqui no GitHub.

Excluir um modelo de tarefa

Crie um módulo do Node.js com o nome de arquivo `emc_deletetemplate.js`. Não se esqueça de configurar o SDK conforme mostrado anteriormente.

Crie um objeto para passar o nome do modelo de tarefa que você deseja excluir como parâmetro do método `deleteJobTemplate` da classe de cliente `AWS.MediaConvert`. Para chamar o método

`deleteJobTemplate`, crie uma promessa para invocar um objeto de serviço do MediaConvert passando os parâmetros. Lide com a resposta no retorno de chamada da promessa.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the Region
AWS.config.update({ region: "us-west-2" });
// Set the customer endpoint
AWS.config.mediaconvert = { endpoint: "ACCOUNT_ENDPOINT" };

var params = {
  Name: "TEMPLATE_NAME",
};

// Create a promise on a MediaConvert object
var deleteTemplatePromise = new AWS.MediaConvert({ apiVersion: "2017-08-29" })
  .deleteJobTemplate(params)
  .promise();

// Handle promise's fulfilled/rejected status
deleteTemplatePromise.then(
  function (data) {
    console.log("Success ", data);
  },
  function (err) {
    console.log("Error", err);
  }
);
```

Para executar o exemplo, digite o seguinte na linha de comando.

```
node emc_deletetemplate.js
```

Este código de exemplo pode ser encontrado aqui no GitHub.

Exemplos de Amazon S3 Glacier

Amazon S3 Glacier é um serviço de armazenamento em nuvem seguro para arquivamento de dados e backup de longo prazo. O serviço está otimizado para dados pouco acessados em que um tempo de recuperação de várias horas é indicado.

A API JavaScript do Amazon S3 Glacier é exposta por meio da classe de cliente `AWS.Glacier`. Para obter mais informações sobre como usar a classe de cliente do S3 Glacier, consulte `Class: AWS.Glacier` na referência da API.

Tópicos

- Criação de um S3 Glacier Vault
- Fazer upload de um arquivamento para o S3 Glacier
- Fazer um multipart upload para o S3 Glacier

Criação de um S3 Glacier Vault

Este exemplo de código Node.js mostra:

- Como criar um cofre usando o método `createVault` do objeto de serviço Amazon S3 Glacier.

Tarefas de pré-requisito

Para configurar e executar este exemplo, você deve primeiro concluir estas tarefas:

- Instale o Node.js. Para obter mais informações sobre como instalar Node.js, consulte o website de Node.js.

- Crie um arquivo de configurações compartilhado com as credenciais de usuário. Para obter mais informações sobre como fornecer um arquivo de credenciais compartilhadas, consulte Carregar credenciais em Node.js do arquivo de credenciais compartilhado.

Criar o cofre

```
// Load the SDK for JavaScript
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create a new service object
var glacier = new AWS.Glacier({ apiVersion: "2012-06-01" });
// Call Glacier to create the vault
glacier.createVault({ vaultName: "YOUR_VAULT_NAME" }, function (err) {
  if (!err) {
    console.log("Created vault!");
  }
});
```

Fazer upload de um arquivamento para o S3 Glacier

Este exemplo de código Node.js mostra:

- Como fazer upload de um arquivo para o Amazon S3 Glacier usando o método `uploadArchive` do objeto de serviço do S3 Glacier.

O exemplo a seguir faz upload de um único objeto `Buffer` como um arquivo inteiro usando o método `uploadArchive` do objeto de serviço do S3 Glacier.

O exemplo pressupõe que você já tenha criado um cofre chamado `YOUR_VAULT_NAME`. O SDK calcula automaticamente a soma de verificação do hash de árvore dos dados carregados. Entretanto, é possível substituí-la especificando o próprio parâmetro da soma de verificação:

Tarefas de pré-requisito

Para configurar e executar este exemplo, você deve primeiro concluir estas tarefas:

- Instale o Node.js. Para obter mais informações sobre como instalar Node.js, consulte o website de Node.js.
- Crie um arquivo de configurações compartilhado com as credenciais de usuário. Para obter mais informações sobre como fornecer um arquivo de credenciais compartilhadas, consulte Carregar credenciais em Node.js do arquivo de credenciais compartilhado.

Fazer upload do arquivo

```
// Load the SDK for JavaScript
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create a new service object and buffer
var glacier = new AWS.Glacier({ apiVersion: "2012-06-01" });
buffer = Buffer.alloc(2.5 * 1024 * 1024); // 2.5MB buffer

var params = { vaultName: "YOUR_VAULT_NAME", body: buffer };
// Call Glacier to upload the archive.
glacier.uploadArchive(params, function (err, data) {
  if (err) {
    console.log("Error uploading archive!", err);
  } else {
    console.log("Archive ID", data.archiveId);
  }
});
```

Fazer um multipart upload para o S3 Glacier

O exemplo a seguir cria um multipart upload de blocos de 1 megabyte de um objeto `Buffer` usando o método `initiateMultipartUpload` do objeto de serviço Amazon S3 Glacier.

O exemplo pressupõe que você já tenha criado um cofre chamado `YOUR_VAULT_NAME`. Um hash de árvore SHA-256 completo costuma ser calculado manualmente usando-se o método `computeChecksums`.

```
// Create a new service object and some supporting variables
var glacier = new AWS.Glacier({ apiVersion: "2012-06-01" }),
  vaultName = "YOUR_VAULT_NAME",
  buffer = new Buffer(2.5 * 1024 * 1024), // 2.5MB buffer
  partSize = 1024 * 1024, // 1MB chunks,
  numPartsLeft = Math.ceil(buffer.length / partSize),
  startTime = new Date(),
  params = { vaultName: vaultName, partSize: partSize.toString() };

// Compute the complete SHA-256 tree hash so we can pass it
// to completeMultipartUpload request at the end
var treeHash = glacier.computeChecksums(buffer).treeHash;

// Initiate the multipart upload
console.log("Initiating upload to", vaultName);
// Call Glacier to initiate the upload.
glacier.initiateMultipartUpload(params, function (mpErr, multipart) {
  if (mpErr) {
    console.log("Error!", mpErr.stack);
    return;
  }
  console.log("Got upload ID", multipart.uploadId);

  // Grab each partSize chunk and upload it as a part
  for (var i = 0; i < buffer.length; i += partSize) {
    var end = Math.min(i + partSize, buffer.length),
      partParams = {
        vaultName: vaultName,
        uploadId: multipart.uploadId,
        range: "bytes " + i + "-" + (end - 1) + "/*",
        body: buffer.slice(i, end),
      };

    // Send a single part
    console.log("Uploading part", i, "=", partParams.range);
    glacier.uploadMultipartPart(partParams, function (multiErr, mData) {
      if (multiErr) return;
      console.log("Completed part", this.request.params.range);
      if (--numPartsLeft > 0) return; // complete only when all parts uploaded

      var doneParams = {
        vaultName: vaultName,
        uploadId: multipart.uploadId,
```

```
        archiveSize: buffer.length.toString(),
        checksum: treeHash, // the computed tree hash
      };

      console.log("Completing upload...");
      glacier.completeMultipartUpload(doneParams, function (err, data) {
        if (err) {
          console.log("An error occurred while uploading the archive");
          console.log(err);
        } else {
          var delta = (new Date() - startTime) / 1000;
          console.log("Completed upload in", delta, "seconds");
          console.log("Archive ID:", data.archiveId);
          console.log("Checksum:  ", data.checksum);
        }
      });
    });
  }
});
```

Exemplos do IAM AWS

O AWS Identity and Access Management (IAM) é um serviço web que permite que os clientes da Amazon Web Services gerenciem usuários e permissões de usuário no AWS. O serviço é destinado a organizações com vários usuários ou sistemas na nuvem que usam os produtos da AWS. Com o IAM, é possível gerenciar usuários, credenciais de segurança como chaves de acesso e permissões que controlam quais recursos da AWS os usuários podem acessar.

A API JavaScript do IAM é exposta por meio da classe de cliente `AWS.IAM`. Para obter mais informações sobre como usar a classe de cliente do IAM, consulte `Class: AWS.IAM` na referência da API.

Tópicos

- Gerenciar usuários do IAM
- Trabalhar com políticas do IAM
- Gerenciar chaves de acesso do IAM
- Trabalhar com certificados de servidor do IAM
- Gerenciar aliases de conta do IAM

Gerenciar usuários do IAM

Este exemplo de código Node.js mostra:

- Como recuperar uma lista de usuários do IAM.
- Como criar e excluir usuários.
- Como atualizar um nome de usuário.

O cenário

Neste exemplo, é usada uma série de módulos do Node.js para criar e gerenciar usuários no IAM. Os módulos do Node.js usam o SDK para criar, excluir e atualizar os usuários usando estes métodos da classe de cliente do `AWS.IAM`:

- `createUser`
- `listUsers`
- `updateUser`
- `getUser`
- `deleteUser`

Para obter mais informações sobre usuários do IAM, consulte Usuários do IAM no Guia do usuário do IAM.

Tarefas de pré-requisito

Para configurar e executar este exemplo, você deve primeiro concluir estas tarefas:

- Instale o Node.js. Para obter mais informações sobre como instalar Node.js, consulte o website de Node.js.
- Crie um arquivo de configurações compartilhado com as credenciais de usuário. Para obter mais informações sobre como fornecer um arquivo de credenciais compartilhadas, consulte Carregar credenciais em Node.js do arquivo de credenciais compartilhado.

Criação de um usuário

Crie um módulo do Node.js com o nome de arquivo `iam_createuser.js`. Não se esqueça de configurar o SDK conforme mostrado anteriormente. Para acessar o IAM, crie um objeto de serviço do `AWS.IAM`. Crie um objeto JSON contendo os parâmetros necessários, compostos pelo nome do usuário que você deseja usar para o novo usuário como um parâmetro de linha de comando.

Chame o método `getUser` do objeto de serviço `AWS.IAM` para ver se o nome de usuário já existe. Se o nome de usuário ainda não existir, chame o método `createUser` para criá-lo. Se o nome já existe, escreva uma mensagem sobre esse efeito para o console.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

var params = {
  UserName: process.argv[2],
};

iam.getUser(params, function (err, data) {
  if (err && err.code === "NoSuchEntity") {
    iam.createUser(params, function (err, data) {
      if (err) {
        console.log("Error", err);
      } else {
        console.log("Success", data);
      }
```

```
    });
  } else {
    console.log(
      "User " + process.argv[2] + " already exists",
      data.User.UserId
    );
  }
});
```

Para executar o exemplo, digite o seguinte na linha de comando.

```
node iam_createuser.js USER_NAME
```

Este código de exemplo pode ser encontrado aqui no GitHub.

Listar os usuários na sua conta

Crie um módulo do Node.js com o nome de arquivo `iam_listusers.js`. Não se esqueça de configurar o SDK conforme mostrado anteriormente. Para acessar o IAM, crie um objeto de serviço do `AWS.IAM`. Crie um objeto JSON contendo os parâmetros necessários para listar seus usuários, limitando o número retornado ao definir o parâmetro `MaxItems` a 10. Chame o método `listUsers` do objeto de serviço do `AWS.IAM`. Grave o nome do primeiro usuário e a data de criação no console.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

var params = {
  MaxItems: 10,
};

iam.listUsers(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    var users = data.Users || [];
    users.forEach(function (user) {
      console.log("User " + user.UserName + " created", user.CreateDate);
```

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

Para executar o exemplo, digite o seguinte na linha de comando.

```
node iam_listusers.js
```

Este código de exemplo pode ser encontrado aqui no GitHub.

Atualizar o nome de um usuário

Crie um módulo do Node.js com o nome de arquivo `iam_updateuser.js`. Não se esqueça de configurar o SDK conforme mostrado anteriormente. Para acessar o IAM, crie um objeto de serviço do `AWS.IAM`. Crie um objeto JSON contendo os parâmetros necessários para listar seus usuários, especificando tanto o nome atual quanto o nome novo do usuário na forma de parâmetros de linha de comando. Chame o método `updateUser` do objeto de serviço do `AWS.IAM`.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

var params = {
  UserName: process.argv[2],
  NewUserName: process.argv[3],
};

iam.updateUser(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

Para executar o exemplo, digite o que está a seguir na linha de comando, especificando o nome atual do usuário seguido pelo nome novo do usuário.

```
node iam_updateuser.js ORIGINAL_USERNAME NEW_USERNAME
```

Este código de exemplo pode ser encontrado aqui no GitHub.

Excluir um usuário

Crie um módulo do Node.js com o nome de arquivo `iam_deleteuser.js`. Não se esqueça de configurar o SDK conforme mostrado anteriormente. Para acessar o IAM, crie um objeto de serviço do `AWS.IAM`. Crie um objeto JSON contendo os parâmetros necessários, compostos pelo nome do usuário que você deseja excluir como um parâmetro de linha de comando.

Chame o método `getUser` do objeto de serviço `AWS.IAM` para ver se o nome de usuário já existe. Se o nome de usuário não existir, escreva uma mensagem sobre esse efeito para o console. Se o usuário existir, chame o método `deleteUser` para excluí-lo.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

var params = {
  UserName: process.argv[2],
};

iam.getUser(params, function (err, data) {
  if (err && err.code === "NoSuchEntity") {
    console.log("User " + process.argv[2] + " does not exist.");
  } else {
    iam.deleteUser(params, function (err, data) {
      if (err) {
        console.log("Error", err);
      } else {
        console.log("Success", data);
      }
    });
  }
});
```

Para executar o exemplo, digite o seguinte na linha de comando.

```
node iam_deleteuser.js USER_NAME
```

Este código de exemplo pode ser encontrado aqui no GitHub.

Trabalhar com políticas do IAM

Este exemplo de código Node.js mostra:

- Como criar e excluir políticas do IAM.
- Como anexar e separar políticas do IAM das funções.

O cenário

Conceda permissões a um usuário ao criar uma política, que é um documento que lista as ações que um usuário pode executar e os recursos afetados por essas ações. Todas as ações ou recursos que não são explicitamente permitidos são negados por padrão. As políticas podem ser criadas e anexadas aos usuários, grupos de usuários, funções assumidas por usuários e recursos.

Neste exemplo, é usada uma série de módulos do Node.js para gerenciar políticas no IAM. Os módulos do Node.js usam o SDK para criar e excluir políticas, bem como para anexar e desanexar políticas da função, usando estes métodos da classe do cliente `AWS.IAM`:

- `createPolicy`
- `getPolicy`
- `listAttachedRolePolicies`
- `attachRolePolicy`
- `detachRolePolicy`

Para obter mais informações sobre as políticas do IAM, consulte Visão geral do gerenciamento de acesso: permissões e políticas no Guia do usuário do IAM.

Tarefas de pré-requisito

Para configurar e executar este exemplo, você deve primeiro concluir estas tarefas:

- Instale o Node.js. Para obter mais informações sobre como instalar Node.js, consulte o website de Node.js.
- Crie um arquivo de configurações compartilhado com as credenciais de usuário. Para obter mais informações sobre como fornecer um arquivo de credenciais compartilhadas, consulte Carregar credenciais em Node.js do arquivo de credenciais compartilhado.
- Crie um perfil do IAM ao qual você possa anexar políticas. Para obter mais informações sobre como criar perfis, consulte Criar perfils do IAM no Guia do usuário do IAM.

Criar uma política do IAM

Crie um módulo do Node.js com o nome de arquivo `iam_createpolicy.js`. Não se esqueça de configurar o SDK conforme mostrado anteriormente. Para acessar o IAM, crie um objeto de serviço do `AWS.IAM`. Crie dois objetos JSON, um contendo o documento de política que você deseja criar e o outro contendo os parâmetros necessários para criar a política, que inclui a política JSON e o nome que você deseja dar a ela. Execute o stringify do objeto JSON da política nos parâmetros. Chame o método `createPolicy` do objeto de serviço do `AWS.IAM`.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

var myManagedPolicy = {
  Version: "2012-10-17",
  Statement: [
    {
      Effect: "Allow",
      Action: "logs:CreateLogGroup",
      Resource: "RESOURCE_ARN",
    },
    {
      Effect: "Allow",
      Action: [
```

```
        "dynamodb:DeleteItem",
        "dynamodb:GetItem",
        "dynamodb:PutItem",
        "dynamodb:Scan",
        "dynamodb:UpdateItem",
      ],
      Resource: "RESOURCE_ARN",
    },
  ],
};

var params = {
  PolicyDocument: JSON.stringify(myManagedPolicy),
  PolicyName: "myDynamoDBPolicy",
};

iam.createPolicy(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

Para executar o exemplo, digite o seguinte na linha de comando.

```
node iam_createpolicy.js
```

Este código de exemplo pode ser encontrado aqui no GitHub.

Obter uma política do IAM

Crie um módulo do Node.js com o nome de arquivo `iam_getpolicy.js`. Não se esqueça de configurar o SDK conforme mostrado anteriormente. Para acessar o IAM, crie um objeto de serviço do `AWS.IAM`. Crie um objeto JSON contendo os parâmetros necessários para recuperar uma política, que é o ARN da política que você deseja obter. Chame o método `getPolicy` do objeto de serviço do `AWS.IAM`. Grave a descrição da política no console.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });
```

```
// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

var params = {
  PolicyArn: "arn:aws:iam::aws:policy/AWSLambdaExecute",
};

iam.getPolicy(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.Policy.Description);
  }
});
```

Para executar o exemplo, digite o seguinte na linha de comando.

```
node iam_getpolicy.js
```

Este código de exemplo pode ser encontrado aqui no GitHub.

Anexar uma política de função gerenciada

Crie um módulo do Node.js com o nome de arquivo `iam_attachrolepolicy.js`. Não se esqueça de configurar o SDK conforme mostrado anteriormente. Para acessar o IAM, crie um objeto de serviço do `AWS.IAM`. Crie um objeto JSON contendo os parâmetros necessários para obter uma lista de políticas do IAM gerenciadas anexadas a uma função, que consiste no nome da função. Forneça o nome da função como parâmetro de linha de comando. Chame o método `listAttachedRolePolicies` do objeto de serviço do `AWS.IAM`, que retorna um array de políticas gerenciadas para a função de retorno de chamada.

Verifique os membros do array para ver se a política que você deseja anexar à função já está anexada. Se a política não estiver anexada, chame o método `attachRolePolicy` para anexá-la.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });
```

```
var paramsRoleList = {
  RoleName: process.argv[2],
};

iam.listAttachedRolePolicies(paramsRoleList, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    var myRolePolicies = data.AttachedPolicies;
    myRolePolicies.forEach(function (val, index, array) {
      if (myRolePolicies[index].PolicyName === "AmazonDynamoDBFullAccess") {
        console.log(
          "AmazonDynamoDBFullAccess is already attached to this role."
        );
        process.exit();
      }
    });
    var params = {
      PolicyArn: "arn:aws:iam::aws:policy/AmazonDynamoDBFullAccess",
      RoleName: process.argv[2],
    };
    iam.attachRolePolicy(params, function (err, data) {
      if (err) {
        console.log("Unable to attach policy to role", err);
      } else {
        console.log("Role attached successfully");
      }
    });
  }
});
```

Para executar o exemplo, digite o seguinte na linha de comando.

```
node iam_attachrolepolicy.js IAM_ROLE_NAME
```

Desanexar uma política de função gerenciada

Crie um módulo do Node.js com o nome de arquivo `iam_detachrolepolicy.js`. Não se esqueça de configurar o SDK conforme mostrado anteriormente. Para acessar o IAM, crie um objeto de serviço do `AWS.IAM`. Crie um objeto JSON contendo os parâmetros necessários para obter uma lista de políticas do IAM gerenciadas anexadas a uma função, que consiste no nome

da função. Forneça o nome da função como parâmetro de linha de comando. Chame o método `listAttachedRolePolicies` do objeto de serviço do `AWS.IAM`, que retorna um array de políticas gerenciadas na função de retorno de chamada.

Verifique se os membros do array para ver se a política que você deseja desanexar da função está anexada. Se a política estiver anexada, chame o método `detachRolePolicy` para desanexá-la.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

var paramsRoleList = {
  RoleName: process.argv[2],
};

iam.listAttachedRolePolicies(paramsRoleList, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    var myRolePolicies = data.AttachedPolicies;
    myRolePolicies.forEach(function (val, index, array) {
      if (myRolePolicies[index].PolicyName === "AmazonDynamoDBFullAccess") {
        var params = {
          PolicyArn: "arn:aws:iam::aws:policy/AmazonDynamoDBFullAccess",
          RoleName: process.argv[2],
        };
        iam.detachRolePolicy(params, function (err, data) {
          if (err) {
            console.log("Unable to detach policy from role", err);
          } else {
            console.log("Policy detached from role successfully");
            process.exit();
          }
        });
      }
    });
  }
});
```

Para executar o exemplo, digite o seguinte na linha de comando.

```
node iam_detachrolepolicy.js IAM_ROLE_NAME
```

Gerenciar chaves de acesso do IAM

Este exemplo de código Node.js mostra:

- Como gerenciar as chaves de acesso dos seus usuários.

O cenário

Os usuários precisam de suas próprias chaves de acesso para fazer chamadas programáticas à AWS no SDK para JavaScript. Para atender a essa necessidade, você pode criar, modificar, exibir ou mudar chaves de acesso (IDs de chave de acesso e chaves de acesso secretas) para os usuários do IAM. Por padrão, quando você cria uma chave de acesso, o status dela é `Active`, o que significa que o usuário pode usar a chave de acesso para chamadas de API.

Neste exemplo, é usada uma série de módulos do Node.js para gerenciar chaves de acesso no IAM. Os módulos do Node.js usam o SDK para JavaScript para gerenciar as chaves de acesso do IAM usando estes métodos da classe de cliente `AWS.IAM`:

- createAccessKey
- listAccessKeys
- getAccessKeyLastUsed
- updateAccessKey
- deleteAccessKey

Para obter mais informações sobre as chaves de acesso, consulte Chaves de acesso no Manual do usuário do IAM.

Tarefas de pré-requisito

Para configurar e executar este exemplo, você deve primeiro concluir estas tarefas:

- Instale o Node.js. Para obter mais informações sobre como instalar Node.js, consulte o website de Node.js.
- Crie um arquivo de configurações compartilhado com as credenciais de usuário. Para obter mais informações sobre como fornecer um arquivo de credenciais compartilhadas, consulte Carregar credenciais em Node.js do arquivo de credenciais compartilhado.

Criar chaves de acesso para um usuário

Crie um módulo do Node.js com o nome de arquivo `iam_createaccesskeys.js`. Não se esqueça de configurar o SDK conforme mostrado anteriormente. Para acessar o IAM, crie um objeto de serviço do `AWS.IAM`. Crie um objeto JSON contendo os parâmetros necessários para criar novas chaves de acesso, que incluem o nome do usuário do IAM. Chame o método `createAccessKey` do objeto de serviço do `AWS.IAM`.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

iam.createAccessKey({ UserName: "IAM_USER_NAME" }, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.AccessKey);
  }
});
```

Para executar o exemplo, digite o seguinte na linha de comando. Redirecione os dados retornados para um arquivo de texto para não perder a chave secreta, que só pode ser fornecida uma vez.

```
node iam_createaccesskeys.js > newuserkeys.txt
```

Este código de exemplo pode ser encontrado aqui no GitHub.

Listar as chaves de acesso de um usuário

Crie um módulo do Node.js com o nome de arquivo `iam_listaccesskeys.js`. Não se esqueça de configurar o SDK conforme mostrado anteriormente. Para acessar o IAM, crie um objeto de serviço do `AWS.IAM`. Crie um objeto JSON contendo os parâmetros necessários para recuperar as chaves de acesso do usuário, que incluem o nome do usuário do IAM e, opcionalmente, o número máximo de pares de chave de acesso que você deseja listar. Chame o método `listAccessKeys` do objeto de serviço do `AWS.IAM`.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

var params = {
  MaxItems: 5,
  UserName: "IAM_USER_NAME",
};

iam.listAccessKeys(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

Para executar o exemplo, digite o seguinte na linha de comando.

```
node iam_listaccesskeys.js
```

Este código de exemplo pode ser encontrado aqui no GitHub.

Fazer o último uso das chaves de acesso

Crie um módulo do Node.js com o nome de arquivo `iam_accesskeylastused.js`. Não se esqueça de configurar o SDK conforme mostrado anteriormente. Para acessar o IAM, crie um objeto de serviço do `AWS.IAM`. Crie um objeto JSON contendo os parâmetros necessários para criar novas

chaves de acesso, que é o ID de chave de acesso para o qual você deseja que a informação de último uso. Chame o método `getAccessKeyLastUsed` do objeto de serviço do `AWS.IAM`.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

iam.getAccessKeyLastUsed(
  { AccessKeyId: "ACCESS_KEY_ID" },
  function (err, data) {
    if (err) {
      console.log("Error", err);
    } else {
      console.log("Success", data.AccessKeyLastUsed);
    }
  }
);
```

Para executar o exemplo, digite o seguinte na linha de comando.

```
node iam_accesskeylastused.js
```

Este código de exemplo pode ser encontrado aqui no GitHub.

Atualizar o status de uma chave de acesso

Crie um módulo do Node.js com o nome de arquivo `iam_updateaccesskey.js`. Não se esqueça de configurar o SDK conforme mostrado anteriormente. Para acessar o IAM, crie um objeto de serviço do `AWS.IAM`. Crie um objeto JSON contendo os parâmetros necessários para atualizar o status de chaves de acesso, que inclui o ID da chave de acesso e o status atualizado. O status pode ser `Active` ou `Inactive`. Chame o método `updateAccessKey` do objeto de serviço do `AWS.IAM`.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
```

```
AWS.config.update({ region: "REGION" });

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

var params = {
  AccessKeyId: "ACCESS_KEY_ID",
  Status: "Active",
  UserName: "USER_NAME",
};

iam.updateAccessKey(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

Para executar o exemplo, digite o seguinte na linha de comando.

```
node iam_updateaccesskey.js
```

Este código de exemplo pode ser encontrado aqui no GitHub.

Excluir chaves de acesso

Crie um módulo do Node.js com o nome de arquivo `iam_deleteaccesskey.js`. Não se esqueça de configurar o SDK conforme mostrado anteriormente. Para acessar o IAM, crie um objeto de serviço do `AWS.IAM`. Crie um objeto JSON contendo os parâmetros necessários para excluir chaves de acesso, que inclui o ID da chave de acesso e o nome do usuário. Chame o método `deleteAccessKey` do objeto de serviço do `AWS.IAM`.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

var params = {
```

```
  AccessKeyId: "ACCESS_KEY_ID",
  UserName: "USER_NAME",
};

iam.deleteAccessKey(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

Para executar o exemplo, digite o seguinte na linha de comando.

```
node iam_deleteaccesskey.js
```

Este código de exemplo pode ser encontrado aqui no GitHub.

Trabalhar com certificados de servidor do IAM

Este exemplo de código Node.js mostra:

- Como executar tarefas básicas no gerenciamento de certificados do servidor para conexões HTTPS.

O cenário

Para habilitar conexões HTTPS para o seu site ou aplicativo na AWS você precisa de um certificado de servidor SSL/TLS. Para usar um certificado que você obteve de um provedor externo com seu site ou aplicativo na AWS, é necessário fazer upload desse certificado no IAM ou importá-lo para o AWS Certificate Manager.

Neste exemplo, é usada uma série de módulos do Node.js para lidar com certificados do servidor no IAM. Os módulos do Node.js usam o SDK para JavaScript para gerenciar certificados do servidor usando estes métodos da classe de cliente do `AWS.IAM`:

- listServerCertificates
- getServerCertificate
- updateServerCertificate
- deleteServerCertificate

Para obter mais informações sobre certificados de servidor, consulte Trabalhar com certificados de servidor no Guia do usuário do IAM.

Tarefas de pré-requisito

Para configurar e executar este exemplo, você deve primeiro concluir estas tarefas:

- Instale o Node.js. Para obter mais informações sobre como instalar Node.js, consulte o website de Node.js.
- Crie um arquivo de configurações compartilhado com as credenciais de usuário. Para obter mais informações sobre como fornecer um arquivo de credenciais compartilhadas, consulte Carregar credenciais em Node.js do arquivo de credenciais compartilhado.

Listar seus certificados do servidor

Crie um módulo do Node.js com o nome de arquivo `iam_listservercerts.js`. Não se esqueça de configurar o SDK conforme mostrado anteriormente. Para acessar o IAM, crie um objeto de serviço do `AWS.IAM`. Chame o método `listServerCertificates` do objeto de serviço do `AWS.IAM`.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

iam.listServerCertificates({}, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
```

```
});
```

Para executar o exemplo, digite o seguinte na linha de comando.

```
node iam_listservercerts.js
```

Este código de exemplo pode ser encontrado aqui no GitHub.

Obter um certificado de servidor

Crie um módulo do Node.js com o nome de arquivo `iam_getservercert.js`. Não se esqueça de configurar o SDK conforme mostrado anteriormente. Para acessar o IAM, crie um objeto de serviço do `AWS.IAM`. Crie um objeto JSON contendo os parâmetros necessários para obter um certificado, que consiste no nome do certificado do servidor que você deseja. Chame o método `getServerCertificates` do objeto de serviço do `AWS.IAM`.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

iam.getServerCertificate(
  { ServerCertificateName: "CERTIFICATE_NAME" },
  function (err, data) {
    if (err) {
      console.log("Error", err);
    } else {
      console.log("Success", data);
    }
  }
);
```

Para executar o exemplo, digite o seguinte na linha de comando.

```
node iam_getservercert.js
```

Este código de exemplo pode ser encontrado aqui no GitHub.

Atualizar um certificado de servidor

Crie um módulo do Node.js com o nome de arquivo `iam_updateservercert.js`. Não se esqueça de configurar o SDK conforme mostrado anteriormente. Para acessar o IAM, crie um objeto de serviço do `AWS.IAM`. Crie um objeto JSON contendo os parâmetros necessários para atualizar um certificado, que consiste no nome do certificado de servidor existente, bem como o nome do novo certificado. Chame o método `updateServerCertificate` do objeto de serviço do `AWS.IAM`.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

var params = {
  ServerCertificateName: "CERTIFICATE_NAME",
  NewServerCertificateName: "NEW_CERTIFICATE_NAME",
};

iam.updateServerCertificate(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

Para executar o exemplo, digite o seguinte na linha de comando.

```
node iam_updateservercert.js
```

Este código de exemplo pode ser encontrado aqui no GitHub.

Excluir um certificado de servidor

Crie um módulo do Node.js com o nome de arquivo `iam_deleteservercert.js`. Não se esqueça de configurar o SDK conforme mostrado anteriormente. Para acessar o IAM, crie um objeto de serviço do `AWS.IAM`. Crie um objeto JSON contendo os parâmetros necessários para excluir um

certificado do servidor, que consiste no nome do certificado que você deseja excluir. Chame o método `deleteServerCertificates` do objeto de serviço do `AWS.IAM`.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

iam.deleteServerCertificate(
  { ServerCertificateName: "CERTIFICATE_NAME" },
  function (err, data) {
    if (err) {
      console.log("Error", err);
    } else {
      console.log("Success", data);
    }
  }
);
```

Para executar o exemplo, digite o seguinte na linha de comando.

```
node iam_deleteservercert.js
```

Este código de exemplo pode ser encontrado aqui no GitHub.

Gerenciar aliases de conta do IAM

Este exemplo de código Node.js mostra:

- Como gerenciar aliases para o ID da sua conta da AWS.

O cenário

Se deseja que o URL para sua página de login contenha o nome da sua empresa (ou outro identificador amigável) em vez do ID da sua conta da AWS, você pode criar um alias para o ID de sua conta da AWS. Se você criar um alias de conta do AWS, a URL de sua página de login será alterada para incorporar esse alias.

Neste exemplo, é usada uma série de módulos do Node.js para criar e gerenciar aliases da conta do IAM. Os módulos do Node.js usam o SDK para JavaScript para gerenciar aliases usando estes métodos da classe de cliente `AWS.IAM`:

- createAccountAlias
- listAccountAliases
- deleteAccountAlias

Para obter mais informações sobre alias de contas do IAM, consulte ID da sua conta da AWS e seu alias no Guia do usuário do IAM.

Tarefas de pré-requisito

Para configurar e executar este exemplo, você deve primeiro concluir estas tarefas:

- Instale o Node.js. Para obter mais informações sobre como instalar Node.js, consulte o website de Node.js.
- Crie um arquivo de configurações compartilhado com as credenciais de usuário. Para obter mais informações sobre como fornecer um arquivo de credenciais compartilhadas, consulte Carregar credenciais em Node.js do arquivo de credenciais compartilhado.

Criar um alias da conta

Crie um módulo do Node.js com o nome de arquivo `iam_createaccountalias.js`. Não se esqueça de configurar o SDK conforme mostrado anteriormente. Para acessar o IAM, crie um objeto de serviço do `AWS.IAM`. Crie um objeto JSON contendo os parâmetros necessários para criar um alias de conta, que inclui o alias que você deseja criar. Chame o método `createAccountAlias` do objeto de serviço do `AWS.IAM`.

```
// Load the AWS SDK for Node.js
```

```
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

iam.createAccountAlias({ AccountAlias: process.argv[2] }, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

Para executar o exemplo, digite o seguinte na linha de comando.

```
node iam_createaccountalias.js ALIAS
```

Este código de exemplo pode ser encontrado aqui no GitHub.

Listar aliases de conta

Crie um módulo do Node.js com o nome de arquivo `iam_listaccountaliases.js`. Não se esqueça de configurar o SDK conforme mostrado anteriormente. Para acessar o IAM, crie um objeto de serviço do `AWS.IAM`. Crie um objeto JSON contendo os parâmetros necessários para listar os aliases de conta, que inclui o número máximo de itens a serem retornados. Chame o método `listAccountAliases` do objeto de serviço do `AWS.IAM`.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

iam.listAccountAliases({ MaxItems: 10 }, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
```

```
    console.log("Success", data);
  }
});
```

Para executar o exemplo, digite o seguinte na linha de comando.

```
node iam_listaccountaliases.js
```

Este código de exemplo pode ser encontrado aqui no GitHub.

Excluir um alias de conta

Crie um módulo do Node.js com o nome de arquivo `iam_deleteaccountalias.js`. Não se esqueça de configurar o SDK conforme mostrado anteriormente. Para acessar o IAM, crie um objeto de serviço do `AWS.IAM`. Crie um objeto JSON contendo os parâmetros necessários para excluir um alias de conta, que inclui o alias que você deseja excluir. Chame o método `deleteAccountAlias` do objeto de serviço do `AWS.IAM`.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the IAM service object
var iam = new AWS.IAM({ apiVersion: "2010-05-08" });

iam.deleteAccountAlias({ AccountAlias: process.argv[2] }, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

Para executar o exemplo, digite o seguinte na linha de comando.

```
node iam_deleteaccountalias.js ALIAS
```

Este código de exemplo pode ser encontrado aqui no GitHub.

Exemplos do Amazon Kinesis

O Amazon Kinesis é uma plataforma de dados de streaming na AWS, que fornece serviços excelentes para facilitar o carregamento e a análise de pilha, além de capacitá-lo a criar aplicativos de pilha personalizados para necessidades específicas.

A API JavaScript do Kinesis é exposta por meio da classe de cliente `AWS.Kinesis`. Para obter mais informações sobre como usar a classe de cliente Kinesis, consulte `Class: AWS.Kinesis` na referência da API.

Tópicos

- Captura do progresso de rolagem da página da web com Amazon Kinesis

Captura do progresso de rolagem da página da web com Amazon Kinesis

Este exemplo de script do navegador mostra:

- Como capturar o progresso de rolagem em uma página da web com o Amazon Kinesis, como exemplo de métricas de uso da página de streaming para análise posterior.

O cenário

Neste exemplo, uma página de HTML simples simula o conteúdo de uma página de blog. À medida que o leitor roda o post no blog simulado, o script do navegador usa o para registrar a distância de

rolagem vertical da página e envia esses dados para o Kinesis usando o método `putRecords` da classe de cliente Kinesis. Os dados de streaming capturados pelo Amazon Kinesis Data Streams podem ser processados por instâncias do Amazon EC2 e armazenados em qualquer um dos vários datastores, incluindo Amazon DynamoDB e Amazon Redshift.

Tarefas de pré-requisito

Para configurar e executar este exemplo, você deve primeiro concluir estas tarefas:

- Crie uma transmissão do Kinesis. Você precisa incluir o ARN do recurso do streaming no script do navegador. Para obter mais informações sobre como criar o Amazon Kinesis Data Streams, consulte Gerenciando Kinesis Streams no Guia do desenvolvedor do Amazon Kinesis Data Streams.
- Crie um banco de identidades do Amazon Cognito com acesso habilitado para identidades não autenticadas. Você precisa incluir o ID do grupo de identidades no código para obter credenciais para o script do navegador. Para obter mais informações, consulte Grupos de identidade do Amazon Cognito no Guia do desenvolvedor do Amazon Cognito.
- Crie um perfil de IAM cuja política conceda permissão para enviar dados a um streaming do Kinesis. Para obter mais informações sobre como criar um perfil do IAM, consulte Criação de um perfil para delegar permissões a um serviço da AWS no Guia do usuário do IAM.

Use a política de função a seguir ao criar a função do IAM.

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Effect": "Allow",
         "Action": [
            "mobileanalytics:PutEvents",
            "cognito-sync:*"
         ],
         "Resource": [
            "*"
         ]
      },
      {
         "Effect": "Allow",
         "Action": [
            "kinesis:Put*"
```

```
         ],
         "Resource": [
            "STREAM_RESOURCE_ARN"
         ]
      }
   ]
}
```

A página do blog

O HTML da página do blog é composto principalmente por uma série de pontos contidos dentro de um elemento `<div>`. A altura rolável desse `<div>` é usada para ajudar a calcular até onde o leitor rolou o conteúdo à medida que lê. O HTML também contém um par de elementos `<script>`. Um desses elementos adiciona o SDK para JavaScript à página e o outro adiciona o script do navegador, que captura o progresso da rolagem na página e o reporta ao Kinesis.

```
<!DOCTYPE html>
<html>
    <head>
        <title>AWS SDK for JavaScript - Amazon Kinesis Application</title>
    </head>
    <body>
        <div id="BlogContent" style="width: 60%; height: 800px; overflow: auto;margin:
 auto; text-align: center;">
            <div>
                <p>
                Lorem ipsum dolor sit amet, consectetur adipiscing elit. Vestibulum
 vitae nulla eget nisl bibendum feugiat. Fusce rhoncus felis at ultricies luctus.
 Vivamus fermentum cursus sem at interdum. Proin vel lobortis nulla. Aenean rutrum
 odio in tellus semper rhoncus. Nam eu felis ac augue dapibus laoreet vel in erat.
 Vivamus vitae mollis turpis. Integer sagittis dictum odio. Duis nec sapien diam.
 In imperdiet sem nec ante laoreet, vehicula facilisis sem placerat. Duis ut metus
 egestas, ullamcorper neque et, accumsan quam. Class aptent taciti sociosqu ad litora
 torquent per conubia nostra, per inceptos himenaeos.
                </p>
                <!-- Additional paragraphs in the blog page appear here -->
            </div>
        </div>
        <script src="https://sdk.amazonaws.com/js/aws-sdk-2.283.1.min.js"></script>
        <script src="kinesis-example.js"></script>
    </body>
</html>
```

Como configurar o SDK

Obtenha as credenciais necessárias para configurar o SDK chamando o método `CognitoIdentityCredentials`, fornecendo o ID do banco de identidades do Amazon Cognito. Após o sucesso, crie o objeto de serviço do Kinesis na função de retorno de chamada.

O snippet de código a seguir mostra essa etapa. (Consulte Captura do código do progresso de rolagem da página da web para ver o exemplo completo.)

```
// Configure Credentials to use Cognito
AWS.config.credentials = new AWS.CognitoIdentityCredentials({
  IdentityPoolId: "IDENTITY_POOL_ID",
});

AWS.config.region = "REGION";
// We're going to partition Amazon Kinesis records based on an identity.
// We need to get credentials first, then attach our event listeners.
AWS.config.credentials.get(function (err) {
  // attach event listener
  if (err) {
    alert("Error retrieving credentials.");
    console.error(err);
    return;
  }
  // create Amazon Kinesis service object
  var kinesis = new AWS.Kinesis({
    apiVersion: "2013-12-02",
  });
```

Criar registros de rolagem

O progresso de rolagem é calculado usando as propriedades `scrollHeight` e `scrollTop` do `<div>` que contém o conteúdo do post do blog. Cada registro de rolagem é criado em uma função do listener do evento para o evento `scroll` e adicionado a um array de registros para envio periódico ao Kinesis.

O snippet de código a seguir mostra essa etapa. (Consulte Captura do código do progresso de rolagem da página da web para ver o exemplo completo.)

```
  // Get the ID of the Web page element.
  var blogContent = document.getElementById("BlogContent");
```

```
// Get Scrollable height
var scrollableHeight = blogContent.clientHeight;

var recordData = [];
var TID = null;
blogContent.addEventListener("scroll", function (event) {
  clearTimeout(TID);
  // Prevent creating a record while a user is actively scrolling
  TID = setTimeout(function () {
    // calculate percentage
    var scrollableElement = event.target;
    var scrollHeight = scrollableElement.scrollHeight;
    var scrollTop = scrollableElement.scrollTop;

    var scrollTopPercentage = Math.round((scrollTop / scrollHeight) * 100);
    var scrollBottomPercentage = Math.round(
      ((scrollTop + scrollableHeight) / scrollHeight) * 100
    );

    // Create the Amazon Kinesis record
    var record = {
      Data: JSON.stringify({
        blog: window.location.href,
        scrollTopPercentage: scrollTopPercentage,
        scrollBottomPercentage: scrollBottomPercentage,
        time: new Date(),
      }),
      PartitionKey: "partition-" + AWS.config.credentials.identityId,
    };
    recordData.push(record);
  }, 100);
});
```

Envio de registros para o Kinesis

Uma vez a cada segundo, se houver registros na matriz, esses registros pendentes são enviados ao Kinesis.

O snippet de código a seguir mostra essa etapa. (Consulte Captura do código do progresso de rolagem da página da web para ver o exemplo completo.)

```
// upload data to Amazon Kinesis every second if data exists
setInterval(function () {
```

```
    if (!recordData.length) {
      return;
    }
    // upload data to Amazon Kinesis
    kinesis.putRecords(
      {
        Records: recordData,
        StreamName: "NAME_OF_STREAM",
      },
      function (err, data) {
        if (err) {
          console.error(err);
        }
      }
    );
    // clear record data
    recordData = [];
  }, 1000);
});
```

Captura do código do progresso de rolagem da página da web

Este é o código de script do navegador para o exemplo de progresso de rolagem da página da web que captura o Kinesis.

```
// Configure Credentials to use Cognito
AWS.config.credentials = new AWS.CognitoIdentityCredentials({
  IdentityPoolId: "IDENTITY_POOL_ID",
});

AWS.config.region = "REGION";
// We're going to partition Amazon Kinesis records based on an identity.
// We need to get credentials first, then attach our event listeners.
AWS.config.credentials.get(function (err) {
  // attach event listener
  if (err) {
    alert("Error retrieving credentials.");
    console.error(err);
    return;
  }
  // create Amazon Kinesis service object
  var kinesis = new AWS.Kinesis({
    apiVersion: "2013-12-02",
```

```
});

// Get the ID of the Web page element.
var blogContent = document.getElementById("BlogContent");

// Get Scrollable height
var scrollableHeight = blogContent.clientHeight;

var recordData = [];
var TID = null;
blogContent.addEventListener("scroll", function (event) {
  clearTimeout(TID);
  // Prevent creating a record while a user is actively scrolling
  TID = setTimeout(function () {
    // calculate percentage
    var scrollableElement = event.target;
    var scrollHeight = scrollableElement.scrollHeight;
    var scrollTop = scrollableElement.scrollTop;

    var scrollTopPercentage = Math.round((scrollTop / scrollHeight) * 100);
    var scrollBottomPercentage = Math.round(
      ((scrollTop + scrollableHeight) / scrollHeight) * 100
    );

    // Create the Amazon Kinesis record
    var record = {
      Data: JSON.stringify({
        blog: window.location.href,
        scrollTopPercentage: scrollTopPercentage,
        scrollBottomPercentage: scrollBottomPercentage,
        time: new Date(),
      }),
      PartitionKey: "partition-" + AWS.config.credentials.identityId,
    };
    recordData.push(record);
  }, 100);
});

// upload data to Amazon Kinesis every second if data exists
setInterval(function () {
  if (!recordData.length) {
    return;
  }
  // upload data to Amazon Kinesis
```

```
    kinesis.putRecords(
      {
        Records: recordData,
        StreamName: "NAME_OF_STREAM",
      },
      function (err, data) {
        if (err) {
          console.error(err);
        }
      }
    );
    // clear record data
    recordData = [];
  }, 1000);
});
```

Exemplos do Amazon S3

O Amazon Simple Storage Service (Amazon S3) oferece um armazenamento na nuvem altamente seguro. O Amazon S3 fornece armazenamento de objetos fácil de usar com uma interface de web service para armazenar e recuperar qualquer quantidade de dados de qualquer lugar da web.

A API JavaScript do Amazon S3 é exposta por meio da classe de cliente `AWS.S3`. Para obter mais informações sobre como usar a classe de cliente do Amazon S3, consulte `Class: AWS.S3` na referência da API.

Tópicos

- Exemplos de navegador Amazon S3
- Exemplos de Node.js do Amazon S3

Exemplos de navegador Amazon S3

Os tópicos a seguir mostram dois exemplos de como o pode ser usado no navegador para interagir com buckets do Amazon S3.

- O primeiro mostra um cenário simples em que as fotos existentes em um bucket do Amazon S3 podem ser visualizadas por qualquer usuário (não autenticado).
- O segundo mostra um cenário mais complexo em que os usuários têm permissão para realizar operações em fotos no bucket, como upload, exclusão etc.

Tópicos

- Como visualizar fotos em um bucket do Amazon S3 em um navegador
- Fazer upload de fotos para o Amazon S3 partir de um navegador

Como visualizar fotos em um bucket do Amazon S3 em um navegador

Este exemplo de código de script do navegador mostra:

- Como criar um álbum de fotos em um bucket do Amazon Simple Storage Service (Amazon S3) e permitir que usuários não autenticados visualizem as fotos.

O cenário

Neste exemplo, uma simples página HTML fornece um aplicativo baseado em navegador para visualizar as fotos em um álbum de fotos. O álbum de fotos está em um bucket do Amazon S3 no qual as fotos são carregadas.

O script do navegador usa o SDK para JavaScript para interagir com um bucket do Amazon S3. O script usa o método `listObjects` da classe cliente do Amazon S3 para permitir que você visualize os álbuns de fotos.

Tarefas de pré-requisito

Para configurar e executar este exemplo, primeiro conclua estas tarefas.

> **Note**
>
> Neste exemplo, você deve usar a mesma região da AWS para o bucket do Amazon S3 e o banco de identidades do Amazon Cognito.

Criar o bucket

No console do Amazon S3, crie um bucket do Amazon S3 em que você possa armazenar álbuns e fotos. Para obter mais informações sobre como criar um bucket do S3, consulte Criar um bucket no Guia do usuário do Amazon Simple Storage Service.

À medida que você criar o bucket do S3, faça o seguinte:

- Anote o nome do bucket para que você possa usá-lo em uma tarefa de pré-requisito subsequente, Configurar permissões de função.
- Escolha uma região da AWS na qual criar o bucket. Essa deve ser a mesma região que você usará para criar um grupo de identidades do Amazon Cognito em uma tarefa de pré-requisito subsequente, Criar um grupo de identidades.
- Configure permissões de bucket seguindo as instruções em Setting permissions for website access no Guia do usuário do Amazon Simple Storage Service.

Criar um grupo de identidades do

No console do Amazon Cognito, crie um banco de identidades do Amazon Cognito conforme é descrito em the section called “Etapa 1: Criar um banco de identidades do Amazon Cognito” do tópico Conceitos básicos de um script de navegador.

Ao criar o banco de identidades, anote o nome dele, bem como o nome do perfil da identidade não autenticada.

Configurar permissões de função

Para permitir a visualização de álbuns e fotos, você precisa adicionar permissões a uma função do IAM do grupo de identidades que acabou de criar. Comece criando uma política como a seguir.

1. Abra o console do IAM.
2. No painel de navegação à esquerda, escolha Policies (Políticas) e o botão Create policy (Criar política).
3. Na guia JSON, insira a definição JSON a seguir, mas substitua BUCKET_NAME pelo nome do bucket.

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Effect": "Allow",
         "Action": [
            "s3:ListBucket"
         ],
         "Resource": [
            "arn:aws:s3:::BUCKET_NAME"
         ]
      }
   ]
}
```

4. Escolha o botão Review policy (Revisar política), nomeie a política, forneça uma descrição (se desejar) e escolha o botão Create policy (Criar política).

   Certifique-se de anotar o nome para que você possa encontrá-lo e anexá-lo à função do IAM mais tarde.

Depois que a política for criada, navegue de volta ao console do IAM. Encontre o perfil do IAM para a identidade não autenticada que o Amazon Cognito criou na tarefa de pré-requisito anterior, Criar um banco de identidades. Você usa a política recém-criada para adicionar permissões a essa identidade.

Embora o fluxo de trabalho desta tarefa geralmente seja igual ao da the section called “Etapa 2: Adicionar uma política ao perfil do IAM criado” do tópico Conceitos básicos de um script de navegador, há algumas diferenças a serem observadas:

- Use a nova política que você acabou de criar, não uma política para Amazon Polly.
- Na página Attach Permissions (Anexar permissões), para encontrar rapidamente a nova política, abra a lista Filter policies (Filtrar políticas) e escolha Customer managed (Gerenciadas pelo cliente).

Para obter mais informações sobre como criar um perfil do IAM, consulte Criação de um perfil para delegar permissões a um serviço da AWS no Guia do usuário do IAM.

Configurar o CORS

Para que o script do navegador possa acessar o bucket do Amazon S3, você precisa fazer a configuração do CORS como a seguir.

> **Important**
>
> No novo console do S3, a configuração CORS deve ser JSON.

JSON

```
[
    {
        "AllowedHeaders": [
            "*"
        ],
        "AllowedMethods": [
            "HEAD",
            "GET"
        ],
        "AllowedOrigins": [
            "*"
```

```
        ]
    }
]
```

XML

```
<?xml version="1.0" encoding="UTF-8"?>
<CORSConfiguration xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
    <CORSRule>
        <AllowedOrigin>*</AllowedOrigin>
        <AllowedMethod>GET</AllowedMethod>
        <AllowedMethod>HEAD</AllowedMethod>
        <AllowedHeader>*</AllowedHeader>
    </CORSRule>
</CORSConfiguration>
```

Criar álbuns e fazer upload de fotos

Como este exemplo permite apenas que os usuários visualizem as fotos que já estiverem no bucket, você precisa criar alguns álbuns no bucket e fazer upload de fotos neles.

> **Note**
>
> Para este exemplo, os nomes dos arquivos de fotos devem começar com um sublinhado único ("_"). Esse caractere é importante para a filtragem mais tarde. Além disso, certifique-se de respeitar os direitos autorais dos proprietários das fotos.

1. No console do Amazon S3, abra o bucket que você criou anteriormente.
2. Na guia Overview (Visão geral), escolha o botão Create folder (Criar pasta) para criar pastas. Para este exemplo, nomeie as pastas como "album1", "album2" e "album3".
3. Para album1 e album2, selecione a pasta e, depois, fazer upload de fotos nela como a seguir:
   a. Escolha o botão Upload (Fazer upload).
   b. Arraste ou escolha os arquivos de fotos que deseja usar e escolha Next (Próximo).
   c. Em Manage public permissions (Gerenciar permissões públicas), escolha Grant public read access to this object(s) (Conceder acesso público de leitura a este(s) objeto(s)).
   d. Escolha o botão Upload (Fazer upload) (no canto inferior esquerdo).

4. Deixe album3 vazio.

Definir a página da web

O HTML do aplicativo de exibição de fotos consiste em um elemento `<div>` no qual o script do navegador cria a interface de visualização. O primeiro elemento `<script>` adiciona o SDK ao script do navegador. O segundo elemento `<script>` adiciona o arquivo JavaScript externo que contém o código do script do navegador.

Para este exemplo, o arquivo é denominado `PhotoViewer.js` e está localizado na mesma pasta que o arquivo HTML. Para encontrar o SDK_VERSION_NUMBER atual, consulte a Referência da API para o SDK para JavaScript no Guia de referência da API. AWS SDK for JavaScript

```
<!DOCTYPE html>
<html>
  <head>
    <!-- **DO THIS**: -->
    <!--   Replace SDK_VERSION_NUMBER with the current SDK version number -->
    <script src="https://sdk.amazonaws.com/js/aws-sdk-SDK_VERSION_NUMBER.js"></script>
    <script src="./PhotoViewer.js"></script>
    <script>listAlbums();</script>
  </head>
  <body>
    <h1>Photo Album Viewer</h1>
    <div id="viewer" />
  </body>
</html>
```

Como configurar o SDK

Obtenha as credenciais necessárias para configurar o SDK chamando o método `CognitoIdentityCredentials`. Você precisa fornecer o ID do grupo de identidades do Amazon Cognito. Depois crie um objeto de serviço `AWS.S3`.

```
// **DO THIS**:
//   Replace BUCKET_NAME with the bucket name.
//
var albumBucketName = "BUCKET_NAME";

// **DO THIS**:
//   Replace this block of code with the sample code located at:
```

```
//   Cognito -- Manage Identity Pools -- [identity_pool_name] -- Sample Code --
 JavaScript
//
// Initialize the Amazon Cognito credentials provider
AWS.config.region = "REGION"; // Region
AWS.config.credentials = new AWS.CognitoIdentityCredentials({
  IdentityPoolId: "IDENTITY_POOL_ID",
});

// Create a new service object
var s3 = new AWS.S3({
  apiVersion: "2006-03-01",
  params: { Bucket: albumBucketName },
});

// A utility function to create HTML.
function getHtml(template) {
  return template.join("\n");
}
```

O restante do código neste exemplo define as funções a seguir para coletar e apresentar informações sobre os álbuns e as fotos no bucket.

- `listAlbums`
- `viewAlbum`

Listar álbuns no bucket

Para listar todos os álbuns existentes no bucket, a função `listAlbums` do aplicativo chama o método `listObjects` do objeto de serviço `AWS.S3`. A função usa a propriedade `CommonPrefixes` para que a chamada retorne somente os objetos que são usados como álbuns (ou seja, as pastas).

O restante da função utiliza a lista de álbuns do bucket do Amazon S3 e gera o HTML necessário para exibir a lista de álbuns na página da web.

```
// List the photo albums that exist in the bucket.
function listAlbums() {
  s3.listObjects({ Delimiter: "/" }, function (err, data) {
    if (err) {
      return alert("There was an error listing your albums: " + err.message);
    } else {
      var albums = data.CommonPrefixes.map(function (commonPrefix) {
```

```
        var prefix = commonPrefix.Prefix;
        var albumName = decodeURIComponent(prefix.replace("/", ""));
        return getHtml([
          "<li>",
          '<button style="margin:5px;" onclick="viewAlbum(\'' +
            albumName +
            "')\">",
          albumName,
          "</button>",
          "</li>",
        ]);
      });
      var message = albums.length
        ? getHtml(["<p>Click on an album name to view it.</p>"])
        : "<p>You do not have any albums. Please Create album.";
      var htmlTemplate = [
        "<h2>Albums</h2>",
        message,
        "<ul>",
        getHtml(albums),
        "</ul>",
      ];
      document.getElementById("viewer").innerHTML = getHtml(htmlTemplate);
    }
  });
}
```

Visualizar um álbum

Para exibir o conteúdo de um álbum no bucket do Amazon S3, a função `viewAlbum` do aplicativo assumirá o nome de um álbum e criará a chave do Amazon S3 para esse álbum. A função chama então o método `listObjects` do objeto de serviço `AWS.S3` para obter uma lista de todos os objetos (as fotos) do álbum.

O restante da função utiliza a lista de objetos do álbum e gera o HTML necessário para exibir as fotos na página da web.

```
// Show the photos that exist in an album.
function viewAlbum(albumName) {
  var albumPhotosKey = encodeURIComponent(albumName) + "/";
  s3.listObjects({ Prefix: albumPhotosKey }, function (err, data) {
    if (err) {
      return alert("There was an error viewing your album: " + err.message);
```

```
    }
    // 'this' references the AWS.Request instance that represents the response
    var href = this.request.httpRequest.endpoint.href;
    var bucketUrl = href + albumBucketName + "/";

    var photos = data.Contents.map(function (photo) {
      var photoKey = photo.Key;
      var photoUrl = bucketUrl + encodeURIComponent(photoKey);
      return getHtml([
        "<span>",
        "<div>",
        "<br/>",
        '<img style="width:128px;height:128px;" src="' + photoUrl + '"/>',
        "</div>",
        "<div>",
        "<span>",
        photoKey.replace(albumPhotosKey, ""),
        "</span>",
        "</div>",
        "</span>",
      ]);
    });
    var message = photos.length
      ? "<p>The following photos are present.</p>"
      : "<p>There are no photos in this album.</p>";
    var htmlTemplate = [
      "<div>",
      '<button onclick="listAlbums()">',
      "Back To Albums",
      "</button>",
      "</div>",
      "<h2>",
      "Album: " + albumName,
      "</h2>",
      message,
      "<div>",
      getHtml(photos),
      "</div>",
      "<h2>",
      "End of Album: " + albumName,
      "</h2>",
      "<div>",
      '<button onclick="listAlbums()">',
      "Back To Albums",
```

```
      "</button>",
      "</div>",
    ];
    document.getElementById("viewer").innerHTML = getHtml(htmlTemplate);
    document
      .getElementsByTagName("img")[0]
      .setAttribute("style", "display:none;");
  });
}
```

Visualizar fotos em um bucket do Amazon S3: Código completo

Esta seção contém todo o código HTML e JavaScript para o exemplo em que as fotos em um bucket do Amazon S3 podem ser visualizadas. Consulte a seção principal para obter detalhes e pré-requisitos.

O HTML para o exemplo:

```
<!DOCTYPE html>
<html>
  <head>
    <!-- **DO THIS**: -->
    <!--   Replace SDK_VERSION_NUMBER with the current SDK version number -->
    <script src="https://sdk.amazonaws.com/js/aws-sdk-SDK_VERSION_NUMBER.js"></script>
    <script src="./PhotoViewer.js"></script>
    <script>listAlbums();</script>
  </head>
  <body>
    <h1>Photo Album Viewer</h1>
    <div id="viewer" />
  </body>
</html>
```

Este código de exemplo pode ser encontrado aqui no GitHub.

O código do script do navegador para o exemplo:

```
//
// Data constructs and initialization.
//

// **DO THIS**:
//   Replace BUCKET_NAME with the bucket name.
```

```
//
var albumBucketName = "BUCKET_NAME";

// **DO THIS**:
//   Replace this block of code with the sample code located at:
//   Cognito -- Manage Identity Pools -- [identity_pool_name] -- Sample Code --
 JavaScript
//
// Initialize the Amazon Cognito credentials provider
AWS.config.region = "REGION"; // Region
AWS.config.credentials = new AWS.CognitoIdentityCredentials({
  IdentityPoolId: "IDENTITY_POOL_ID",
});

// Create a new service object
var s3 = new AWS.S3({
  apiVersion: "2006-03-01",
  params: { Bucket: albumBucketName },
});

// A utility function to create HTML.
function getHtml(template) {
  return template.join("\n");
}

//
// Functions
//

// List the photo albums that exist in the bucket.
function listAlbums() {
  s3.listObjects({ Delimiter: "/" }, function (err, data) {
    if (err) {
      return alert("There was an error listing your albums: " + err.message);
    } else {
      var albums = data.CommonPrefixes.map(function (commonPrefix) {
        var prefix = commonPrefix.Prefix;
        var albumName = decodeURIComponent(prefix.replace("/", ""));
        return getHtml([
          "<li>",
          '<button style="margin:5px;" onclick="viewAlbum(\'' +
            albumName +
            "')\">",
          albumName,
```

```
          "</button>",
          "</li>",
        ]);
      });
      var message = albums.length
        ? getHtml(["<p>Click on an album name to view it.</p>"])
        : "<p>You do not have any albums. Please Create album.";
      var htmlTemplate = [
        "<h2>Albums</h2>",
        message,
        "<ul>",
        getHtml(albums),
        "</ul>",
      ];
      document.getElementById("viewer").innerHTML = getHtml(htmlTemplate);
    }
  });
}

// Show the photos that exist in an album.
function viewAlbum(albumName) {
  var albumPhotosKey = encodeURIComponent(albumName) + "/";
  s3.listObjects({ Prefix: albumPhotosKey }, function (err, data) {
    if (err) {
      return alert("There was an error viewing your album: " + err.message);
    }
    // 'this' references the AWS.Request instance that represents the response
    var href = this.request.httpRequest.endpoint.href;
    var bucketUrl = href + albumBucketName + "/";

    var photos = data.Contents.map(function (photo) {
      var photoKey = photo.Key;
      var photoUrl = bucketUrl + encodeURIComponent(photoKey);
      return getHtml([
        "<span>",
        "<div>",
        "<br/>",
        '<img style="width:128px;height:128px;" src="' + photoUrl + '"/>',
        "</div>",
        "<div>",
        "<span>",
        photoKey.replace(albumPhotosKey, ""),
        "</span>",
        "</div>",
```

```
        "</span>",
      ]);
    });
    var message = photos.length
      ? "<p>The following photos are present.</p>"
      : "<p>There are no photos in this album.</p>";
    var htmlTemplate = [
      "<div>",
      '<button onclick="listAlbums()">',
      "Back To Albums",
      "</button>",
      "</div>",
      "<h2>",
      "Album: " + albumName,
      "</h2>",
      message,
      "<div>",
      getHtml(photos),
      "</div>",
      "<h2>",
      "End of Album: " + albumName,
      "</h2>",
      "<div>",
      '<button onclick="listAlbums()">',
      "Back To Albums",
      "</button>",
      "</div>",
    ];
    document.getElementById("viewer").innerHTML = getHtml(htmlTemplate);
    document
      .getElementsByTagName("img")[0]
      .setAttribute("style", "display:none;");
  });
}
```

Este código de exemplo pode ser encontrado aqui no GitHub.

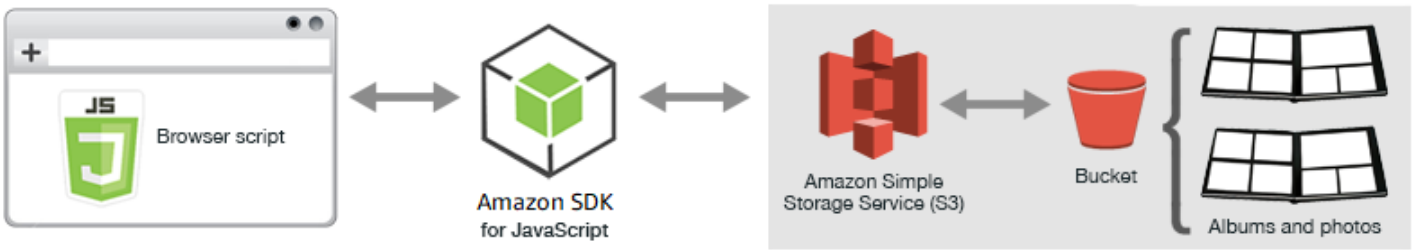
Fazer upload de fotos para o Amazon S3 partir de um navegador

Este exemplo de código de script do navegador mostra:

- Como criar um aplicativo de navegador que permita que os usuários criem álbuns de fotos em um bucket do Amazon S3 e façam upload de fotos nos álbuns.

O cenário

Neste exemplo, uma página em HTML simples fornece um aplicativo no navegador para criar álbuns de fotos em um bucket do Amazon S3 para o qual você pode enviar fotos. O aplicativo permite que você exclua fotos e álbuns que adicionar.



O script do navegador usa o SDK para JavaScript para interagir com um bucket do Amazon S3. Use os seguintes métodos da classe de cliente Amazon S3 para habilitar o aplicativo do álbum de fotos:

- `listObjects`
- `headObject`
- `putObject`
- `upload`
- `deleteObject`
- `deleteObjects`

Tarefas de pré-requisito

Para configurar e executar este exemplo, você deve primeiro concluir estas tarefas:

- No console do console do Amazon S3, crie um bucket do Amazon S3 para usar ao armazenar as fotos no álbum. Para obter mais informações sobre como criar um bucket, consulte Criar um bucket, no Guia do usuário do Amazon Simple Storage Service. Você deve ter as permissões Read (Leitura) e Write (Gravação) de Objects (Objetos). Para obter mais informações sobre definir permissões de bucket, consulte Definir permissões para acesso ao site.

- No console do Amazon Cognito, crie um banco de identidades do usando identidades federadas com acesso habilitado para usuários não autenticados na mesma região que o bucket do Amazon S3. Você precisa incluir o ID do grupo de identidades no código para obter credenciais para o script do navegador. Para obter mais informações sobre as identidades federadas do Amazon Cognito, consulte Bancos de identidade do Amazon Cognito (Identidades federadas (identidades federadas) no Guia do desenvolvedor do Amazon Cognito.
- No console do IAM, localize o perfil do IAM criado pelo Amazon Cognito para usuários não autenticados. Adicione a política a seguir para conceder permissões de leitura e gravação a um bucket do Amazon S3. Para obter mais informações sobre como criar um perfil do IAM, consulte Criação de uma função para delegar permissões a um serviço da AWS no Guia do usuário do IAM.

  Use essa política de perfil para o perfil do criada pelo para usuários não autenticados.

  > **Warning**
  >
  > Se habilitar o acesso para usuários não autenticados, você concederá acesso de gravação no bucket, e todos os objetos do bucket, a qualquer pessoa no mundo. Essa postura de segurança é útil neste exemplo para manter o foco nos principais objetivos do exemplo. Em muitas situações em tempo real, porém, é altamente recomendável usar uma segurança mais restrita, como usuários autenticados e propriedade de objetos.

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Effect": "Allow",
         "Action": [
            "s3:DeleteObject",
            "s3:GetObject",
            "s3:ListBucket",
            "s3:PutObject",
            "s3:PutObjectAcl"
         ],
         "Resource": [
            "arn:aws:s3:::BUCKET_NAME",
            "arn:aws:s3:::BUCKET_NAME/*"
         ]
      }
```

```
    ]
}
```

Configurar CORS

Antes de o script do navegador acessar o bucket do Amazon S3, primeiro faça a configuração do CORS como a seguir.

> **Important**
>
> No novo console do S3, a configuração CORS deve ser JSON.

JSON

```
[
    {
        "AllowedHeaders": [
            "*"
        ],
        "AllowedMethods": [
            "HEAD",
            "GET",
            "PUT",
            "POST",
            "DELETE"
        ],
        "AllowedOrigins": [
            "*"
        ],
        "ExposeHeaders": [
            "ETag"
        ]
    }
]
```

XML

```
<?xml version="1.0" encoding="UTF-8"?>
<CORSConfiguration xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
```

```
    <CORSRule>
        <AllowedOrigin>*</AllowedOrigin>
        <AllowedMethod>POST</AllowedMethod>
        <AllowedMethod>GET</AllowedMethod>
        <AllowedMethod>PUT</AllowedMethod>
        <AllowedMethod>DELETE</AllowedMethod>
        <AllowedMethod>HEAD</AllowedMethod>
        <AllowedHeader>*</AllowedHeader>
        <ExposeHeader>ETag</ExposeHeader>
    </CORSRule>
</CORSConfiguration>
```

A página da web

O HTML do aplicativo de upload de fotos é formado por um elemento ** ** dentro do qual o script do navegador cria a interface do usuário do upload. O primeiro elemento **<script>** adiciona o SDK ao script do navegador. O segundo elemento **<script>** adiciona o arquivo JavaScript externo que contém o código de script do navegador.

```
<!DOCTYPE html>
<html>
  <head>
     <!-- **DO THIS**: -->
    <!--   Replace SDK_VERSION_NUMBER with the current SDK version number -->
    <script src="https://sdk.amazonaws.com/js/aws-sdk-SDK_VERSION_NUMBER.js"></script>
    <script src="./s3_photoExample.js"></script>
    <script>
       function getHtml(template) {
          return template.join('\n');
       }
       listAlbums();
    </script>
  </head>
  <body>
    <h1>My Photo Albums App</h1>
    <div id="app"></div>
  </body>
</html>
```

Como configurar o SDK

Obtenha as credenciais necessárias para configurar o SDK chamando o método `CognitoIdentityCredentials`, fornecendo o ID do grupo de identidades do Amazon Cognito. Em seguida, crie um objeto de serviço do `AWS.S3`.

```
var albumBucketName = "BUCKET_NAME";
var bucketRegion = "REGION";
var IdentityPoolId = "IDENTITY_POOL_ID";

AWS.config.update({
  region: bucketRegion,
  credentials: new AWS.CognitoIdentityCredentials({
    IdentityPoolId: IdentityPoolId,
  }),
});

var s3 = new AWS.S3({
  apiVersion: "2006-03-01",
  params: { Bucket: albumBucketName },
});
```

Quase todo o restante do código neste exemplo é organizado em uma série de funções que coletam e apresentam informações sobre os álbuns do bucket, sobem e exibem as fotos carregadas dentro dos álbuns e excluem fotos e álbuns. Essas funções são:

- `listAlbums`
- `createAlbum`
- `viewAlbum`
- `addPhoto`
- `deleteAlbum`
- `deletePhoto`

Listar álbuns no bucket

O aplicativo cria álbuns no bucket do Amazon S3 como objetos cujas chaves comecem com um caractere de barra, indicando que o objeto funciona como uma pasta. Para listar todos os álbuns existentes no bucket, a função `listAlbums` do aplicativo chama o método `listObjects` do objeto

de serviço do `AWS.S3` ao usar `commonPrefix` para que a chamada retorne apenas objetos usados como álbuns.

O restante da função utiliza a lista de álbuns do bucket do Amazon S3 e gera o HTML necessário para exibir a lista de álbuns na página da web. Ela também permite excluir e abrir álbuns individuais.

```
function listAlbums() {
  s3.listObjects({ Delimiter: "/" }, function (err, data) {
    if (err) {
      return alert("There was an error listing your albums: " + err.message);
    } else {
      var albums = data.CommonPrefixes.map(function (commonPrefix) {
        var prefix = commonPrefix.Prefix;
        var albumName = decodeURIComponent(prefix.replace("/", ""));
        return getHtml([
          "<li>",
          "<span onclick=\"deleteAlbum('" + albumName + "')\">X</span>",
          "<span onclick=\"viewAlbum('" + albumName + "')\">",
          albumName,
          "</span>",
          "</li>",
        ]);
      });
      var message = albums.length
        ? getHtml([
            "<p>Click on an album name to view it.</p>",
            "<p>Click on the X to delete the album.</p>",
          ])
        : "<p>You do not have any albums. Please Create album.";
      var htmlTemplate = [
        "<h2>Albums</h2>",
        message,
        "<ul>",
        getHtml(albums),
        "</ul>",
        "<button onclick=\"createAlbum(prompt('Enter Album Name:'))\">",
        "Create New Album",
        "</button>",
      ];
      document.getElementById("app").innerHTML = getHtml(htmlTemplate);
    }
  });
}
```

Criar um álbum no bucket

Para criar um álbum no bucket do Amazon S3, a função `createAlbum` do aplicativo primeiro valida o nome para o novo álbum para garantir que ele contém caracteres adequados. Em seguida, forma uma chave de objeto do Amazon S3, passando para o método `headObject` do objeto de serviço do Amazon S3. Esse método retorna os metadados para a chave especificada, de forma que, se retornar dados, já existe um objeto com essa chave.

Se o álbum ainda não existir, a função chamará o método `putObject` do objeto de serviço do `AWS.S3` para criar o álbum. Em seguida, chama a função `viewAlbum` para exibir o novo álbum vazio.

```
function createAlbum(albumName) {
  albumName = albumName.trim();
  if (!albumName) {
    return alert("Album names must contain at least one non-space character.");
  }
  if (albumName.indexOf("/") !== -1) {
    return alert("Album names cannot contain slashes.");
  }
  var albumKey = encodeURIComponent(albumName);
  s3.headObject({ Key: albumKey }, function (err, data) {
    if (!err) {
      return alert("Album already exists.");
    }
    if (err.code !== "NotFound") {
      return alert("There was an error creating your album: " + err.message);
    }
    s3.putObject({ Key: albumKey }, function (err, data) {
      if (err) {
        return alert("There was an error creating your album: " + err.message);
      }
      alert("Successfully created album.");
      viewAlbum(albumName);
    });
  });
}
```

Visualizar um álbum

Para exibir o conteúdo de um álbum no bucket do Amazon S3, a função `viewAlbum` do aplicativo assumirá o nome de um álbum e criará a chave do Amazon S3 para esse álbum. A função chama o

método `listObjects` do objeto de serviço do `AWS.S3` para obter uma lista com todos os objetos (fotos) no álbum.

O restante da função utiliza a lista de objetos (fotos) do álbum e gera o HTML necessário para exibir as fotos na página da web. Ela também permite a exclusão de fotos individuais e que se volte para a lista de álbuns.

```
function viewAlbum(albumName) {
  var albumPhotosKey = encodeURIComponent(albumName) + "/";
  s3.listObjects({ Prefix: albumPhotosKey }, function (err, data) {
    if (err) {
      return alert("There was an error viewing your album: " + err.message);
    }
    // 'this' references the AWS.Response instance that represents the response
    var href = this.request.httpRequest.endpoint.href;
    var bucketUrl = href + albumBucketName + "/";

    var photos = data.Contents.map(function (photo) {
      var photoKey = photo.Key;
      var photoUrl = bucketUrl + encodeURIComponent(photoKey);
      return getHtml([
        "<span>",
        "<div>",
        '<img style="width:128px;height:128px;" src="' + photoUrl + '"/>',
        "</div>",
        "<div>",
        "<span onclick=\"deletePhoto('" +
          albumName +
          "','" +
          photoKey +
          "')\">",
        "X",
        "</span>",
        "<span>",
        photoKey.replace(albumPhotosKey, ""),
        "</span>",
        "</div>",
        "</span>",
      ]);
    });
    var message = photos.length
      ? "<p>Click on the X to delete the photo</p>"
      : "<p>You do not have any photos in this album. Please add photos.</p>";
```

```
    var htmlTemplate = [
      "<h2>",
      "Album: " + albumName,
      "</h2>",
      message,
      "<div>",
      getHtml(photos),
      "</div>",
      '<input id="photoupload" type="file" accept="image/*">',
      '<button id="addphoto" onclick="addPhoto(\'' + albumName + "')\">",
      "Add Photo",
      "</button>",
      '<button onclick="listAlbums()">',
      "Back To Albums",
      "</button>",
    ];
    document.getElementById("app").innerHTML = getHtml(htmlTemplate);
  });
}
```

Adicionar fotos a um álbum

Para fazer upload de uma foto a um álbum no bucket do Amazon S3, a função `addPhoto` do aplicativo usa um elemento seletor na página da web para identificar um arquivo para upload. Em seguida, forma uma chave para a foto ser carregada pelo nome do álbum atual e pelo nome do arquivo.

A função chama o método `upload` do objeto de serviço do Amazon S3 para fazer upload da foto. Depois de carregar a foto, a função reexibe o álbum, para que a foto carregada seja exibida.

```
function addPhoto(albumName) {
  var files = document.getElementById("photoupload").files;
  if (!files.length) {
    return alert("Please choose a file to upload first.");
  }
  var file = files[0];
  var fileName = file.name;
  var albumPhotosKey = encodeURIComponent(albumName) + "/";

  var photoKey = albumPhotosKey + fileName;

  // Use S3 ManagedUpload class as it supports multipart uploads
  var upload = new AWS.S3.ManagedUpload({
```

```
    params: {
      Bucket: albumBucketName,
      Key: photoKey,
      Body: file,
    },
  });

  var promise = upload.promise();

  promise.then(
    function (data) {
      alert("Successfully uploaded photo.");
      viewAlbum(albumName);
    },
    function (err) {
      return alert("There was an error uploading your photo: ", err.message);
    }
  );
}
```

Excluir uma foto

Para excluir uma foto de um álbum no bucket do Amazon S3, a função `deletePhoto` do aplicativo chama o método `deleteObject` do objeto de serviço do Amazon S3. Isso exclui a foto especificada pelo valor `photoKey` passado à função.

```
function deletePhoto(albumName, photoKey) {
  s3.deleteObject({ Key: photoKey }, function (err, data) {
    if (err) {
      return alert("There was an error deleting your photo: ", err.message);
    }
    alert("Successfully deleted photo.");
    viewAlbum(albumName);
  });
}
```

Excluir um álbum

Para excluir um álbum no bucket do Amazon S3, a função `deleteAlbum` do aplicativo chama o método `deleteObjects` do objeto de serviço do Amazon S3.

```
function deleteAlbum(albumName) {
  var albumKey = encodeURIComponent(albumName) + "/";
```

```
  s3.listObjects({ Prefix: albumKey }, function (err, data) {
    if (err) {
      return alert("There was an error deleting your album: ", err.message);
    }
    var objects = data.Contents.map(function (object) {
      return { Key: object.Key };
    });
    s3.deleteObjects(
      {
        Delete: { Objects: objects, Quiet: true },
      },
      function (err, data) {
        if (err) {
          return alert("There was an error deleting your album: ", err.message);
        }
        alert("Successfully deleted album.");
        listAlbums();
      }
    );
  });
}
```

Fazer upload de fotos no Amazon S3: código completo

Esta seção contém todo o código HTML e JavaScript para o exemplo em que é feito o upload de fotos em um álbum de fotos do Amazon S3. Consulte a seção principal para obter detalhes e pré-requisitos.

O HTML para o exemplo:

```
<!DOCTYPE html>
<html>
  <head>
     <!-- **DO THIS**: -->
    <!--   Replace SDK_VERSION_NUMBER with the current SDK version number -->
    <script src="https://sdk.amazonaws.com/js/aws-sdk-SDK_VERSION_NUMBER.js"></script>
    <script src="./s3_photoExample.js"></script>
    <script>
       function getHtml(template) {
          return template.join('\n');
       }
       listAlbums();
    </script>
```

```
  </head>
  <body>
    <h1>My Photo Albums App</h1>
    <div id="app"></div>
  </body>
</html>
```

Este código de exemplo pode ser encontrado aqui no GitHub.

O código do script do navegador para o exemplo:

```
var albumBucketName = "BUCKET_NAME";
var bucketRegion = "REGION";
var IdentityPoolId = "IDENTITY_POOL_ID";

AWS.config.update({
  region: bucketRegion,
  credentials: new AWS.CognitoIdentityCredentials({
    IdentityPoolId: IdentityPoolId,
  }),
});

var s3 = new AWS.S3({
  apiVersion: "2006-03-01",
  params: { Bucket: albumBucketName },
});

function listAlbums() {
  s3.listObjects({ Delimiter: "/" }, function (err, data) {
    if (err) {
      return alert("There was an error listing your albums: " + err.message);
    } else {
      var albums = data.CommonPrefixes.map(function (commonPrefix) {
        var prefix = commonPrefix.Prefix;
        var albumName = decodeURIComponent(prefix.replace("/", ""));
        return getHtml([
          "<li>",
          "<span onclick=\"deleteAlbum('" + albumName + "')\">X</span>",
          "<span onclick=\"viewAlbum('" + albumName + "')\">",
          albumName,
          "</span>",
          "</li>",
        ]);
      });
```

```
      var message = albums.length
        ? getHtml([
            "<p>Click on an album name to view it.</p>",
            "<p>Click on the X to delete the album.</p>",
          ])
        : "<p>You do not have any albums. Please Create album.";
      var htmlTemplate = [
        "<h2>Albums</h2>",
        message,
        "<ul>",
        getHtml(albums),
        "</ul>",
        "<button onclick=\"createAlbum(prompt('Enter Album Name:'))\">",
        "Create New Album",
        "</button>",
      ];
      document.getElementById("app").innerHTML = getHtml(htmlTemplate);
    }
  });
}

function createAlbum(albumName) {
  albumName = albumName.trim();
  if (!albumName) {
    return alert("Album names must contain at least one non-space character.");
  }
  if (albumName.indexOf("/") !== -1) {
    return alert("Album names cannot contain slashes.");
  }
  var albumKey = encodeURIComponent(albumName);
  s3.headObject({ Key: albumKey }, function (err, data) {
    if (!err) {
      return alert("Album already exists.");
    }
    if (err.code !== "NotFound") {
      return alert("There was an error creating your album: " + err.message);
    }
    s3.putObject({ Key: albumKey }, function (err, data) {
      if (err) {
        return alert("There was an error creating your album: " + err.message);
      }
      alert("Successfully created album.");
      viewAlbum(albumName);
    });
```

```
  });
}

function viewAlbum(albumName) {
  var albumPhotosKey = encodeURIComponent(albumName) + "/";
  s3.listObjects({ Prefix: albumPhotosKey }, function (err, data) {
    if (err) {
      return alert("There was an error viewing your album: " + err.message);
    }
    // 'this' references the AWS.Response instance that represents the response
    var href = this.request.httpRequest.endpoint.href;
    var bucketUrl = href + albumBucketName + "/";

    var photos = data.Contents.map(function (photo) {
      var photoKey = photo.Key;
      var photoUrl = bucketUrl + encodeURIComponent(photoKey);
      return getHtml([
        "<span>",
        "<div>",
        '<img style="width:128px;height:128px;" src="' + photoUrl + '"/>',
        "</div>",
        "<div>",
        "<span onclick=\"deletePhoto('" +
          albumName +
          "','" +
          photoKey +
          "')\">",
        "X",
        "</span>",
        "<span>",
        photoKey.replace(albumPhotosKey, ""),
        "</span>",
        "</div>",
        "</span>",
      ]);
    });
    var message = photos.length
      ? "<p>Click on the X to delete the photo</p>"
      : "<p>You do not have any photos in this album. Please add photos.</p>";
    var htmlTemplate = [
      "<h2>",
      "Album: " + albumName,
      "</h2>",
      message,
```

```
      "<div>",
      getHtml(photos),
      "</div>",
      '<input id="photoupload" type="file" accept="image/*">',
      '<button id="addphoto" onclick="addPhoto(\'' + albumName + "')\">",
      "Add Photo",
      "</button>",
      '<button onclick="listAlbums()">',
      "Back To Albums",
      "</button>",
    ];
    document.getElementById("app").innerHTML = getHtml(htmlTemplate);
  });
}

function addPhoto(albumName) {
  var files = document.getElementById("photoupload").files;
  if (!files.length) {
    return alert("Please choose a file to upload first.");
  }
  var file = files[0];
  var fileName = file.name;
  var albumPhotosKey = encodeURIComponent(albumName) + "/";

  var photoKey = albumPhotosKey + fileName;

  // Use S3 ManagedUpload class as it supports multipart uploads
  var upload = new AWS.S3.ManagedUpload({
    params: {
      Bucket: albumBucketName,
      Key: photoKey,
      Body: file,
    },
  });

  var promise = upload.promise();

  promise.then(
    function (data) {
      alert("Successfully uploaded photo.");
      viewAlbum(albumName);
    },
    function (err) {
      return alert("There was an error uploading your photo: ", err.message);
```

```
    }
  );
}

function deletePhoto(albumName, photoKey) {
  s3.deleteObject({ Key: photoKey }, function (err, data) {
    if (err) {
      return alert("There was an error deleting your photo: ", err.message);
    }
    alert("Successfully deleted photo.");
    viewAlbum(albumName);
  });
}

function deleteAlbum(albumName) {
  var albumKey = encodeURIComponent(albumName) + "/";
  s3.listObjects({ Prefix: albumKey }, function (err, data) {
    if (err) {
      return alert("There was an error deleting your album: ", err.message);
    }
    var objects = data.Contents.map(function (object) {
      return { Key: object.Key };
    });
    s3.deleteObjects(
      {
        Delete: { Objects: objects, Quiet: true },
      },
      function (err, data) {
        if (err) {
          return alert("There was an error deleting your album: ", err.message);
        }
        alert("Successfully deleted album.");
        listAlbums();
      }
    );
  });
}
```

Este código de exemplo pode ser encontrado aqui no GitHub.

Exemplos de Node.js do Amazon S3

Os tópicos a seguir mostram exemplos de como o AWS SDK for JavaScript pode ser usado para interagir com buckets do usando Node.js.

Tópicos

- Criação e uso de buckets do Amazon S3
- Configurar buckets do Amazon S3
- Gerenciar permissões de acesso a buckets do Amazon S3
- Trabalhar com políticas de bucket do Amazon S3
- Usar um bucket do Amazon S3 como um host estático da Web

Criação e uso de buckets do Amazon S3

Este exemplo de código Node.js mostra:

- Como obter e exibir uma lista de buckets do Amazon S3 na sua conta.
- Como criar um bucket do Amazon S3.
- Como fazer upload de um objeto em um bucket especificado.

O cenário

Neste exemplo, uma série de módulos do Node.js é usada para obter uma lista dos buckets do Amazon S3 existentes, criar um bucket e fazer upload do arquivo a um bucket específico. Esses módulos do Node.js usam o SDK para JavaScript para obter informações dos arquivos e carregá-los em um bucket do Amazon S3 usando estes métodos da classe de cliente do Amazon S3:

- `listBuckets`
- `createBucket`
- `listObjects`
- `upload`

- `deleteBucket`

Tarefas de pré-requisito

Para configurar e executar este exemplo, você deve primeiro concluir estas tarefas:

- Instale o Node.js. Para obter mais informações sobre como instalar Node.js, consulte o website de Node.js.
- Crie um arquivo de configurações compartilhado com as credenciais de usuário. Para obter mais informações sobre como fornecer um arquivo de credenciais compartilhadas, consulte Carregar credenciais em Node.js do arquivo de credenciais compartilhado.

Como configurar o SDK

Configure o SDK para JavaScript criando um objeto de configuração global e definindo a região do código. Neste exemplo, a região é definida como `us-west-2`.

```
// Load the SDK for JavaScript
var AWS = require('aws-sdk');
// Set the Region
AWS.config.update({region: 'us-west-2'});
```

Exibir uma lista de buckets no Amazon S3

Crie um módulo do Node.js com o nome de arquivo `s3_listbuckets.js`. Configure o SDK conforme mostrado anteriormente. Para acessar o Amazon Simple Storage Service, crie um objeto de serviço de `AWS.S3`. Chame o método `listBuckets` do objeto de serviço do Amazon S3 para recuperar uma lista dos seus buckets. O parâmetro `data` da função de retorno de chamada tem uma propriedade `Buckets` que contém um array de mapas para representar os buckets. Exiba a lista de buckets ao fazer login no console.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create S3 service object
s3 = new AWS.S3({ apiVersion: "2006-03-01" });
```

```
// Call S3 to list the buckets
s3.listBuckets(function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.Buckets);
  }
});
```

Para executar o exemplo, digite o seguinte na linha de comando.

```
node s3_listbuckets.js
```

Este código de exemplo pode ser encontrado aqui no GitHub.

Criando um Bucket do Amazon S3

Crie um módulo do Node.js com o nome de arquivo `s3_createbucket.js`. Configure o SDK conforme mostrado anteriormente. Crie um objeto de serviço do `AWS.S3`. O módulo precisa de um único argumento de linha de comando para especificar um nome para o novo bucket.

Adicione uma variável para armazenar os parâmetros usados para chamar o método `createBucket` do objeto de serviço do , incluindo o nome do bucket recém-criado. A função de retorno de chamada registra o local do novo bucket no console depois de criar o Amazon S3 com êxito.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create S3 service object
s3 = new AWS.S3({ apiVersion: "2006-03-01" });

// Create the parameters for calling createBucket
var bucketParams = {
  Bucket: process.argv[2],
};

// call S3 to create the bucket
s3.createBucket(bucketParams, function (err, data) {
  if (err) {
```

```
    console.log("Error", err);
  } else {
    console.log("Success", data.Location);
  }
});
```

Para executar o exemplo, digite o seguinte na linha de comando.

```
node s3_createbucket.js BUCKET_NAME
```

Este código de exemplo pode ser encontrado aqui no GitHub.

Fazer upload de um arquivo em um bucket do Amazon S3

Crie um módulo do Node.js com o nome de arquivo `s3_upload.js`. Configure o SDK conforme mostrado anteriormente. Crie um objeto de serviço do `AWS.S3`. O módulo levará dois argumentos de linha de comando: o primeiro para especificar o bucket de destino e o segundo para especificar o arquivo para upload.

Crie uma variável com os parâmetros necessários para chamar o método `upload` do objeto de serviço do Amazon S3. Forneça o nome do bucket de destino no parâmetro `Bucket`. O parâmetro `Key` é definido como o nome do arquivo selecionado, que você pode obter usando o módulo `path` do Node.js. O parâmetro `Body` é definido como o conteúdo do arquivo, que você pode obter usando `createReadStream` do módulo `fs` do Node.js.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create S3 service object
var s3 = new AWS.S3({ apiVersion: "2006-03-01" });

// call S3 to retrieve upload file to specified bucket
var uploadParams = { Bucket: process.argv[2], Key: "", Body: "" };
var file = process.argv[3];

// Configure the file stream and obtain the upload parameters
var fs = require("fs");
var fileStream = fs.createReadStream(file);
fileStream.on("error", function (err) {
  console.log("File Error", err);
```

```
});
uploadParams.Body = fileStream;
var path = require("path");
uploadParams.Key = path.basename(file);

// call S3 to retrieve upload file to specified bucket
s3.upload(uploadParams, function (err, data) {
  if (err) {
    console.log("Error", err);
  }
  if (data) {
    console.log("Upload Success", data.Location);
  }
});
```

Para executar o exemplo, digite o seguinte na linha de comando.

```
node s3_upload.js BUCKET_NAME FILE_NAME
```

Este código de exemplo pode ser encontrado aqui no GitHub.

Listar os objetos em um bucket do Amazon S3

Crie um módulo do Node.js com o nome de arquivo `s3_listobjects.js`. Configure o SDK conforme mostrado anteriormente. Crie um objeto de serviço do `AWS.S3`.

Adicione uma variável para armazenar os parâmetros usados para chamar o método `listObjects` do objeto de serviço do Amazon S3, incluindo o nome do bucket a ser lido. A função de retorno de chamada registra uma lista de objetos (arquivos) ou uma mensagem de falha.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create S3 service object
s3 = new AWS.S3({ apiVersion: "2006-03-01" });

// Create the parameters for calling listObjects
var bucketParams = {
  Bucket: "BUCKET_NAME",
};
```

```
// Call S3 to obtain a list of the objects in the bucket
s3.listObjects(bucketParams, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

Para executar o exemplo, digite o seguinte na linha de comando.

```
node s3_listobjects.js
```

Este código de exemplo pode ser encontrado aqui no GitHub.

Excluir um bucket do Amazon S3

Crie um módulo do Node.js com o nome de arquivo `s3_deletebucket.js`. Configure o SDK conforme mostrado anteriormente. Crie um objeto de serviço do `AWS.S3`.

Adicione uma variável para armazenar os parâmetros usados para chamar o método `createBucket` do objeto de serviço do Amazon S3, incluindo o nome do bucket a ser excluído. Para ser excluído, o bucket deve estar vazio. A função de retorno de chamada registra uma mensagem de êxito ou falha.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create S3 service object
s3 = new AWS.S3({ apiVersion: "2006-03-01" });

// Create params for S3.deleteBucket
var bucketParams = {
  Bucket: "BUCKET_NAME",
};

// Call S3 to delete the bucket
s3.deleteBucket(bucketParams, function (err, data) {
  if (err) {
    console.log("Error", err);
```

```
  } else {
    console.log("Success", data);
  }
});
```

Para executar o exemplo, digite o seguinte na linha de comando.

```
node s3_deletebucket.js
```

Este código de exemplo pode ser encontrado aqui no GitHub.

Configurar buckets do Amazon S3

Este exemplo de código Node.js mostra:

- Como configurar as permissões do CORS (cross-origin resource sharing, compartilhamento de recursos de origem cruzada) para um bucket.

O cenário

Neste exemplo, uma série de módulos do Node.js é usada para listar seus buckets do Amazon S3 e configurar o CORS e o registro dos buckets. Os módulos do Node.js usam o SDK para JavaScript para configurar um bucket do Amazon S3 selecionado usando estes métodos na classe de cliente do Amazon S3:

- `getBucketCors`
- `putBucketCors`

Para obter mais informações sobre a configuração do CORS com um bucket do Amazon S3, consulte Compartilhamento de recursos entre origens (CORS) no Guia do usuário do Amazon Simple Storage Service.

Tarefas de pré-requisito

Para configurar e executar este exemplo, você deve primeiro concluir estas tarefas:

- Instale o Node.js. Para obter mais informações sobre como instalar Node.js, consulte o website de Node.js.
- Crie um arquivo de configurações compartilhado com as credenciais de usuário. Para obter mais informações sobre como fornecer um arquivo de credenciais compartilhadas, consulte Carregar credenciais em Node.js do arquivo de credenciais compartilhado.

Como configurar o SDK

Configure o SDK para JavaScript criando um objeto de configuração global e definindo a região do código. Neste exemplo, a região é definida como `us-west-2`.

```
// Load the SDK for JavaScript
var AWS = require('aws-sdk');
// Set the Region
AWS.config.update({region: 'us-west-2'});
```

Recuperar a configuração do CORS de um bucket

Crie um módulo do Node.js com o nome de arquivo `s3_getcors.js`. O módulo usará um único argumento de linha de comando para especificar o bucket que você deseja para a configuração do CORS. Configure o SDK conforme mostrado anteriormente. Crie um objeto de serviço do `AWS.S3`.

O único parâmetro que você precisa passar é o nome do bucket selecionado ao chamar o método `getBucketCors`. Se o bucket atualmente tiver uma configuração do CORS, essa configuração será retornada pelo Amazon S3 como a propriedade `CORSRules` do parâmetro passado para a função de retorno de chamada.

Se o bucket selecionado ainda não tiver a configuração do CORS, essa informação será retornada para a função de retorno de chamada no parâmetro `error`.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create S3 service object
s3 = new AWS.S3({ apiVersion: "2006-03-01" });

// Set the parameters for S3.getBucketCors
var bucketParams = { Bucket: process.argv[2] };
```

```
// call S3 to retrieve CORS configuration for selected bucket
s3.getBucketCors(bucketParams, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else if (data) {
    console.log("Success", JSON.stringify(data.CORSRules));
  }
});
```

Para executar o exemplo, digite o seguinte na linha de comando.

```
node s3_getcors.js BUCKET_NAME
```

Este código de exemplo pode ser encontrado aqui no GitHub.

Definir a configuração de CORS de um bucket

Crie um módulo do Node.js com o nome de arquivo `s3_setcors.js`. O módulo leva vários argumentos de linha de comando, sendo que o primeiro especifica o bucket cuja configuração do CORS você deseja definir. Outros argumentos enumeram os métodos HTTP (POST, GET, PUT, PATCH, DELETE, POST) que você deseja permitir para o bucket. Configure o SDK como mostrado anteriormente.

Crie um objeto de serviço do `AWS.S3`. Em seguida, crie um objeto JSON para armazenar os valores para a configuração do CORS, conforme exigido pelo método `putBucketCors` do objeto de serviço do `AWS.S3`. Especifique `"Authorization"` para o valor `AllowedHeaders` e `"*"` para o valor `AllowedOrigins`. Inicialmente, defina o valor de `AllowedMethods` como array vazio.

Especifique os métodos permitidos como parâmetros de linha de comando para o módulo Node.js, adicionando cada um dos métodos que correspondem a um dos parâmetros. Adicione a configuração do CORS resultante ao array de configurações contido no parâmetro `CORSRules`. Especifique o bucket que você deseja configurar para o CORS no parâmetro `Bucket`.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create S3 service object
s3 = new AWS.S3({ apiVersion: "2006-03-01" });
```

```
// Create initial parameters JSON for putBucketCors
var thisConfig = {
  AllowedHeaders: ["Authorization"],
  AllowedMethods: [],
  AllowedOrigins: ["*"],
  ExposeHeaders: [],
  MaxAgeSeconds: 3000,
};

// Assemble the list of allowed methods based on command line parameters
var allowedMethods = [];
process.argv.forEach(function (val, index, array) {
  if (val.toUpperCase() === "POST") {
    allowedMethods.push("POST");
  }
  if (val.toUpperCase() === "GET") {
    allowedMethods.push("GET");
  }
  if (val.toUpperCase() === "PUT") {
    allowedMethods.push("PUT");
  }
  if (val.toUpperCase() === "PATCH") {
    allowedMethods.push("PATCH");
  }
  if (val.toUpperCase() === "DELETE") {
    allowedMethods.push("DELETE");
  }
  if (val.toUpperCase() === "HEAD") {
    allowedMethods.push("HEAD");
  }
});

// Copy the array of allowed methods into the config object
thisConfig.AllowedMethods = allowedMethods;
// Create array of configs then add the config object to it
var corsRules = new Array(thisConfig);

// Create CORS params
var corsParams = {
  Bucket: process.argv[2],
  CORSConfiguration: { CORSRules: corsRules },
};
```

```
// set the new CORS configuration on the selected bucket
s3.putBucketCors(corsParams, function (err, data) {
  if (err) {
    // display error message
    console.log("Error", err);
  } else {
    // update the displayed CORS config for the selected bucket
    console.log("Success", data);
  }
});
```

Para executar o exemplo, digite o seguinte na linha de comando, incluindo um ou mais métodos HTTP, como mostrado.

```
node s3_setcors.js BUCKET_NAME get put
```

Este código de exemplo pode ser encontrado aqui no GitHub.

Gerenciar permissões de acesso a buckets do Amazon S3

Este exemplo de código Node.js mostra:

- Como recuperar ou definir a lista de controle de acesso para um bucket do Amazon S3.

O cenário

Neste exemplo, um módulo do Node.js é usado para exibir a lista de controle de acesso (ACL) de um bucket selecionado e aplicar alterações para a ACL de um bucket selecionado. O módulo do Node.js usa o SDK para JavaScript para gerenciar permissões de acesso ao bucket do Amazon S3 usando estes métodos da classe de cliente Amazon S3:

- `getBucketAcl`
- `putBucketAcl`

Para obter mais informações sobre listas de controle de acesso dos buckets do Amazon S3, consulte Gerenciar acesso com ACLs no Manual do usuário do Amazon Simple Storage Service.

Tarefas de pré-requisito

Para configurar e executar este exemplo, você deve primeiro concluir estas tarefas:

- Instale o Node.js. Para obter mais informações sobre como instalar Node.js, consulte o website de Node.js.
- Crie um arquivo de configurações compartilhado com as credenciais de usuário. Para obter mais informações sobre como fornecer um arquivo de credenciais compartilhadas, consulte Carregar credenciais em Node.js do arquivo de credenciais compartilhado.

Como configurar o SDK

Configure o SDK para JavaScript criando um objeto de configuração global e definindo a região do código. Neste exemplo, a região é definida como `us-west-2`.

```
// Load the SDK for JavaScript
var AWS = require('aws-sdk');
// Set the Region
AWS.config.update({region: 'us-west-2'});
```

Recuperar a lista de controle de acesso do bucket atual

Crie um módulo do Node.js com o nome de arquivo `s3_getbucketacl.js`. O módulo usará um único argumento de linha de comando para especificar o bucket que você deseja para a configuração do ACL. Configure o SDK conforme mostrado anteriormente.

Crie um objeto de serviço do `AWS.S3`. O único parâmetro que você precisa passar é o nome do bucket selecionado ao chamar o método `getBucketAcl`. A lista de controle de acesso atual é retornada pelo Amazon S3 no parâmetro `data` passado para a função de retorno de chamada.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create S3 service object
s3 = new AWS.S3({ apiVersion: "2006-03-01" });
```

```
var bucketParams = { Bucket: process.argv[2] };
// call S3 to retrieve policy for selected bucket
s3.getBucketAcl(bucketParams, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else if (data) {
    console.log("Success", data.Grants);
  }
});
```

Para executar o exemplo, digite o seguinte na linha de comando.

```
node s3_getbucketacl.js BUCKET_NAME
```

Este código de exemplo pode ser encontrado aqui no GitHub.

Trabalhar com políticas de bucket do Amazon S3

Este exemplo de código Node.js mostra:

- Como recuperar a política de buckets de um bucket do Amazon S3.
- Como adicionar ou atualizar uma política de buckets de um bucket do Amazon S3.
- Como excluir a política de buckets de um bucket do Amazon S3.

O cenário

Neste exemplo, uma série de módulos do Node.js é usada para recuperar, definir ou excluir uma política de buckets em um bucket do Amazon S3. Os módulos do Node.js usam o SDK para JavaScript para configurar a política de um bucket do Amazon S3 selecionado usando estes métodos na classe de cliente do Amazon S3:

- `getBucketPolicy`
- `putBucketPolicy`

- `deleteBucketPolicy`

Para obter mais informações sobre políticas de bucket dos buckets do Amazon S3, consulte Uso de políticas de bucket e políticas de usuário no Guia do usuário do Amazon Simple Storage Service.

Tarefas de pré-requisito

Para configurar e executar este exemplo, você deve primeiro concluir estas tarefas:

- Instale o Node.js. Para obter mais informações sobre como instalar Node.js, consulte o website de Node.js.
- Crie um arquivo de configurações compartilhado com as credenciais de usuário. Para obter mais informações sobre como fornecer um arquivo de credenciais compartilhadas, consulte Carregar credenciais em Node.js do arquivo de credenciais compartilhado.

Como configurar o SDK

Configure o SDK para JavaScript criando um objeto de configuração global e definindo a região do código. Neste exemplo, a região é definida como `us-west-2`.

```
// Load the SDK for JavaScript
var AWS = require('aws-sdk');
// Set the Region
AWS.config.update({region: 'us-west-2'});
```

Recuperar a política atual de buckets

Crie um módulo do Node.js com o nome de arquivo `s3_getbucketpolicy.js`. O módulo usa um único argumento de linha de comando que especifica o bucket cuja política você deseja. Configure o SDK conforme mostrado anteriormente.

Crie um objeto de serviço do `AWS.S3`. O único parâmetro que você precisa passar é o nome do bucket selecionado ao chamar o método `getBucketPolicy`. Se o bucket já tiver uma política, ela será retornada pelo Amazon S3 no parâmetro `data` transmitido para a função de retorno de chamada.

Se o bucket selecionado ainda não tiver política, essa informação será retornada para a função de retorno de chamada no parâmetro `error`.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create S3 service object
s3 = new AWS.S3({ apiVersion: "2006-03-01" });

var bucketParams = { Bucket: process.argv[2] };
// call S3 to retrieve policy for selected bucket
s3.getBucketPolicy(bucketParams, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else if (data) {
    console.log("Success", data.Policy);
  }
});
```

Para executar o exemplo, digite o seguinte na linha de comando.

```
node s3_getbucketpolicy.js BUCKET_NAME
```

Este código de exemplo pode ser encontrado aqui no GitHub.

Definir uma política de bucket simples

Crie um módulo do Node.js com o nome de arquivo `s3_setbucketpolicy.js`. O módulo usa um único argumento de linha de comando que especifica o bucket cuja política você deseja aplicar. Configure o SDK como mostrado anteriormente.

Crie um objeto de serviço do `AWS.S3`. As políticas do bucket estão especificadas no JSON. Primeiro, crie um objeto JSON contendo todos os valores para especificar a política, exceto para o valor `Resource`, que identifica o bucket.

Formate a string `Resource` exigida pela política, incorporando o nome do bucket selecionado. Insira essa string no objeto JSON. Prepare os parâmetros do método `putBucketPolicy`, incluindo o nome do bucket e a política JSON convertida em um valor de string.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
```

```
AWS.config.update({ region: "REGION" });

// Create S3 service object
s3 = new AWS.S3({ apiVersion: "2006-03-01" });

var readOnlyAnonUserPolicy = {
  Version: "2012-10-17",
  Statement: [
    {
      Sid: "AddPerm",
      Effect: "Allow",
      Principal: "*",
      Action: ["s3:GetObject"],
      Resource: [""],
    },
  ],
};

// create selected bucket resource string for bucket policy
var bucketResource = "arn:aws:s3:::" + process.argv[2] + "/*";
readOnlyAnonUserPolicy.Statement[0].Resource[0] = bucketResource;

// convert policy JSON into string and assign into params
var bucketPolicyParams = {
  Bucket: process.argv[2],
  Policy: JSON.stringify(readOnlyAnonUserPolicy),
};

// set the new policy on the selected bucket
s3.putBucketPolicy(bucketPolicyParams, function (err, data) {
  if (err) {
    // display error message
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

Para executar o exemplo, digite o seguinte na linha de comando.

```
node s3_setbucketpolicy.js BUCKET_NAME
```

Este código de exemplo pode ser encontrado aqui no GitHub.

Excluir uma política de bucket

Crie um módulo do Node.js com o nome de arquivo `s3_deletebucketpolicy.js`. O módulo usa um único argumento de linha de comando que especifica o bucket cuja política você deseja excluir. Configure o SDK como mostrado anteriormente.

Crie um objeto de serviço do `AWS.S3`. O único parâmetro que você precisa passar ao chamar o método `deleteBucketPolicy` é o nome do bucket selecionado.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create S3 service object
s3 = new AWS.S3({ apiVersion: "2006-03-01" });

var bucketParams = { Bucket: process.argv[2] };
// call S3 to delete policy for selected bucket
s3.deleteBucketPolicy(bucketParams, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else if (data) {
    console.log("Success", data);
  }
});
```

Para executar o exemplo, digite o seguinte na linha de comando.

```
node s3_deletebucketpolicy.js BUCKET_NAME
```

Este código de exemplo pode ser encontrado aqui no GitHub.

Usar um bucket do Amazon S3 como um host estático da Web

Este exemplo de código Node.js mostra:

- Como configurar um bucket do Amazon S3 como web host estático.

O cenário

Neste exemplo, uma série de módulos do Node.js é usada para configurar qualquer um dos seus buckets para atuar como um web host estático. Os módulos do Node.js usam o SDK para JavaScript para configurar um bucket do Amazon S3 selecionado usando estes métodos na classe de cliente do Amazon S3:

- `getBucketWebsite`
- `putBucketWebsite`
- `deleteBucketWebsite`

Para obter mais informações sobre como usar um bucket do Amazon S3 como host estático da Web, consulte Como hospedar um site estático no Amazon S3 no Guia de usuário do Amazon Simple Storage Service.

Tarefas de pré-requisito

Para configurar e executar este exemplo, você deve primeiro concluir estas tarefas:

- Instale o Node.js. Para obter mais informações sobre como instalar Node.js, consulte o website de Node.js.
- Crie um arquivo de configurações compartilhado com as credenciais de usuário. Para obter mais informações sobre como fornecer um arquivo de credenciais compartilhadas, consulte Carregar credenciais em Node.js do arquivo de credenciais compartilhado.

Como configurar o SDK

Configure o SDK para JavaScript criando um objeto de configuração global e definindo a região do código. Neste exemplo, a região é definida como `us-west-2`.

```
// Load the SDK for JavaScript
var AWS = require('aws-sdk');
// Set the Region
AWS.config.update({region: 'us-west-2'});
```

Recuperar a configuração de site do bucket atual

Crie um módulo do Node.js com o nome de arquivo `s3_getbucketwebsite.js`. O módulo usa um único argumento de linha de comando que especifica o bucket do qual você deseja a configuração do website. Configure o SDK como mostrado anteriormente.

Crie um objeto de serviço do `AWS.S3`. Crie uma função que recupera a configuração do site do bucket atual para o bucket selecionado na lista de buckets. O único parâmetro que você precisa passar é o nome do bucket selecionado ao chamar o método `getBucketWebsite`. Se o bucket já tiver uma configuração do site, ela será retornada pelo Amazon S3 no parâmetro `data` passado para a função de retorno de chamada.

Se o bucket selecionado ainda não tiver a configuração de site, essa informação será retornada para a função de retorno de chamada no parâmetro `err`.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create S3 service object
s3 = new AWS.S3({ apiVersion: "2006-03-01" });

var bucketParams = { Bucket: process.argv[2] };

// call S3 to retrieve the website configuration for selected bucket
s3.getBucketWebsite(bucketParams, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else if (data) {
    console.log("Success", data);
  }
});
```

Para executar o exemplo, digite o seguinte na linha de comando.

```
node s3_getbucketwebsite.js BUCKET_NAME
```

Este código de exemplo pode ser encontrado aqui no GitHub.

Definir uma configuração de site do bucket

Crie um módulo do Node.js com o nome de arquivo `s3_setbucketwebsite.js`. Configure o SDK conforme mostrado anteriormente. Crie um objeto de serviço do `AWS.S3`.

Crie uma função que aplica uma configuração do site do bucket. A configuração permite que o bucket selecionado sirva como web host estático. As configurações do site são especificadas no JSON. Primeiro, crie um objeto JSON que contenha todos os valores para especificar a configuração do site, exceto para o valor `Key`, que identifica o documento de erro e o valor `Suffix` que identifica o documento de índice.

Insira os valores dos elementos de entrada de texto no objeto JSON. Prepare os parâmetros para o método `putBucketWebsite`, incluindo o nome do bucket e a configuração do site JSON.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create S3 service object
s3 = new AWS.S3({ apiVersion: "2006-03-01" });

// Create JSON for putBucketWebsite parameters
var staticHostParams = {
  Bucket: "",
  WebsiteConfiguration: {
    ErrorDocument: {
      Key: "",
    },
    IndexDocument: {
      Suffix: "",
    },
  },
};

// Insert specified bucket name and index and error documents into params JSON
// from command line arguments
staticHostParams.Bucket = process.argv[2];
staticHostParams.WebsiteConfiguration.IndexDocument.Suffix = process.argv[3];
staticHostParams.WebsiteConfiguration.ErrorDocument.Key = process.argv[4];

// set the new website configuration on the selected bucket
s3.putBucketWebsite(staticHostParams, function (err, data) {
```

```
  if (err) {
    // display error message
    console.log("Error", err);
  } else {
    // update the displayed website configuration for the selected bucket
    console.log("Success", data);
  }
});
```

Para executar o exemplo, digite o seguinte na linha de comando.

```
node s3_setbucketwebsite.js BUCKET_NAME INDEX_PAGE ERROR_PAGE
```

Este código de exemplo pode ser encontrado aqui no GitHub.

Excluir uma configuração do site de um bucket

Crie um módulo do Node.js com o nome de arquivo `s3_deletebucketwebsite.js`. Configure o SDK conforme mostrado anteriormente. Crie um objeto de serviço do `AWS.S3`.

Crie uma função que exclui a configuração de site do bucket selecionado. O único parâmetro que você precisa passar ao chamar o método `deleteBucketWebsite` é o nome do bucket selecionado.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create S3 service object
s3 = new AWS.S3({ apiVersion: "2006-03-01" });

var bucketParams = { Bucket: process.argv[2] };

// call S3 to delete website configuration for selected bucket
s3.deleteBucketWebsite(bucketParams, function (error, data) {
  if (error) {
    console.log("Error", err);
  } else if (data) {
    console.log("Success", data);
  }
});
```

Para executar o exemplo, digite o seguinte na linha de comando.

```
node s3_deletebucketwebsite.js BUCKET_NAME
```

Este código de exemplo pode ser encontrado aqui no GitHub.

Exemplos do Amazon Simple Email Service

O Amazon Simple Email Service (Amazon SES) é um serviço de envio de e-mails baseado na nuvem criado para ajudar profissionais de marketing digital e desenvolvedores de aplicativos a enviar e-mails de marketing, notificações e mensagens transacionais. Ele é um serviço confiável e econômico para empresas de todos os tamanhos que usam e-mail para manter contato com seus clientes.

A API JavaScript do Amazon SES é exposta por meio da classe de cliente `AWS.SES`. Para obter mais informações sobre como usar a classe de cliente do Amazon SES, consulte `Class: AWS.SES` na referência da API.

Tópicos

- Gerenciamento de identidades do Amazon SES
- Lidar com modelos de e-mail no Amazon SES
- Envio de e-mail usando o Amazon SES
- Usar filtros de endereços IP para o recebimento de e-mails do Amazon SES
- Usando regras de recebimento no Amazon SES

Gerenciamento de identidades do Amazon SES

Este exemplo de código Node.js mostra:

- Como verificar endereços de e-mail e domínios usados com o Amazon SES.
- Como atribuir a política do IAM às suas identidades do Amazon SES.
- Como listar todas as identidades do Amazon SES para sua conta da AWS.
- Como excluir identidades usadas com o Amazon SES.

Uma identidade do Amazon SES é um endereço de e-mail ou domínio que o Amazon SES usa para enviar e-mails. O Amazon SES requer que você verifique suas identidades de e-mail, confirmando que você é o proprietário delas e impedindo que outras pessoas as utilizem.

Para obter detalhes sobre como verificar endereços de e-mail e domínios no Amazon SES, consulte Verificar endereços de e-mail e domínios no Amazon SES no Guia do desenvolvedor do Amazon Simple Email Service. Para obter informações sobre a autorização de envio no Amazon SES, consulte Visão geral da autorização de envio do .

O cenário

Neste exemplo, você usa uma série de módulos do Node.js para verificar e gerenciar identidades do Amazon SES. O Node.js usa os módulos do SDK para JavaScript para verificar endereços de e-mail e domínios, usando estes métodos da classe de cliente do `AWS.SES`:

- `listIdentities`
- `deleteIdentity`
- `verifyEmailIdentity`
- `verifyDomainIdentity`

Tarefas de pré-requisito

Para configurar e executar este exemplo, você deve primeiro concluir estas tarefas:

- Instale o Node.js. Para obter mais informações sobre como instalar Node.js, consulte o website de Node.js.
- Crie um arquivo de configurações compartilhado com as credenciais de usuário. Para obter mais informações sobre como fornecer um arquivo de credenciais do JSON, consulte Carregar credenciais em Node.js do arquivo de credenciais compartilhado.

Como configurar o SDK

Configure o SDK para JavaScript criando um objeto de configuração global e definindo a região do código. Neste exemplo, a região é definida como `us-west-2`.

```
// Load the SDK for JavaScript
var AWS = require('aws-sdk');

// Set the Region
AWS.config.update({region: 'us-west-2'});
```

Listar suas identidades

Neste exemplo, use um módulo do Node.js para listar endereços de e-mail e domínios para uso com o Amazon SES. Crie um módulo do Node.js com o nome de arquivo `ses_listidentities.js`. Configure o SDK como mostrado anteriormente.

Crie um objeto para passar o `IdentityType` e outros parâmetros para o método `listIdentities` da classe de cliente `AWS.SES`. Para chamar o método `listIdentities`, crie uma promessa para invocar um objeto de serviço do Amazon SES passando o objeto dos parâmetros.

Depois, lide com `response` no retorno de chamada da promessa. O `data` retornado pela promessa contém um array de identidades de domínio, conforme especificado pelo parâmetro `IdentityType`.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set region
AWS.config.update({ region: "REGION" });

// Create listIdentities params
var params = {
  IdentityType: "Domain",
  MaxItems: 10,
```

```
};

// Create the promise and SES service object
var listIDsPromise = new AWS.SES({ apiVersion: "2010-12-01" })
  .listIdentities(params)
  .promise();

// Handle promise's fulfilled/rejected states
listIDsPromise
  .then(function (data) {
    console.log(data.Identities);
  })
  .catch(function (err) {
    console.error(err, err.stack);
  });
```

Para executar o exemplo, digite o seguinte na linha de comando.

```
node ses_listidentities.js
```

Este código de exemplo pode ser encontrado aqui no GitHub.

Verificar a identidade de um endereço de e-mail

Neste exemplo, use um módulo Node.js para verificar remetentes de e-mail para uso com o Amazon SES. Crie um módulo do Node.js com o nome de arquivo `ses_verifyemailidentity.js`. Configure o SDK como mostrado anteriormente. Para acessar o Amazon SES, crie um objeto de serviço do `AWS.SES`.

Crie um objeto para passar o parâmetro `EmailAddress` para o método `verifyEmailIdentity` da classe de cliente `AWS.SES`. Para chamar o método `verifyEmailIdentity`, crie uma promessa para invocar um objeto de serviço do Amazon SES passando os parâmetros. Depois, lide com `response` no retorno de chamada da promessa.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set region
AWS.config.update({ region: "REGION" });

// Create promise and SES service object
var verifyEmailPromise = new AWS.SES({ apiVersion: "2010-12-01" })
```

```
  .verifyEmailIdentity({ EmailAddress: "ADDRESS@DOMAIN.EXT" })
  .promise();

// Handle promise's fulfilled/rejected states
verifyEmailPromise
  .then(function (data) {
    console.log("Email verification initiated");
  })
  .catch(function (err) {
    console.error(err, err.stack);
  });
```

Para executar o exemplo, digite o seguinte na linha de comando. O domínio é adicionado ao Amazon SES a ser verificado.

```
node ses_verifyemailidentity.js
```

Este código de exemplo pode ser encontrado aqui no GitHub.

Verificar uma identidade de domínio

Neste exemplo, use um módulo do Node.js para verificar domínios de e-mail a serem usados com o Amazon SES. Crie um módulo do Node.js com o nome de arquivo `ses_verifydomainidentity.js`. Configure o SDK como mostrado anteriormente.

Crie um objeto para passar o parâmetro `Domain` para o método `verifyDomainIdentity` da classe de cliente `AWS.SES`. Para chamar o método `verifyDomainIdentity`, crie uma promessa para invocar um objeto de serviço do Amazon SES passando objeto dos parâmetros. Depois, lide com `response` no retorno de chamada da promessa.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set region
AWS.config.update({ region: "REGION" });

// Create the promise and SES service object
var verifyDomainPromise = new AWS.SES({ apiVersion: "2010-12-01" })
  .verifyDomainIdentity({ Domain: "DOMAIN_NAME" })
  .promise();

// Handle promise's fulfilled/rejected states
```

```
verifyDomainPromise
  .then(function (data) {
    console.log("Verification Token: " + data.VerificationToken);
  })
  .catch(function (err) {
    console.error(err, err.stack);
  });
```

Para executar o exemplo, digite o seguinte na linha de comando. O domínio é adicionado ao Amazon SES a ser verificado.

```
node ses_verifydomainidentity.js
```

Este código de exemplo pode ser encontrado aqui no GitHub.

Excluir identidades

Neste exemplo, use um módulo do Node.js para excluir endereços de e-mail ou domínios usados com o Amazon SES. Crie um módulo do Node.js com o nome de arquivo `ses_deleteidentity.js`. Configure o SDK como mostrado anteriormente.

Crie um objeto para passar o parâmetro `Identity` para o método `deleteIdentity` da classe de cliente `AWS.SES`. Para chamar o método `deleteIdentity`, crie uma `request` para invocar um objeto de serviço do Amazon SES passando os parâmetros. Depois, lide com `response` no retorno de chamada da promessa.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set region
AWS.config.update({ region: "REGION" });

// Create the promise and SES service object
var deletePromise = new AWS.SES({ apiVersion: "2010-12-01" })
  .deleteIdentity({ Identity: "DOMAIN_NAME" })
  .promise();

// Handle promise's fulfilled/rejected states
deletePromise
  .then(function (data) {
    console.log("Identity Deleted");
  })
```

```
  .catch(function (err) {
    console.error(err, err.stack);
  });
```

Para executar o exemplo, digite o seguinte na linha de comando.

```
node ses_deleteidentity.js
```

Este código de exemplo pode ser encontrado aqui no GitHub.

Lidar com modelos de e-mail no Amazon SES

Este exemplo de código Node.js mostra:

- Obter uma lista de todos os modelos de e-mail
- Recupere e atualize os modelos de e-mail.
- Crie e exclua os modelos de e-mail.

O Amazon SES permite que você envie mensagens de e-mail personalizadas usando modelos de e-mail. Para obter detalhes sobre como criar e usar modelos de e-mail no Amazon Simple Email Service, consulte Envio de e-mail personalizado usando a API do Amazon SES no Guia do desenvolvedor do Amazon Simple Email Service.

O cenário

Neste exemplo, você usa uma série de módulos do Node.js para trabalhar com modelos de e-mail. Os módulos do Node.js usam o SDK para JavaScript para criar e usar modelos de e-mail com estes métodos da classe de cliente do `AWS.SES`:

- `listTemplates`
- `createTemplate`
- `getTemplate`

- `deleteTemplate`
- `updateTemplate`

Tarefas de pré-requisito

Para configurar e executar este exemplo, você deve primeiro concluir estas tarefas:

- Instale o Node.js. Para obter mais informações sobre como instalar Node.js, consulte o website de Node.js.
- Crie um arquivo de configurações compartilhado com as credenciais de usuário. Para obter mais informações sobre a criação de um arquivo de credenciais, consulte Carregar credenciais em Node.js do arquivo de credenciais compartilhado.

Listar seus modelos de e-mail

Neste exemplo, use um módulo do Node.js para criar um modelo de e-mail a ser usado com o Amazon SES. Crie um módulo do Node.js com o nome de arquivo `ses_listtemplates.js`. Configure o SDK como mostrado anteriormente.

Crie um objeto para passar os parâmetros para o método `listTemplates` da classe de cliente `AWS.SES`. Para chamar o método `listTemplates`, crie uma promessa para invocar um objeto de serviço do Amazon SES passando os parâmetros. Depois, lide com `response` no retorno de chamada da promessa.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the promise and SES service object
var templatePromise = new AWS.SES({ apiVersion: "2010-12-01" })
  .listTemplates({ MaxItems: ITEMS_COUNT })
  .promise();

// Handle promise's fulfilled/rejected states
templatePromise
  .then(function (data) {
    console.log(data);
  })
```

```
  .catch(function (err) {
    console.error(err, err.stack);
  });
```

Para executar o exemplo, digite o seguinte na linha de comando. O Amazon SES retorna a lista de modelos.

```
node ses_listtemplates.js
```

Este código de exemplo pode ser encontrado aqui no GitHub.

Obter um modelo de e-mail

Neste exemplo, use um módulo do Node.js para obter um modelo de e-mail a ser usado com o Amazon SES. Crie um módulo do Node.js com o nome de arquivo `ses_gettemplate.js`. Configure o SDK como mostrado anteriormente.

Crie um objeto para passar o parâmetro `TemplateName` para o método `getTemplate` da classe de cliente `AWS.SES`. Para chamar o método `getTemplate`, crie uma promessa para invocar um objeto de serviço do Amazon SES passando os parâmetros. Depois, lide com `response` no retorno de chamada da promessa.

```
// Load the AWS SDK for Node.js.
var AWS = require("aws-sdk");
// Set the AWS Region.
AWS.config.update({ region: "REGION" });

// Create the promise and Amazon Simple Email Service (Amazon SES) service object.
var templatePromise = new AWS.SES({ apiVersion: "2010-12-01" })
  .getTemplate({ TemplateName: "TEMPLATE_NAME" })
  .promise();

// Handle promise's fulfilled/rejected states
templatePromise
  .then(function (data) {
    console.log(data.Template.SubjectPart);
  })
  .catch(function (err) {
    console.error(err, err.stack);
  });
```

Para executar o exemplo, digite o seguinte na linha de comando. O Amazon SES retorna os detalhes do modelo.

```
node ses_gettemplate.js
```

Este código de exemplo pode ser encontrado aqui no GitHub.

Criar um modelo de e-mail

Neste exemplo, use um módulo do Node.js para criar um modelo de e-mail a ser usado com o Amazon SES. Crie um módulo do Node.js com o nome de arquivo `ses_createtemplate.js`. Configure o SDK como mostrado anteriormente.

Crie um objeto para passar os parâmetros do método `createTemplate` da classe de cliente `AWS.SES`, incluindo `TemplateName`, `HtmlPart`, `SubjectPart` e `TextPart`. Para chamar o método `createTemplate`, crie uma promessa para invocar um objeto de serviço do Amazon SES passando os parâmetros. Depois, lide com `response` no retorno de chamada da promessa.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create createTemplate params
var params = {
  Template: {
    TemplateName: "TEMPLATE_NAME" /* required */,
    HtmlPart: "HTML_CONTENT",
    SubjectPart: "SUBJECT_LINE",
    TextPart: "TEXT_CONTENT",
  },
};

// Create the promise and SES service object
var templatePromise = new AWS.SES({ apiVersion: "2010-12-01" })
  .createTemplate(params)
  .promise();

// Handle promise's fulfilled/rejected states
templatePromise
  .then(function (data) {
```

```
    console.log(data);
  })
  .catch(function (err) {
    console.error(err, err.stack);
  });
```

Para executar o exemplo, digite o seguinte na linha de comando. O modelo é adicionado ao Amazon SES.

```
node ses_createtemplate.js
```

Este código de exemplo pode ser encontrado aqui no GitHub.

Atualizar um modelo de e-mail

Neste exemplo, use um módulo do Node.js para criar um modelo de e-mail a ser usado com o Amazon SES. Crie um módulo do Node.js com o nome de arquivo `ses_updatetemplate.js`. Configure o SDK como mostrado anteriormente.

Crie um objeto para passar os valores do parâmetro `Template` que você deseja atualizar no modelo, com o parâmetro `TemplateName` passado para o método `updateTemplate` da classe de cliente `AWS.SES`. Para chamar o método `updateTemplate`, crie uma promessa para invocar um objeto de serviço do Amazon SES passando os parâmetros. Depois, lide com `response` no retorno de chamada da promessa.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create updateTemplate parameters
var params = {
  Template: {
    TemplateName: "TEMPLATE_NAME" /* required */,
    HtmlPart: "HTML_CONTENT",
    SubjectPart: "SUBJECT_LINE",
    TextPart: "TEXT_CONTENT",
  },
};

// Create the promise and SES service object
```

```
var templatePromise = new AWS.SES({ apiVersion: "2010-12-01" })
  .updateTemplate(params)
  .promise();

// Handle promise's fulfilled/rejected states
templatePromise
  .then(function (data) {
    console.log("Template Updated");
  })
  .catch(function (err) {
    console.error(err, err.stack);
  });
```

Para executar o exemplo, digite o seguinte na linha de comando. O Amazon SES retorna os detalhes do modelo.

```
node ses_updatetemplate.js
```

Este código de exemplo pode ser encontrado aqui no GitHub.

Excluir um modelo de e-mail

Neste exemplo, use um módulo do Node.js para criar um modelo de e-mail a ser usado com o Amazon SES. Crie um módulo do Node.js com o nome de arquivo `ses_deletetemplate.js`. Configure o SDK como mostrado anteriormente.

Crie um objeto para passar o parâmetro `TemplateName` obrigatório para o método `deleteTemplate` da classe de cliente `AWS.SES`. Para chamar o método `deleteTemplate`, crie uma promessa para invocar um objeto de serviço do Amazon SES passando os parâmetros. Depois, lide com `response` no retorno de chamada da promessa.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the promise and SES service object
var templatePromise = new AWS.SES({ apiVersion: "2010-12-01" })
  .deleteTemplate({ TemplateName: "TEMPLATE_NAME" })
  .promise();
```

```
// Handle promise's fulfilled/rejected states
templatePromise
  .then(function (data) {
    console.log("Template Deleted");
  })
  .catch(function (err) {
    console.error(err, err.stack);
  });
```

Para executar o exemplo, digite o seguinte na linha de comando. O Amazon SES retorna os detalhes do modelo.

```
node ses_deletetemplate.js
```

Este código de exemplo pode ser encontrado aqui no GitHub.

Envio de e-mail usando o Amazon SES

Este exemplo de código Node.js mostra:

- Envie uma e-mail de texto ou HTML.
- Envie e-mails usando um modelo de e-mail.
- Envie e-mails em massa usando um modelo de e-mail.

A API do Amazon SES fornece duas maneiras diferentes para você enviar um e-mail, dependendo de quanto controle você deseja ter sobre a composição da mensagem de e-mail: formatada e bruta. Para obter detalhes, consulte Enviar e-mail formatado usando a API do Amazon SES e Enviar e-mail bruto usando a API do Amazon SES.

O cenário

Neste exemplo, você usa uma série de módulos do Node.js para enviar e-mails de várias maneiras. Os módulos do Node.js usam o SDK para JavaScript para criar e usar modelos de e-mail com estes métodos da classe de cliente do `AWS.SES`:

- `sendEmail`
- `sendTemplatedEmail`
- `sendBulkTemplatedEmail`

Tarefas de pré-requisito

- Instale o Node.js. Para obter mais informações sobre como instalar Node.js, consulte o website de Node.js.
- Crie um arquivo de configurações compartilhado com as credenciais de usuário. Para obter mais informações sobre como fornecer um arquivo de credenciais do JSON, consulte Carregar credenciais em Node.js do arquivo de credenciais compartilhado.

Requisitos de envio de mensagens de e-mail

O Amazon SES compõe uma mensagem de e-mail e imediatamente a coloca na fila para envio. Para enviar e-mail usando o método `SES.sendEmail`, sua mensagem deve atender aos seguintes requisitos:

- Você deve enviar a mensagem a partir de um domínio ou endereço de e-mail verificado. Se você tentar enviar um e-mail usando um domínio ou endereço não verificados, a operação resultará no erro `"Email address not verified"`.
- Se sua conta ainda estiver na sandbox do Amazon SES, você só poderá enviar para endereços ou domínios verificados ou para endereços de e-mail associados simulador de caixa postal do Amazon SES. Para obter mais informações, consulte Verificar endereços de e-mail e domínios no Amazon SES no Guia do desenvolvedor do Amazon Simple Email Service.
- O tamanho total da mensagem, incluindo anexos, deve ser menor que 10 MB.
- A mensagem deve incluir pelo menos um endereço de e-mail de destinatário. O endereço do destinatário pode ser um endereço Para:, um endereço CC: ou um endereço CCO:. Se o endereço de e-mail do destinatário for inválido (ou seja, não estiver no formato `UserName@[SubDomain.]Domain.TopLevelDomain`), a mensagem inteira será rejeitada, mesmo se a mensagem contiver outros destinatários válidos.
- A mensagem não pode incluir mais de 50 destinatários nos campos Para:, CC: e CCO:. Se você precisar enviar uma mensagem de e-mail para um público maior, pode dividir a lista de destinatários em grupos de 50 ou menos e chamar o método `sendEmail` várias vezes para enviar a mensagem para cada grupo.

Enviar um e-mail

Neste exemplo, use um módulo do Node.js para enviar e-mail com o Amazon SES. Crie um módulo do Node.js com o nome de arquivo `ses_sendemail.js`. Configure o SDK como mostrado anteriormente.

Crie um objeto para transmitir os valores de parâmetro que definem o e-mail a ser enviado, incluindo endereços do remetente e do destinatário, assunto, corpo do e-mail em texto sem formatação e formato HTML, para o método `sendEmail` da classe de cliente `AWS.SES`. Para chamar o método `sendEmail`, crie uma promessa para invocar um objeto de serviço do Amazon SES passando os parâmetros. Depois, lide com `response` no retorno de chamada da promessa.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create sendEmail params
var params = {
  Destination: {
    /* required */
    CcAddresses: [
      "EMAIL_ADDRESS",
      /* more items */
    ],
    ToAddresses: [
      "EMAIL_ADDRESS",
      /* more items */
    ],
  },
  Message: {
    /* required */
    Body: {
      /* required */
      Html: {
        Charset: "UTF-8",
        Data: "HTML_FORMAT_BODY",
      },
      Text: {
        Charset: "UTF-8",
        Data: "TEXT_FORMAT_BODY",
      },
```

```
    },
    Subject: {
      Charset: "UTF-8",
      Data: "Test email",
    },
  },
  Source: "SENDER_EMAIL_ADDRESS" /* required */,
  ReplyToAddresses: [
    "EMAIL_ADDRESS",
    /* more items */
  ],
};

// Create the promise and SES service object
var sendPromise = new AWS.SES({ apiVersion: "2010-12-01" })
  .sendEmail(params)
  .promise();

// Handle promise's fulfilled/rejected states
sendPromise
  .then(function (data) {
    console.log(data.MessageId);
  })
  .catch(function (err) {
    console.error(err, err.stack);
  });
```

Para executar o exemplo, digite o seguinte na linha de comando. O e-mail é colocado na fila para ser enviado pelo Amazon SES.

```
node ses_sendemail.js
```

Este código de exemplo pode ser encontrado aqui no GitHub.

Enviar um e-mail usando um modelo

Neste exemplo, use um módulo do Node.js para enviar e-mail com o Amazon SES. Crie um módulo do Node.js com o nome de arquivo `ses_sendtemplatedemail.js`. Configure o SDK como mostrado anteriormente.

Crie um objeto para transmitir os valores de parâmetro que definem o e-mail a ser enviado, incluindo endereços do remetente e do destinatário, assunto, corpo do e-mail em texto sem formatação e

formato HTML, para o método `sendTemplatedEmail` da classe de cliente `AWS.SES`. Para chamar o método `sendTemplatedEmail`, crie uma promessa para invocar um objeto de serviço do Amazon SES passando os parâmetros. Depois, lide com `response` no retorno de chamada da promessa.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create sendTemplatedEmail params
var params = {
  Destination: {
    /* required */
    CcAddresses: [
      "EMAIL_ADDRESS",
      /* more CC email addresses */
    ],
    ToAddresses: [
      "EMAIL_ADDRESS",
      /* more To email addresses */
    ],
  },
  Source: "EMAIL_ADDRESS" /* required */,
  Template: "TEMPLATE_NAME" /* required */,
  TemplateData: '{ "REPLACEMENT_TAG_NAME":"REPLACEMENT_VALUE" }' /* required */,
  ReplyToAddresses: ["EMAIL_ADDRESS"],
};

// Create the promise and SES service object
var sendPromise = new AWS.SES({ apiVersion: "2010-12-01" })
  .sendTemplatedEmail(params)
  .promise();

// Handle promise's fulfilled/rejected states
sendPromise
  .then(function (data) {
    console.log(data);
  })
  .catch(function (err) {
    console.error(err, err.stack);
  });
```

Para executar o exemplo, digite o seguinte na linha de comando. O e-mail é colocado na fila para ser enviado pelo Amazon SES.

```
node ses_sendtemplatedemail.js
```

Este código de exemplo pode ser encontrado aqui no GitHub.

Enviar um e-mail em massa usando um modelo

Neste exemplo, use um módulo do Node.js para enviar e-mail com o Amazon SES. Crie um módulo do Node.js com o nome de arquivo `ses_sendbulktemplatedemail.js`. Configure o SDK como mostrado anteriormente.

Crie um objeto para transmitir os valores de parâmetro que definem o e-mail a ser enviado, incluindo endereços do remetente e do destinatário, assunto, corpo do e-mail em texto sem formatação e formato HTML, para o método `sendBulkTemplatedEmail` da classe de cliente `AWS.SES`. Para chamar o método `sendBulkTemplatedEmail`, crie uma promessa para invocar um objeto de serviço do Amazon SES passando os parâmetros. Depois, lide com `response` no retorno de chamada da promessa.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create sendBulkTemplatedEmail params
var params = {
  Destinations: [
    /* required */
    {
      Destination: {
        /* required */
        CcAddresses: [
          "EMAIL_ADDRESS",
          /* more items */
        ],
        ToAddresses: [
          "EMAIL_ADDRESS",
          "EMAIL_ADDRESS",
          /* more items */
        ],
```

```
      },
      ReplacementTemplateData: '{ "REPLACEMENT_TAG_NAME":"REPLACEMENT_VALUE" }',
    },
  ],
  Source: "EMAIL_ADDRESS" /* required */,
  Template: "TEMPLATE_NAME" /* required */,
  DefaultTemplateData: '{ "REPLACEMENT_TAG_NAME":"REPLACEMENT_VALUE" }',
  ReplyToAddresses: ["EMAIL_ADDRESS"],
};

// Create the promise and SES service object
var sendPromise = new AWS.SES({ apiVersion: "2010-12-01" })
  .sendBulkTemplatedEmail(params)
  .promise();

// Handle promise's fulfilled/rejected states
sendPromise
  .then(function (data) {
    console.log(data);
  })
  .catch(function (err) {
    console.log(err, err.stack);
  });
```

Para executar o exemplo, digite o seguinte na linha de comando. O e-mail é colocado na fila para ser enviado pelo Amazon SES.

```
node ses_sendbulktemplatedemail.js
```

Este código de exemplo pode ser encontrado aqui no GitHub.

Usar filtros de endereços IP para o recebimento de e-mails do Amazon SES

Este exemplo de código Node.js mostra:

- Crie filtros de endereços IP para aceitar ou rejeitar e-mails provenientes de um endereço IP ou de um intervalo de endereços IP.

- Liste seus filtros atuais de endereços IP.
- Exclua um filtro de endereço IP.

No Amazon SES, um filtro é uma estrutura de dados formada por nome, intervalo de endereços IP e se deseja-se permitir ou bloquear e-mails dele. Os endereços IP que você deseja bloquear ou permitir são especificados como endereço IP único ou intervalo de endereços IP na notação Classless Inter-Domain Routing (CIDR). Para obter detalhes sobre como o Amazon SES recebe e-mails, consulte Conceitos de recebimento de e-mails do Amazon SES no Guia do desenvolvedor do Amazon Simple Email Service.

O cenário

Neste exemplo, uma série de módulos do Node.js é usada para enviar e-mails de várias maneiras. Os módulos do Node.js usam o SDK para JavaScript para criar e usar modelos de e-mail com estes métodos da classe de cliente do `AWS.SES`:

- `createReceiptFilter`
- `listReceiptFilters`
- `deleteReceiptFilter`

Tarefas de pré-requisito

Para configurar e executar este exemplo, você deve primeiro concluir estas tarefas:

- Instale o Node.js. Para obter mais informações sobre como instalar Node.js, consulte o website de Node.js.
- Crie um arquivo de configurações compartilhado com as credenciais de usuário. Para obter mais informações sobre como fornecer um arquivo de credenciais compartilhadas, consulte Carregar credenciais em Node.js do arquivo de credenciais compartilhado.

Como configurar o SDK

Configure o SDK para JavaScript criando um objeto de configuração global e definindo a região do código. Neste exemplo, a região é definida como `us-west-2`.

```
// Load the SDK for JavaScript
```

```
var AWS = require('aws-sdk');
// Set the Region
AWS.config.update({region: 'us-west-2'});
```

Criar um filtro de endereços IP

Neste exemplo, use um módulo do Node.js para enviar e-mail com o Amazon SES. Crie um módulo do Node.js com o nome de arquivo `ses_createreceiptfilter.js`. Configure o SDK como mostrado anteriormente.

Crie um objeto para passar os valores de parâmetro que definem o filtro IP, incluindo nome do filtro, endereço IP ou um intervalo de endereços para filtrar e se deseja-se permitir ou bloquear o tráfego vindo dos endereços de e-mail filtrados. Para chamar o método `createReceiptFilter`, crie uma promessa para invocar um objeto de serviço do Amazon SES passando os parâmetros. Depois, lide com `response` no retorno de chamada da promessa.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create createReceiptFilter params
var params = {
  Filter: {
    IpFilter: {
      Cidr: "IP_ADDRESS_OR_RANGE",
      Policy: "Allow" | "Block",
    },
    Name: "NAME",
  },
};

// Create the promise and SES service object
var sendPromise = new AWS.SES({ apiVersion: "2010-12-01" })
  .createReceiptFilter(params)
  .promise();

// Handle promise's fulfilled/rejected states
sendPromise
  .then(function (data) {
    console.log(data);
  })
```

```
  .catch(function (err) {
    console.error(err, err.stack);
  });
```

Para executar o exemplo, digite o seguinte na linha de comando. O filtro é criado no Amazon SES.

```
node ses_createreceiptfilter.js
```

Este código de exemplo pode ser encontrado aqui no GitHub.

Listar os filtros dos seus endereços IP

Neste exemplo, use um módulo do Node.js para enviar e-mail com o Amazon SES. Crie um módulo do Node.js com o nome de arquivo `ses_listreceiptfilters.js`. Configure o SDK como mostrado anteriormente.

Crie um objeto de parâmetros vazio. Para chamar o método `listReceiptFilters`, crie uma promessa para invocar um objeto de serviço do Amazon SES passando os parâmetros. Depois, lide com `response` no retorno de chamada da promessa.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the promise and SES service object
var sendPromise = new AWS.SES({ apiVersion: "2010-12-01" })
  .listReceiptFilters({})
  .promise();

// Handle promise's fulfilled/rejected states
sendPromise
  .then(function (data) {
    console.log(data.Filters);
  })
  .catch(function (err) {
    console.error(err, err.stack);
  });
```

Para executar o exemplo, digite o seguinte na linha de comando. O Amazon SES retorna a lista de filtros.

```
node ses_listreceiptfilters.js
```

Este código de exemplo pode ser encontrado aqui no GitHub.

Exclusão de um filtro de endereços IP

Neste exemplo, use um módulo do Node.js para enviar e-mail com o Amazon SES. Crie um módulo do Node.js com o nome de arquivo `ses_deletereceiptfilter.js`. Configure o SDK como mostrado anteriormente.

Crie um objeto para passar o nome do filtro IP a ser excluído. Para chamar o método `deleteReceiptFilter`, crie uma promessa para invocar um objeto de serviço do Amazon SES passando os parâmetros. Depois, lide com `response` no retorno de chamada da promessa.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the promise and SES service object
var sendPromise = new AWS.SES({ apiVersion: "2010-12-01" })
  .deleteReceiptFilter({ FilterName: "NAME" })
  .promise();

// Handle promise's fulfilled/rejected states
sendPromise
  .then(function (data) {
    console.log("IP Filter deleted");
  })
  .catch(function (err) {
    console.error(err, err.stack);
  });
```

Para executar o exemplo, digite o seguinte na linha de comando. O filtro é excluído do Amazon SES.

```
node ses_deletereceiptfilter.js
```

Este código de exemplo pode ser encontrado aqui no GitHub.

Usando regras de recebimento no Amazon SES

Este exemplo de código Node.js mostra:

- Crie e exclua regras de recebimento.
- Organize regras de recebimento em conjuntos de regras de recebimento.

As regras de recebimento no Amazon SES especificam o que fazer com e-mails recebidos para endereços de e-mail ou domínios que você possui. Uma regra de recebimento contém uma condição e uma lista ordenada de ações. Se o destinatário de um e-mail recebido corresponder a um destinatário especificado nas condições da regra de recebimento, o Amazon SES executará as ações especificadas nessa regra de recebimento.

Para usar o Amazon SES como seu receptor de e-mails, você deve ter, pelo menos, um conjunto de regras de recebimento ativo. O conjunto de regras de recebimento é uma coleção ordenada de regras de recebimento que especifica o que o Amazon SES deverá fazer com os e-mails recebidos em seus domínios verificados. Para obter mais informações, consulte Criar regras de recebimento para o recebimento de e-mails do Amazon SES e Criar um conjunto de regras de recebimento para o recebimento de e-mails do Amazon SES no Guia do desenvolvedor do Amazon Simple Email Service.

O cenário

Neste exemplo, uma série de módulos do Node.js é usada para enviar e-mails de várias maneiras. Os módulos do Node.js usam o SDK para JavaScript para criar e usar modelos de e-mail com estes métodos da classe de cliente do `AWS.SES`:

- `createReceiptRule`
- `deleteReceiptRule`
- `createReceiptRuleSet`
- `deleteReceiptRuleSet`

Tarefas de pré-requisito

Para configurar e executar este exemplo, você deve primeiro concluir estas tarefas:

- Instale o Node.js. Para obter mais informações sobre como instalar Node.js, consulte o website de Node.js.
- Crie um arquivo de configurações compartilhado com as credenciais de usuário. Para obter mais informações sobre como fornecer um arquivo de credenciais do JSON, consulte Carregar credenciais em Node.js do arquivo de credenciais compartilhado.

Criar uma regra de recebimento do Amazon S3

Cada regra de recebimento do Amazon SES contém uma lista ordenada de ações. Este exemplo cria uma regra de recebimento com uma ação do Amazon S3, que fornece a mensagem de e-mail para um bucket do Amazon S3. Para obter detalhes sobre as ações das regras de recebimento, consulte Opções de ação no Guia do desenvolvedor do Amazon Simple Email Service.

Para o Amazon SES gravar o e-mail em um bucket do Amazon S3, crie uma política de bucket que conceda a permissão `PutObject` para o Amazon SES. Para obter informações sobre a criação dessa política de bucket, consulte Conceder permissão ao Amazon SES para gravar em seu bucket do Amazon S3 no Guia do desenvolvedor do Amazon Simple Email Service.

Neste exemplo, use um módulo do Node.js para criar uma regra de recebimento no Amazon SES para salvar mensagens recebidas em um bucket do Amazon S3. Crie um módulo do Node.js com o nome de arquivo `ses_createreceiptrule.js`. Configure o SDK como mostrado anteriormente.

Crie um objeto de parâmetros para passar os valores necessários a serem criados para o conjunto de regras de recebimento. Para chamar o método `createReceiptRuleSet`, crie uma promessa para invocar um objeto de serviço do Amazon SES passando os parâmetros. Depois, lide com `response` no retorno de chamada da promessa.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create createReceiptRule params
var params = {
  Rule: {
```

```
    Actions: [
      {
        S3Action: {
          BucketName: "S3_BUCKET_NAME",
          ObjectKeyPrefix: "email",
        },
      },
    ],
    Recipients: [
      "DOMAIN | EMAIL_ADDRESS",
      /* more items */
    ],
    Enabled: true | false,
    Name: "RULE_NAME",
    ScanEnabled: true | false,
    TlsPolicy: "Optional",
  },
  RuleSetName: "RULE_SET_NAME",
};

// Create the promise and SES service object
var newRulePromise = new AWS.SES({ apiVersion: "2010-12-01" })
  .createReceiptRule(params)
  .promise();

// Handle promise's fulfilled/rejected states
newRulePromise
  .then(function (data) {
    console.log("Rule created");
  })
  .catch(function (err) {
    console.error(err, err.stack);
  });
```

Para executar o exemplo, digite o seguinte na linha de comando. O Amazon SES cria a regra de recebimento.

```
node ses_createreceiptrule.js
```

Este código de exemplo pode ser encontrado aqui no GitHub.

Exclusão de uma regra de recebimento

Neste exemplo, use um módulo do Node.js para enviar e-mail com o Amazon SES. Crie um módulo do Node.js com o nome de arquivo `ses_deletereceiptrule.js`. Configure o SDK como mostrado anteriormente.

Crie um objeto de parâmetros para passar o nome que a regra de recebimento deve excluir. Para chamar o método `deleteReceiptRule`, crie uma promessa para invocar um objeto de serviço do Amazon SES passando os parâmetros. Depois, lide com `response` no retorno de chamada da promessa.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create deleteReceiptRule params
var params = {
  RuleName: "RULE_NAME" /* required */,
  RuleSetName: "RULE_SET_NAME" /* required */,
};

// Create the promise and SES service object
var newRulePromise = new AWS.SES({ apiVersion: "2010-12-01" })
  .deleteReceiptRule(params)
  .promise();

// Handle promise's fulfilled/rejected states
newRulePromise
  .then(function (data) {
    console.log("Receipt Rule Deleted");
  })
  .catch(function (err) {
    console.error(err, err.stack);
  });
```

Para executar o exemplo, digite o seguinte na linha de comando. O Amazon SES cria a lista do conjunto de regras de recebimento.

```
node ses_deletereceiptrule.js
```

Este código de exemplo pode ser encontrado aqui no GitHub.

Criação de um conjunto de regras de recebimento

Neste exemplo, use um módulo do Node.js para enviar e-mail com o Amazon SES. Crie um módulo do Node.js com o nome de arquivo `ses_createreceiptruleset.js`. Configure o SDK como mostrado anteriormente.

Crie um objeto de parâmetros para passar o nome do novo conjunto de regras de recebimento. Para chamar o método `createReceiptRuleSet`, crie uma promessa para invocar um objeto de serviço do Amazon SES passando os parâmetros. Depois, lide com `response` no retorno de chamada da promessa.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the promise and SES service object
var newRulePromise = new AWS.SES({ apiVersion: "2010-12-01" })
  .createReceiptRuleSet({ RuleSetName: "NAME" })
  .promise();

// Handle promise's fulfilled/rejected states
newRulePromise
  .then(function (data) {
    console.log(data);
  })
  .catch(function (err) {
    console.error(err, err.stack);
  });
```

Para executar o exemplo, digite o seguinte na linha de comando. O Amazon SES cria a lista do conjunto de regras de recebimento.

```
node ses_createreceiptruleset.js
```

Este código de exemplo pode ser encontrado aqui no GitHub.

Exclusão de um conjunto de regras de recebimento

Neste exemplo, use um módulo do Node.js para enviar e-mail com o Amazon SES. Crie um módulo do Node.js com o nome de arquivo `ses_deletereceiptruleset.js`. Configure o SDK como mostrado anteriormente.

Crie um objeto para passar o nome que o conjunto de regra do recebimento deve excluir. Para chamar o método `deleeReceiptRuleSet`, crie uma promessa para invocar um objeto de serviço do Amazon SES passando os parâmetros. Depois, lide com `response` no retorno de chamada da promessa.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the promise and SES service object
var newRulePromise = new AWS.SES({ apiVersion: "2010-12-01" })
  .deleteReceiptRuleSet({ RuleSetName: "NAME" })
  .promise();

// Handle promise's fulfilled/rejected states
newRulePromise
  .then(function (data) {
    console.log(data);
  })
  .catch(function (err) {
    console.error(err, err.stack);
  });
```

Para executar o exemplo, digite o seguinte na linha de comando. O Amazon SES cria a lista do conjunto de regras de recebimento.

```
node ses_deletereceiptruleset.js
```

Este código de exemplo pode ser encontrado aqui no GitHub.

Exemplos do Amazon Simple Notification Service

O Amazon Simple Notification Service (Amazon SNS) é um serviço da Web que coordena e gerencia a entrega ou o envio de mensagens para endpoints ou clientes inscritos.

No Amazon SNS, há dois tipos de clientes – os publicadores e os assinantes – também chamados de produtores e consumidores.

Os editores se comunicam de maneira assíncrona com os inscritos produzindo e enviando uma mensagem para um tópico, que é um canal de comunicação e um ponto de acesso lógico. Os inscritos (isto é, os servidores da web, os endereços de e-mail, as filas do Amazon SQS, as funções do Lambda) consomem ou recebem a mensagem ou a notificação em um dos protocolos compatíveis (Amazon SQS , HTTP/S, e-mail, SMS,AWS Lambda) quando estão inscritos no tópico.

A API JavaScript do Amazon SNS é exposta por meio do `Class: AWS.SNS`.

Tópicos

- Gerenciamento de tópicos no Amazon SNS
- Publicação de mensagens no Amazon SNS
- Gerenciamento de assinaturas no Amazon SNS
- Enviar mensagens SMS com o Amazon SNS

Gerenciamento de tópicos no Amazon SNS

Este exemplo de código Node.js mostra:

- Como criar tópicos no Amazon SNS para os quais você pode publicar notificações.
- Como excluir tópicos criados no Amazon SNS.

- Como obter uma lista de tópicos disponíveis.
- Como obter e definir atributos de tópicos.

O cenário

Neste exemplo, você usa uma série de módulos do Node.js para criar, listar e excluir tópicos do Amazon SNS e para lidar com atributos de tópicos. Os módulos Node.js usam o SDK para JavaScript para gerenciar tópicos usando estes métodos da classe de cliente `AWS.SNS`:

- `createTopic`
- `listTopics`
- `deleteTopic`
- `getTopicAttributes`
- `setTopicAttributes`

Tarefas de pré-requisito

Para configurar e executar este exemplo, você deve primeiro concluir estas tarefas:

- Instale o Node.js. Para obter mais informações sobre como instalar Node.js, consulte o website de Node.js.
- Crie um arquivo de configurações compartilhado com as credenciais de usuário. Para obter mais informações sobre como fornecer um arquivo de credenciais do JSON, consulte Carregar credenciais em Node.js do arquivo de credenciais compartilhado.

Criar um tópico

Neste exemplo, use um módulo do Node.js para criar um tópico do Amazon SNS. Crie um módulo do Node.js com o nome de arquivo `sns_createtopic.js`. Configure o SDK como mostrado anteriormente.

Crie um objeto para passar o `Name` para o novo tópico para o método `createTopic` da classe de cliente `AWS.SNS`. Para chamar o método `createTopic`, crie uma promessa para invocar um objeto de serviço do Amazon SNS passando o objeto dos parâmetros. Depois, lide com `response` no retorno de chamada da promessa. O `data` retornado pela promessa contém o ARN do tópico.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set region
AWS.config.update({ region: "REGION" });

// Create promise and SNS service object
var createTopicPromise = new AWS.SNS({ apiVersion: "2010-03-31" })
  .createTopic({ Name: "TOPIC_NAME" })
  .promise();

// Handle promise's fulfilled/rejected states
createTopicPromise
  .then(function (data) {
    console.log("Topic ARN is " + data.TopicArn);
  })
  .catch(function (err) {
    console.error(err, err.stack);
  });
```

Para executar o exemplo, digite o seguinte na linha de comando.

```
node sns_createtopic.js
```

Este código de exemplo pode ser encontrado aqui no GitHub.

Listar os tópicos do

Neste exemplo, use um módulo do Node.js para listar todos os tópicos do Amazon SNS. Crie um módulo do Node.js com o nome de arquivo `sns_listtopics.js`. Configure o SDK como mostrado anteriormente.

Crie um objeto vazio para passar para o método `listTopics` da classe de cliente `AWS.SNS`. Para chamar o método `listTopics`, crie uma promessa para invocar um objeto de serviço do Amazon SNS passando o objeto dos parâmetros. Depois, lide com `response` no retorno de chamada da promessa. O `data` retornado pela promessa contém um array dos ARNs do seu tópico.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set region
AWS.config.update({ region: "REGION" });
```

```
// Create promise and SNS service object
var listTopicsPromise = new AWS.SNS({ apiVersion: "2010-03-31" })
  .listTopics({})
  .promise();

// Handle promise's fulfilled/rejected states
listTopicsPromise
  .then(function (data) {
    console.log(data.Topics);
  })
  .catch(function (err) {
    console.error(err, err.stack);
  });
```

Para executar o exemplo, digite o seguinte na linha de comando.

```
node sns_listtopics.js
```

Este código de exemplo pode ser encontrado aqui no GitHub.

Exclusão de um tópico

Neste exemplo, use um módulo do Node.js para excluir um tópico do Amazon SNS . Crie um módulo do Node.js com o nome de arquivo `sns_deletetopic.js`. Configure o SDK como mostrado anteriormente.

Crie um objeto contendo o `TopicArn` do tópico para excluir e passar para o método `deleteTopic` da classe de cliente `AWS.SNS`. Para chamar o método `deleteTopic`, crie uma promessa para invocar um objeto de serviço do Amazon SNS passando o objeto dos parâmetros. Depois, lide com `response` no retorno de chamada da promessa.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set region
AWS.config.update({ region: "REGION" });

// Create promise and SNS service object
var deleteTopicPromise = new AWS.SNS({ apiVersion: "2010-03-31" })
  .deleteTopic({ TopicArn: "TOPIC_ARN" })
  .promise();
```

```
// Handle promise's fulfilled/rejected states
deleteTopicPromise
  .then(function (data) {
    console.log("Topic Deleted");
  })
  .catch(function (err) {
    console.error(err, err.stack);
  });
```

Para executar o exemplo, digite o seguinte na linha de comando.

```
node sns_deletetopic.js
```

Este código de exemplo pode ser encontrado aqui no GitHub.

Obter atributos do tópico

Neste exemplo, use um módulo do Node.js para recuperar atributos de um tópico do Amazon SNS. Crie um módulo do Node.js com o nome de arquivo `sns_gettopicattributes.js`. Configure o SDK como mostrado anteriormente.

Crie um objeto contendo o `TopicArn` de um tópico para excluir e passar para o método `getTopicAttributes` da classe de cliente `AWS.SNS`. Para chamar o método `getTopicAttributes`, crie uma promessa para invocar um objeto de serviço do Amazon SNS passando o objeto dos parâmetros. Depois, lide com `response` no retorno de chamada da promessa.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set region
AWS.config.update({ region: "REGION" });

// Create promise and SNS service object
var getTopicAttribsPromise = new AWS.SNS({ apiVersion: "2010-03-31" })
  .getTopicAttributes({ TopicArn: "TOPIC_ARN" })
  .promise();

// Handle promise's fulfilled/rejected states
getTopicAttribsPromise
  .then(function (data) {
    console.log(data);
```

```
  })
  .catch(function (err) {
    console.error(err, err.stack);
  });
```

Para executar o exemplo, digite o seguinte na linha de comando.

```
node sns_gettopicattributes.js
```

Este código de exemplo pode ser encontrado aqui no GitHub.

Definir atributos do tópico

Neste exemplo, use um módulo do Node.js para definir os atributos mutáveis de um tópico do Amazon SNS. Crie um módulo do Node.js com o nome de arquivo `sns_settopicattributes.js`. Configure o SDK como mostrado anteriormente.

Crie um objeto contendo os parâmetros para a atualização do atributo, incluindo o `TopicArn` do tópico cujos atributos você deseja definir, o nome do atributo a ser definido e o novo valor desse atributo. É possível definir apenas os atributos `Policy`, `DisplayName` e `DeliveryPolicy`. Passe os parâmetros para o método `setTopicAttributes` da classe de cliente `AWS.SNS`. Para chamar o método `setTopicAttributes`, crie uma promessa para invocar um objeto de serviço do Amazon SNS passando o objeto dos parâmetros. Depois, lide com `response` no retorno de chamada da promessa.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set region
AWS.config.update({ region: "REGION" });

// Create setTopicAttributes parameters
var params = {
  AttributeName: "ATTRIBUTE_NAME" /* required */,
  TopicArn: "TOPIC_ARN" /* required */,
  AttributeValue: "NEW_ATTRIBUTE_VALUE",
};

// Create promise and SNS service object
var setTopicAttribsPromise = new AWS.SNS({ apiVersion: "2010-03-31" })
  .setTopicAttributes(params)
```

```
  .promise();

// Handle promise's fulfilled/rejected states
setTopicAttribsPromise
  .then(function (data) {
    console.log(data);
  })
  .catch(function (err) {
    console.error(err, err.stack);
  });
```

Para executar o exemplo, digite o seguinte na linha de comando.

```
node sns_settopicattributes.js
```

Este código de exemplo pode ser encontrado aqui no GitHub.

Publicação de mensagens no Amazon SNS

Este exemplo de código Node.js mostra:

- Como publicar mensagens em um tópico do Amazon SNS.

O cenário

Neste exemplo, você usa uma série de módulos do Node.js para publicar mensagens do Amazon SNS nos endpoints do tópico, e-mails ou números de telefone. Os módulos do Node.js usam o SDK para JavaScript para enviar mensagens usando este método da classe de cliente `AWS.SNS`:

- `publish`

Tarefas de pré-requisito

Para configurar e executar este exemplo, você deve primeiro concluir estas tarefas:

- Instale o Node.js. Para obter mais informações sobre como instalar Node.js, consulte o website de Node.js.
- Crie um arquivo de configurações compartilhado com as credenciais de usuário. Para obter mais informações sobre como fornecer um arquivo de credenciais do JSON, consulte Carregar credenciais em Node.js do arquivo de credenciais compartilhado.

Publicar uma mensagem em um tópico do Amazon SNS

Neste exemplo, use um módulo do Node.js para publicar uma mensagem em um tópico do Amazon SNS. Crie um módulo do Node.js com o nome de arquivo `sns_publishtotopic.js`. Configure o SDK como mostrado anteriormente.

Crie um objeto contendo os parâmetros para publicar uma mensagem, incluindo o texto da mensagem e o ARN do tópico do Amazon SNS. Para obter detalhes sobre os atributos de SMS disponíveis, consulte SetSMSAttributes.

Passe os parâmetros para o método `publish` da classe de cliente `AWS.SNS`. Crie uma promessa para invocar um objeto de serviço do Amazon SNS, passando o objeto dos parâmetros. Depois, lide com a resposta no retorno de chamada da promessa.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set region
AWS.config.update({ region: "REGION" });

// Create publish parameters
var params = {
  Message: "MESSAGE_TEXT" /* required */,
  TopicArn: "TOPIC_ARN",
};

// Create promise and SNS service object
var publishTextPromise = new AWS.SNS({ apiVersion: "2010-03-31" })
  .publish(params)
  .promise();

// Handle promise's fulfilled/rejected states
publishTextPromise
  .then(function (data) {
    console.log(
      `Message ${params.Message} sent to the topic ${params.TopicArn}`
```

```
    );
    console.log("MessageID is " + data.MessageId);
  })
  .catch(function (err) {
    console.error(err, err.stack);
  });
```

Para executar o exemplo, digite o seguinte na linha de comando.

```
node sns_publishtotopic.js
```

Este código de exemplo pode ser encontrado aqui no GitHub.

Gerenciamento de assinaturas no Amazon SNS

Este exemplo de código Node.js mostra:

- Como listar todas as assinaturas para um tópico do Amazon SNS.
- Como inscrever um endereço de e-mail, um endpoint de aplicativo ou uma função do AWS Lambda em um tópico do Amazon SNS.
- Como cancelar a assinatura dos tópicos do Amazon SNS.

O cenário

Neste exemplo, você usa uma série de módulos do Node.js para publicar mensagens de notificação em tópicos do Amazon SNS. Os módulos Node.js usam o SDK para JavaScript para gerenciar tópicos usando estes métodos da classe de cliente `AWS.SNS`:

- `subscribe`
- `confirmSubscription`
- `listSubscriptionsByTopic`
- `unsubscribe`

Tarefas de pré-requisito

Para configurar e executar este exemplo, você deve primeiro concluir estas tarefas:

- Instale o Node.js. Para obter mais informações sobre como instalar Node.js, consulte o website de Node.js.
- Crie um arquivo de configurações compartilhado com as credenciais de usuário. Para obter mais informações sobre como fornecer um arquivo de credenciais do JSON, consulte Carregar credenciais em Node.js do arquivo de credenciais compartilhado.

Listar assinaturas em um tópico

Neste exemplo, use um módulo do Node.js para listar todas as inscrições em um tópico do Amazon SNS. Crie um módulo do Node.js com o nome de arquivo `sns_listsubscriptions.js`. Configure o SDK como mostrado anteriormente.

Crie um objeto contendo o parâmetro `TopicArn` para o tópico cujas assinaturas você deseja listar. Passe os parâmetros para o método `listSubscriptionsByTopic` da classe de cliente `AWS.SNS`. Para chamar o método `listSubscriptionsByTopic`, crie uma promessa para invocar um objeto de serviço do Amazon SNS passando o objeto dos parâmetros. Depois, lide com `response` no retorno de chamada da promessa.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set region
AWS.config.update({ region: "REGION" });

const params = {
  TopicArn: "TOPIC_ARN",
};

// Create promise and SNS service object
var subslistPromise = new AWS.SNS({ apiVersion: "2010-03-31" })
  .listSubscriptionsByTopic(params)
  .promise();

// Handle promise's fulfilled/rejected states
subslistPromise
  .then(function (data) {
    console.log(data);
```

```
  })
  .catch(function (err) {
    console.error(err, err.stack);
  });
```

Para executar o exemplo, digite o seguinte na linha de comando.

```
node sns_listsubscriptions.js
```

Este código de exemplo pode ser encontrado aqui no GitHub.

Inscrever um endereço de e-mail em um tópico

Neste exemplo, use um módulo do Node.js para inscrever um endereço de e-mail de forma que ele receba mensagens de e-mail SMTP de um tópico do Amazon SNS. Crie um módulo do Node.js com o nome de arquivo `sns_subscribeemail.js`. Configure o SDK como mostrado anteriormente.

Crie um objeto que contém o parâmetro `Protocol` para especificar o protocolo `email`, o `TopicArn` do tópico a ser assinado e um endereço de e-mail como `Endpoint` da mensagem. Passe os parâmetros para o método `subscribe` da classe de cliente `AWS.SNS`. Você pode usar o método `subscribe` para assinar vários endpoints diferentes para um tópico do Amazon SNS, dependendo dos valores usados para os parâmetros passados, como outros exemplos neste tópico mostrarão.

Para chamar o método `subscribe`, crie uma promessa para invocar um objeto de serviço do Amazon SNS passando o objeto dos parâmetros. Depois, lide com `response` no retorno de chamada da promessa.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set region
AWS.config.update({ region: "REGION" });

// Create subscribe/email parameters
var params = {
  Protocol: "EMAIL" /* required */,
  TopicArn: "TOPIC_ARN" /* required */,
  Endpoint: "EMAIL_ADDRESS",
};

// Create promise and SNS service object
var subscribePromise = new AWS.SNS({ apiVersion: "2010-03-31" })
```

```
  .subscribe(params)
  .promise();

// Handle promise's fulfilled/rejected states
subscribePromise
  .then(function (data) {
    console.log("Subscription ARN is " + data.SubscriptionArn);
  })
  .catch(function (err) {
    console.error(err, err.stack);
  });
```

Para executar o exemplo, digite o seguinte na linha de comando.

```
node sns_subscribeemail.js
```

Este código de exemplo pode ser encontrado aqui no GitHub.

Inscrever um endpoint de aplicativo em um tópico

Neste exemplo, use um módulo do Node.js para inscrever um endpoint de aplicativo móvel para receber notificações de um tópico do Amazon SNS. Crie um módulo do Node.js com o nome de arquivo `sns_subscribeapp.js`. Configure o SDK como mostrado anteriormente.

Crie um objeto contendo o parâmetro `Protocol` para especificar o protocolo `application`, o `TopicArn` para o tópico no qual será feita a inscrição e o ARN do endpoint de um aplicativo móvel para o parâmetro `Endpoint`. Passe os parâmetros para o método `subscribe` da classe de cliente `AWS.SNS`.

Para chamar o método `subscribe`, crie uma promessa para invocar um objeto de serviço do Amazon SNS passando o objeto dos parâmetros. Depois, lide com `response` no retorno de chamada da promessa.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set region
AWS.config.update({ region: "REGION" });

// Create subscribe/email parameters
var params = {
  Protocol: "application" /* required */,
```

```
  TopicArn: "TOPIC_ARN" /* required */,
  Endpoint: "MOBILE_ENDPOINT_ARN",
};

// Create promise and SNS service object
var subscribePromise = new AWS.SNS({ apiVersion: "2010-03-31" })
  .subscribe(params)
  .promise();

// Handle promise's fulfilled/rejected states
subscribePromise
  .then(function (data) {
    console.log("Subscription ARN is " + data.SubscriptionArn);
  })
  .catch(function (err) {
    console.error(err, err.stack);
  });
```

Para executar o exemplo, digite o seguinte na linha de comando.

```
node sns_subscribeapp.js
```

Este código de exemplo pode ser encontrado aqui no GitHub.

Inscrever uma função do Lambda em um tópico

Neste exemplo, use um módulo do Node.js para inscrever uma função do para receber notificações de um tópico do Amazon SNS. Crie um módulo do Node.js com o nome de arquivo `sns_subscribelambda.js`. Configure o SDK como mostrado anteriormente.

Crie um objeto contendo o parâmetro `Protocol`, especificando o protocolo `lambda`, o `TopicArn` do tópico de inscrição e o ARN de uma função AWS Lambda como o parâmetro do `Endpoint`. Passe os parâmetros para o método `subscribe` da classe de cliente `AWS.SNS`.

Para chamar o método `subscribe`, crie uma promessa para invocar um objeto de serviço do Amazon SNS passando o objeto dos parâmetros. Depois, lide com `response` no retorno de chamada da promessa.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set region
```

```
AWS.config.update({ region: "REGION" });

// Create subscribe/email parameters
var params = {
  Protocol: "lambda" /* required */,
  TopicArn: "TOPIC_ARN" /* required */,
  Endpoint: "LAMBDA_FUNCTION_ARN",
};

// Create promise and SNS service object
var subscribePromise = new AWS.SNS({ apiVersion: "2010-03-31" })
  .subscribe(params)
  .promise();

// Handle promise's fulfilled/rejected states
subscribePromise
  .then(function (data) {
    console.log("Subscription ARN is " + data.SubscriptionArn);
  })
  .catch(function (err) {
    console.error(err, err.stack);
  });
```

Para executar o exemplo, digite o seguinte na linha de comando.

```
node sns_subscribelambda.js
```

Este código de exemplo pode ser encontrado aqui no GitHub.

Cancelar a inscrição em um tópico

Neste exemplo, use um módulo do Node.js para cancelar a inscrição a um tópico do Amazon SNS. Crie um módulo do Node.js com o nome de arquivo `sns_unsubscribe.js`. Configure o SDK como mostrado anteriormente.

Crie um objeto que contém o parâmetro `SubscriptionArn`, especificando o ARN da inscrição para cancelar a assinatura. Passe os parâmetros para o método `unsubscribe` da classe de cliente `AWS.SNS`.

Para chamar o método `unsubscribe`, crie uma promessa para invocar um objeto de serviço do Amazon SNS passando o objeto dos parâmetros. Depois, lide com `response` no retorno de chamada da promessa.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set region
AWS.config.update({ region: "REGION" });

// Create promise and SNS service object
var subscribePromise = new AWS.SNS({ apiVersion: "2010-03-31" })
  .unsubscribe({ SubscriptionArn: TOPIC_SUBSCRIPTION_ARN })
  .promise();

// Handle promise's fulfilled/rejected states
subscribePromise
  .then(function (data) {
    console.log(data);
  })
  .catch(function (err) {
    console.error(err, err.stack);
  });
```

Para executar o exemplo, digite o seguinte na linha de comando.

```
node sns_unsubscribe.js
```

Este código de exemplo pode ser encontrado aqui no GitHub.

Enviar mensagens SMS com o Amazon SNS

Este exemplo de código Node.js mostra:

- Como obter e definir as preferências de mensagens SMS para Amazon SNS.
- Como verificar um número de telefone para definir se ele não permite o recebimento de mensagens SMS.
- Como obter uma lista de números de telefone que cancelaram o recebimento de mensagens SMS.
- Como enviar uma mensagem SMS.

O cenário

Você pode usar o Amazon SNS para enviar mensagens de texto ou mensagens SMS para dispositivos habilitados para SMS. Você pode enviar uma mensagem diretamente para um número de telefone, ou enviar uma mensagem para vários números de telefone de uma só vez inscrevendo esses números em um tópico e enviando sua mensagem para o tópico.

Neste exemplo, você usa uma série de módulos do Node.js para publicar mensagens de texto SMS do Amazon SNS em dispositivos habilitados para SMS. Os módulos do Node.js usam o SDK para JavaScript para publicar mensagens SMS usando estes métodos da classe de cliente :

- `getSMSAttributes`
- `setSMSAttributes`
- `checkIfPhoneNumberIsOptedOut`
- `listPhoneNumbersOptedOut`
- `publish`

Tarefas de pré-requisito

Para configurar e executar este exemplo, você deve primeiro concluir estas tarefas:

- Instale o Node.js. Para obter mais informações sobre como instalar Node.js, consulte o website de Node.js.
- Crie um arquivo de configurações compartilhado com as credenciais de usuário. Para obter mais informações sobre como fornecer um arquivo de credenciais do JSON, consulte Carregar credenciais em Node.js do arquivo de credenciais compartilhado.

Obter atributos do SMS

Use o Amazon SNS para especificar as preferências para o uso de mensagens SMS, por exemplo, como suas entregas serão otimizadas (para fins de custo ou confiabilidade), o limite de gastos mensais, como as entregas de mensagens serão registradas e a assinatura em relatórios diários de uso de SMS. Essas preferências são recuperadas e definidas como atributos de SMS para Amazon SNS.

Neste exemplo, use um módulo do Node.js para obter os atributos de SMS atuais no Amazon SNS. Crie um módulo do Node.js com o nome de arquivo `sns_getsmstype.js`. Configure o SDK como

mostrado anteriormente. Crie um objeto contendo os parâmetros para obter atributos de SMS, incluindo os nomes dos atributos individuais a serem obtidos. Para obter detalhes sobre os atributos de SMS disponíveis, consulte SetSMSAttributes na Referência de API do Amazon Simple Notification Service.

Esse exemplo usa o atributo `DefaultSMSType`, que controla se serão enviadas mensagens SMS como `Promotional`, o que otimiza a entrega de mensagens para gerar custos mais baixos, ou como `Transactional`, que otimiza a entrega de mensagens para gerar a mais alta confiabilidade. Passe os parâmetros para o método `setTopicAttributes` da classe de cliente `AWS.SNS`. Para chamar o método `getSMSAttributes`, crie uma promessa para invocar um objeto de serviço do Amazon SNS passando o objeto dos parâmetros. Depois, lide com `response` no retorno de chamada da promessa.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set region
AWS.config.update({ region: "REGION" });

// Create SMS Attribute parameter you want to get
var params = {
  attributes: [
    "DefaultSMSType",
    "ATTRIBUTE_NAME",
    /* more items */
  ],
};

// Create promise and SNS service object
var getSMSTypePromise = new AWS.SNS({ apiVersion: "2010-03-31" })
  .getSMSAttributes(params)
  .promise();

// Handle promise's fulfilled/rejected states
getSMSTypePromise
  .then(function (data) {
    console.log(data);
  })
  .catch(function (err) {
    console.error(err, err.stack);
  });
```

Para executar o exemplo, digite o seguinte na linha de comando.

```
node sns_getsmstype.js
```

Este código de exemplo pode ser encontrado aqui no GitHub.

Definir atributos do SMS

Neste exemplo, use um módulo do Node.js para obter os atributos de SMS atuais no Amazon SNS. Crie um módulo do Node.js com o nome de arquivo `sns_setsmstype.js`. Configure o SDK como mostrado anteriormente. Crie um objeto contendo os parâmetros para definir atributos de SMS, incluindo os nomes dos atributos individuais a serem definidos e os valores para cada um. Para obter detalhes sobre os atributos de SMS disponíveis, consulte SetSMSAttributes na Referência de API do Amazon Simple Notification Service.

Este exemplo define o atributo `DefaultSMSType` para `Transactional`, que otimiza a entrega de mensagens para gerar a mais alta confiabilidade. Passe os parâmetros para o método `setTopicAttributes` da classe de cliente `AWS.SNS`. Para chamar o método `getSMSAttributes`, crie uma promessa para invocar um objeto de serviço do Amazon SNS passando o objeto dos parâmetros. Depois, lide com `response` no retorno de chamada da promessa.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set region
AWS.config.update({ region: "REGION" });

// Create SMS Attribute parameters
var params = {
  attributes: {
    /* required */
    DefaultSMSType: "Transactional" /* highest reliability */,
    //'DefaultSMSType': 'Promotional' /* lowest cost */
  },
};

// Create promise and SNS service object
var setSMSTypePromise = new AWS.SNS({ apiVersion: "2010-03-31" })
  .setSMSAttributes(params)
  .promise();
```

```
// Handle promise's fulfilled/rejected states
setSMSTypePromise
  .then(function (data) {
    console.log(data);
  })
  .catch(function (err) {
    console.error(err, err.stack);
  });
```

Para executar o exemplo, digite o seguinte na linha de comando.

```
node sns_setsmstype.js
```

Este código de exemplo pode ser encontrado aqui no GitHub.

Verificar se um número de telefone cancelou o recebimento

Neste exemplo, use um módulo do Node.js para verificar um número de telefone e determinar se ele cancelou o recebimento de mensagens SMS. Crie um módulo do Node.js com o nome de arquivo `sns_checkphoneoptout.js`. Configure o SDK como mostrado anteriormente. Crie um objeto contendo o número de telefone a ser verificado como parâmetro.

Este exemplo define o parâmetro `PhoneNumber` para especificar o número de telefone a ser verificado. Passe o objeto para o método `checkIfPhoneNumberIsOptedOut` da classe de cliente `AWS.SNS`. Para chamar o método `checkIfPhoneNumberIsOptedOut`, crie uma promessa para invocar um objeto de serviço do Amazon SNS passando o objeto dos parâmetros. Depois, lide com `response` no retorno de chamada da promessa.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set region
AWS.config.update({ region: "REGION" });

// Create promise and SNS service object
var phonenumPromise = new AWS.SNS({ apiVersion: "2010-03-31" })
  .checkIfPhoneNumberIsOptedOut({ phoneNumber: "PHONE_NUMBER" })
  .promise();

// Handle promise's fulfilled/rejected states
phonenumPromise
  .then(function (data) {
```

```
    console.log("Phone Opt Out is " + data.isOptedOut);
  })
  .catch(function (err) {
    console.error(err, err.stack);
  });
```

Para executar o exemplo, digite o seguinte na linha de comando.

```
node sns_checkphoneoptout.js
```

Este código de exemplo pode ser encontrado aqui no GitHub.

Listar números de telefone que cancelaram o recebimento

Neste exemplo, use um módulo do Node.js para obter uma lista de telefones que cancelaram o recebimento de mensagens SMS. Crie um módulo do Node.js com o nome de arquivo `sns_listnumbersoptedout.js`. Configure o SDK como mostrado anteriormente. Crie um objeto vazio como parâmetro.

Passe o objeto para o método `listPhoneNumbersOptedOut` da classe de cliente `AWS.SNS`. Para chamar o método `listPhoneNumbersOptedOut`, crie uma promessa para invocar um objeto de serviço do Amazon SNS passando o objeto dos parâmetros. Depois, lide com `response` no retorno de chamada da promessa.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set region
AWS.config.update({ region: "REGION" });

// Create promise and SNS service object
var phonelistPromise = new AWS.SNS({ apiVersion: "2010-03-31" })
  .listPhoneNumbersOptedOut({})
  .promise();

// Handle promise's fulfilled/rejected states
phonelistPromise
  .then(function (data) {
    console.log(data);
  })
  .catch(function (err) {
    console.error(err, err.stack);
```

```
  });
```

Para executar o exemplo, digite o seguinte na linha de comando.

```
node sns_listnumbersoptedout.js
```

Este código de exemplo pode ser encontrado aqui no GitHub.

Publicar uma mensagem SMS

Neste exemplo, use um módulo do Node.js para enviar uma mensagem SMS a um número de telefone. Crie um módulo do Node.js com o nome de arquivo `sns_publishsms.js`. Configure o SDK como mostrado anteriormente. Crie um objeto contendo os parâmetros `Message` e `PhoneNumber`.

Ao enviar uma mensagem SMS, especifique o número de telefone usando o formato E.164. E.164 é um padrão para a estrutura de número de telefone usada para telecomunicações internacionais. Números de telefone que seguem esse formato podem ter um máximo de 15 dígitos, e eles são prefixados com o caractere de mais (+) e o código do país. Por exemplo, um número de telefone dos EUA no formato E.164 seria exibido como +1001XXX5550100.

Este exemplo define o parâmetro `PhoneNumber` para especificar o número de telefone ao qual a mensagem deve ser enviada. Passe o objeto para o método `publish` da classe de cliente `AWS.SNS`. Para chamar o método `publish`, crie uma promessa para invocar um objeto de serviço do Amazon SNS passando o objeto dos parâmetros. Depois, lide com `response` no retorno de chamada da promessa.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set region
AWS.config.update({ region: "REGION" });

// Create publish parameters
var params = {
  Message: "TEXT_MESSAGE" /* required */,
  PhoneNumber: "E.164_PHONE_NUMBER",
};

// Create promise and SNS service object
var publishTextPromise = new AWS.SNS({ apiVersion: "2010-03-31" })
  .publish(params)
```

```
  .promise();

// Handle promise's fulfilled/rejected states
publishTextPromise
  .then(function (data) {
    console.log("MessageID is " + data.MessageId);
  })
  .catch(function (err) {
    console.error(err, err.stack);
  });
```

Para executar o exemplo, digite o seguinte na linha de comando.

```
node sns_publishsms.js
```

Este código de exemplo pode ser encontrado aqui no GitHub.

Exemplos do Amazon SQS

O Amazon Simple Queue Service (Amazon SQS) é um serviço de fila de mensagens rápido, confiável, escalável e totalmente gerenciado. O Amazon SQS permite que você desacople os componentes de um aplicativo na nuvem. O Amazon SQS inclui filas padrão com grande throughput e pelo menos um processamento, além de filas FIFO que fornecem entrega primeiro a entrar, primeiro a sair (FIFO) e processamento único.

A API JavaScript do Amazon SQS é exposta por meio da classe de cliente `AWS.SQS`. Para obter mais informações sobre como usar a classe de cliente do Amazon SQS, consulte `Class: AWS.SQS` na referência da API.

Tópicos

- Uso de filas no Amazon SQS
- Enviar e receber mensagens no Amazon SQS
- Gerenciar o tempo limite de visibilidade no Amazon SQS
- Habilitar a sondagem longa no Amazon SQS
- Usar dead letter queues no Amazon SQS

Uso de filas no Amazon SQS

Este exemplo de código Node.js mostra:

- Como obter uma lista das filas de mensagens
- Como obter a URL de uma fila específica
- Como criar e excluir filas

Sobre o exemplo

Neste exemplo, é usada uma série de módulos do Node.js para trabalhar com filas. Os módulos do Node.js usam o SDK para JavaScript para permitir que as filas chamem os seguintes métodos da classe de cliente `AWS.SQS`:

- `listQueues`
- `createQueue`
- `getQueueUrl`
- `deleteQueue`

Para obter mais informações sobre as mensagens do Amazon SQS, consulte como Como as filas funcionam no Guia do desenvolvedor do Amazon Simple Queue Service.

Tarefas de pré-requisito

Para configurar e executar este exemplo, você deve primeiro concluir estas tarefas:

- Instale o Node.js. Para obter mais informações sobre como instalar Node.js, consulte o website de Node.js.
- Crie um arquivo de configurações compartilhado com as credenciais de usuário. Para obter mais informações sobre como fornecer um arquivo de credenciais compartilhadas, consulte Carregar credenciais em Node.js do arquivo de credenciais compartilhado.

Listar suas filas

Crie um módulo do Node.js com o nome de arquivo `sqs_listqueues.js`. Não se esqueça de configurar o SDK conforme mostrado anteriormente. Para acessar o Amazon SQS, crie um objeto de serviço do `AWS.SQS`. Crie um objeto JSON contendo os parâmetros necessários para listar as filas, que é um objeto vazio, por padrão. Chame o método `listQueues` para recuperar a lista de filas. O retorno de chamada retorna os URLs de todas as filas.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create an SQS service object
var sqs = new AWS.SQS({ apiVersion: "2012-11-05" });

var params = {};

sqs.listQueues(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.QueueUrls);
  }
});
```

Para executar o exemplo, digite o seguinte na linha de comando.

```
node sqs_listqueues.js
```

Este código de exemplo pode ser encontrado aqui no GitHub.

Criação de uma fila

Crie um módulo do Node.js com o nome de arquivo `sqs_createqueue.js`. Não se esqueça de configurar o SDK conforme mostrado anteriormente. Para acessar o Amazon SQS, crie um objeto de serviço do `AWS.SQS`. Crie um objeto JSON contendo os parâmetros necessários para listar as filas, que deve incluir o nome para a fila criada. Os parâmetros também podem conter atributos para a fila, como número de segundos para os quais a entrega de mensagens está atrasada ou o número de segundos para manter uma mensagem recebida. Chame o método `createQueue`. O retorno de chamada retorna o URL da fila criada.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create an SQS service object
var sqs = new AWS.SQS({ apiVersion: "2012-11-05" });

var params = {
  QueueName: "SQS_QUEUE_NAME",
  Attributes: {
    DelaySeconds: "60",
    MessageRetentionPeriod: "86400",
  },
};

sqs.createQueue(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.QueueUrl);
  }
});
```

Para executar o exemplo, digite o seguinte na linha de comando.

```
node sqs_createqueue.js
```

Este código de exemplo pode ser encontrado aqui no GitHub.

Obter o URL de uma fila

Crie um módulo do Node.js com o nome de arquivo `sqs_getqueueurl.js`. Não se esqueça de configurar o SDK conforme mostrado anteriormente. Para acessar o Amazon SQS, crie um objeto de serviço do `AWS.SQS`. Crie um objeto JSON contendo os parâmetros necessários para listar as filas, que deve incluir o nome para a fila cujo URL você quer. Chame o método `getQueueUrl`. O retorno de chamada retorna o URL da fila especificada.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create an SQS service object
var sqs = new AWS.SQS({ apiVersion: "2012-11-05" });

var params = {
  QueueName: "SQS_QUEUE_NAME",
};

sqs.getQueueUrl(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.QueueUrl);
  }
});
```

Para executar o exemplo, digite o seguinte na linha de comando.

```
node sqs_getqueueurl.js
```

Este código de exemplo pode ser encontrado aqui no GitHub.

Exclusão de uma fila

Crie um módulo do Node.js com o nome de arquivo `sqs_deletequeue.js`. Não se esqueça de configurar o SDK conforme mostrado anteriormente. Para acessar o Amazon SQS, crie um objeto de serviço do `AWS.SQS`. Crie um objeto JSON contendo os parâmetros necessários para excluir uma fila, que consiste no URL da fila que você deseja excluir. Chame o método `deleteQueue`.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create an SQS service object
var sqs = new AWS.SQS({ apiVersion: "2012-11-05" });

var params = {
  QueueUrl: "SQS_QUEUE_URL",
};

sqs.deleteQueue(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

Para executar o exemplo, digite o seguinte na linha de comando.

```
node sqs_deletequeue.js
```

Este código de exemplo pode ser encontrado aqui no GitHub.

Enviar e receber mensagens no Amazon SQS

Este exemplo de código Node.js mostra:

- Como enviar mensagens em uma fila.
- Como receber mensagens em uma fila.
- Como excluir mensagens em uma fila.

O cenário

Neste exemplo, é usada uma série de módulos do Node.js para enviar e receber mensagens. Os módulos do Node.js usam o SDK para JavaScript para enviar e receber mensagens usando estes métodos da classe de cliente do `AWS.SQS`:

- `sendMessage`
- `receiveMessage`
- `deleteMessage`

Para obter mais informações sobre mensagens do Amazon SQS, consulte Envio de uma mensagem para uma fila do Amazon SQS Queue e Recebimento e exclusão de uma mensagem de uma fila do Amazon SQS Queue no Guia do desenvolvedor do Amazon SQS Queue.

Tarefas de pré-requisito

Para configurar e executar este exemplo, você deve primeiro concluir estas tarefas:

- Instale o Node.js. Para obter mais informações sobre como instalar Node.js, consulte o website de Node.js.
- Crie um arquivo de configurações compartilhado com as credenciais de usuário. Para obter mais informações sobre como fornecer um arquivo de credenciais compartilhadas, consulte Carregar credenciais em Node.js do arquivo de credenciais compartilhado.
- Como criar uma fila do Amazon SQS. Para ver um exemplo de como criar uma fila, consulte Uso de filas no Amazon SQS.

Enviar uma mensagem para uma fila

Crie um módulo do Node.js com o nome de arquivo `sqs_sendmessage.js`. Não se esqueça de configurar o SDK conforme mostrado anteriormente. Para acessar o Amazon SQS, crie um objeto de serviço do `AWS.SQS`. Crie um objeto JSON que contém os parâmetros necessários para sua mensagem, incluindo o URL da fila para a qual você deseja enviar essa mensagem. Neste exemplo, a mensagem fornece detalhes sobre um livro em uma lista de mais vendidos de ficção e inclui o título, o autor e o número de semanas na lista dos mais vendidos.

Chame o método `sendMessage`. O retorno de chamada retorna o ID exclusivo da mensagem.

```
// Load the AWS SDK for Node.js
```

```
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create an SQS service object
var sqs = new AWS.SQS({ apiVersion: "2012-11-05" });

var params = {
  // Remove DelaySeconds parameter and value for FIFO queues
  DelaySeconds: 10,
  MessageAttributes: {
    Title: {
      DataType: "String",
      StringValue: "The Whistler",
    },
    Author: {
      DataType: "String",
      StringValue: "John Grisham",
    },
    WeeksOn: {
      DataType: "Number",
      StringValue: "6",
    },
  },
  MessageBody:
    "Information about current NY Times fiction bestseller for week of 12/11/2016.",
  // MessageDeduplicationId: "TheWhistler",  // Required for FIFO queues
  // MessageGroupId: "Group1",  // Required for FIFO queues
  QueueUrl: "SQS_QUEUE_URL",
};

sqs.sendMessage(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.MessageId);
  }
});
```

Para executar o exemplo, digite o seguinte na linha de comando.

```
node sqs_sendmessage.js
```

Este código de exemplo pode ser encontrado aqui no GitHub.

Receber e excluir mensagens de uma fila

Crie um módulo do Node.js com o nome de arquivo `sqs_receivemessage.js`. Não se esqueça de configurar o SDK conforme mostrado anteriormente. Para acessar o Amazon SQS, crie um objeto de serviço do `AWS.SQS`. Crie um objeto JSON que contém os parâmetros necessários para sua mensagem, incluindo o URL da fila da qual você deseja receber mensagens. Neste exemplo, os parâmetros especificam o recebimento de todos os atributos de mensagem, bem como o recebimento de não mais de 10 mensagens.

Chame o método `receiveMessage`. O retorno de chamada retorna um array de objetos `Message` a partir do qual você pode recuperar `ReceiptHandle` para cada mensagem que usar para mais tarde excluir essa mensagem. Crie outro objeto JSON contendo os parâmetros necessários para excluir a mensagem, que são o URL da fila e o valor `ReceiptHandle`. Chame o método `deleteMessage` para excluir a mensagem que você recebeu.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create an SQS service object
var sqs = new AWS.SQS({ apiVersion: "2012-11-05" });

var queueURL = "SQS_QUEUE_URL";

var params = {
  AttributeNames: ["SentTimestamp"],
  MaxNumberOfMessages: 10,
  MessageAttributeNames: ["All"],
  QueueUrl: queueURL,
  VisibilityTimeout: 20,
  WaitTimeSeconds: 0,
};

sqs.receiveMessage(params, function (err, data) {
  if (err) {
    console.log("Receive Error", err);
  } else if (data.Messages) {
    var deleteParams = {
      QueueUrl: queueURL,
```

```
      ReceiptHandle: data.Messages[0].ReceiptHandle,
    };
    sqs.deleteMessage(deleteParams, function (err, data) {
      if (err) {
        console.log("Delete Error", err);
      } else {
        console.log("Message Deleted", data);
      }
    });
  }
});
```

Para executar o exemplo, digite o seguinte na linha de comando.

```
node sqs_receivemessage.js
```

Este código de exemplo pode ser encontrado aqui no GitHub.

Gerenciar o tempo limite de visibilidade no Amazon SQS

Este exemplo de código Node.js mostra:

- Como especificar o intervalo de tempo durante o qual as mensagens recebidas por uma fila não ficam visíveis.

O cenário

Neste exemplo, o módulo Node.js é usado para gerenciar o tempo limite de visibilidade. O Node.js usa o SDK para JavaScript para gerenciar o tempo limite de visibilidade usando este método da classe de cliente `AWS.SQS`:

- `changeMessageVisibility`

Para obter mais informações sobre tempos limite de visibilidade do Amazon SQS, consulte Tempo limite de visibilidade no Guia do desenvolvedor do Amazon Simple Queue Service.

Tarefas de pré-requisito

Para configurar e executar este exemplo, você deve primeiro concluir estas tarefas:

- Instale o Node.js. Para obter mais informações sobre como instalar Node.js, consulte o website de Node.js.
- Crie um arquivo de configurações compartilhado com as credenciais de usuário. Para obter mais informações sobre como fornecer um arquivo de credenciais compartilhadas, consulte Carregar credenciais em Node.js do arquivo de credenciais compartilhado.
- Como criar uma fila do Amazon SQS. Para ver um exemplo de como criar uma fila, consulte Uso de filas no Amazon SQS.
- Envie uma mensagem para a fila. Para obter um exemplo de envio de uma mensagem para uma fila, consulte Enviar e receber mensagens no Amazon SQS.

Alterar o tempo limite de visibilidade

Crie um módulo do Node.js com o nome de arquivo `sqs_changingvisibility.js`. Não se esqueça de configurar o SDK conforme mostrado anteriormente. Para acessar o Amazon Simple Queue Service, crie um objeto de serviço do `AWS.SQS`. Receba a mensagem da fila.

Após o recebimento da mensagem da fila, crie um objeto JSON contendo os parâmetros necessários para configurar o tempo limite, incluindo o URL da fila que contém a mensagem, o `ReceiptHandle` retornado quando a mensagem foi recebida e o novo tempo limite, em segundos. Chame o método `changeMessageVisibility`.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region to us-west-2
AWS.config.update({ region: "us-west-2" });

// Create the SQS service object
var sqs = new AWS.SQS({ apiVersion: "2012-11-05" });

var queueURL = "https://sqs.REGION.amazonaws.com/ACCOUNT-ID/QUEUE-NAME";

var params = {
  AttributeNames: ["SentTimestamp"],
  MaxNumberOfMessages: 1,
  MessageAttributeNames: ["All"],
```

```
  QueueUrl: queueURL,
};

sqs.receiveMessage(params, function (err, data) {
  if (err) {
    console.log("Receive Error", err);
  } else {
    // Make sure we have a message
    if (data.Messages != null) {
      var visibilityParams = {
        QueueUrl: queueURL,
        ReceiptHandle: data.Messages[0].ReceiptHandle,
        VisibilityTimeout: 20, // 20 second timeout
      };
      sqs.changeMessageVisibility(visibilityParams, function (err, data) {
        if (err) {
          console.log("Delete Error", err);
        } else {
          console.log("Timeout Changed", data);
        }
      });
    } else {
      console.log("No messages to change");
    }
  }
});
```

Para executar o exemplo, digite o seguinte na linha de comando.

```
node sqs_changingvisibility.js
```

Este código de exemplo pode ser encontrado aqui no GitHub.

Habilitar a sondagem longa no Amazon SQS

Este exemplo de código Node.js mostra:

- Como para habilitar a sondagem longa para uma fila recém-criada

- Como habilitar a sondagem longa para uma fila existente
- Como habilitar a sondagem longa ao receber uma mensagem

O cenário

A sondagem longa reduz o número de respostas vazias ao permitir que o Amazon SQS espere um tempo especificado para que uma mensagem se torne disponível na fila antes de enviar uma resposta. Além disso, a sondagem longa elimina respostas vazias falsas consultando todos os servidores em vez de apenas uma amostragem de servidores. Para habilitar a sondagem longa, é necessário especificar um tempo de espera diferente de zero para mensagens recebidas. Isso pode ser feito configurando o parâmetro `ReceiveMessageWaitTimeSeconds` de uma fila ou configurando o parâmetro `WaitTimeSeconds` em uma mensagem quando ela for recebida.

Neste exemplo, é usada uma série de módulos do Node.js para habilitar a sondagem longa. Os módulos do Node.js usam o SDK para JavaScript para habilitar a sondagem longa usando estes métodos da classe de cliente `AWS.SQS`:

- `setQueueAttributes`
- `receiveMessage`
- `createQueue`

Para obter mais informações sobre a sondagem longa do Amazon SQS, consulte o Guia do desenvolvedor do Amazon Simple Queue Service.

Tarefas de pré-requisito

Para configurar e executar este exemplo, você deve primeiro concluir estas tarefas:

- Instale o Node.js. Para obter mais informações sobre como instalar Node.js, consulte o website de Node.js.
- Crie um arquivo de configurações compartilhado com as credenciais de usuário. Para obter mais informações sobre como fornecer um arquivo de credenciais compartilhadas, consulte Carregar credenciais em Node.js do arquivo de credenciais compartilhado.

Habilitar sondagem longa ao criar uma fila

Crie um módulo do Node.js com o nome de arquivo `sqs_longpolling_createqueue.js`. Não se esqueça de configurar o SDK conforme mostrado anteriormente. Para acessar o Amazon SQS, crie um objeto de serviço do `AWS.SQS`. Crie um objeto JSON contendo os parâmetros necessários para criar uma fila, incluindo um valor diferente de zero para o parâmetro `ReceiveMessageWaitTimeSeconds`. Chame o método `createQueue`. A sondagem longa será habilitada para a fila.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the SQS service object
var sqs = new AWS.SQS({ apiVersion: "2012-11-05" });

var params = {
  QueueName: "SQS_QUEUE_NAME",
  Attributes: {
    ReceiveMessageWaitTimeSeconds: "20",
  },
};

sqs.createQueue(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.QueueUrl);
  }
});
```

Para executar o exemplo, digite o seguinte na linha de comando.

```
node sqs_longpolling_createqueue.js
```

Este código de exemplo pode ser encontrado aqui no GitHub.

Habilitar sondagem longa em uma fila existente

Crie um módulo do Node.js com o nome de arquivo `sqs_longpolling_existingqueue.js`. Não se esqueça de configurar o SDK conforme mostrado anteriormente. Para acessar o Amazon

Simple Queue Service, crie um objeto de serviço do `AWS.SQS`. Crie um objeto JSON contendo os parâmetros necessários para definir os atributos da fila, incluindo um valor diferente de zero para o parâmetro `ReceiveMessageWaitTimeSeconds` e o URL da fila. Chame o método `setQueueAttributes`. A sondagem longa será habilitada para a fila.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the SQS service object
var sqs = new AWS.SQS({ apiVersion: "2012-11-05" });

var params = {
  Attributes: {
    ReceiveMessageWaitTimeSeconds: "20",
  },
  QueueUrl: "SQS_QUEUE_URL",
};

sqs.setQueueAttributes(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

Para executar o exemplo, digite o seguinte na linha de comando.

```
node sqs_longpolling_existingqueue.js
```

Este código de exemplo pode ser encontrado aqui no GitHub.

Habilitar sondagem longa no recebimento da mensagem

Crie um módulo do Node.js com o nome de arquivo `sqs_longpolling_receivemessage.js`. Não se esqueça de configurar o SDK conforme mostrado anteriormente. Para acessar o Amazon Simple Queue Service, crie um objeto de serviço do `AWS.SQS`. Crie um objeto JSON contendo os parâmetros necessários para receber mensagens, incluindo um valor diferente de zero para o parâmetro `WaitTimeSeconds` e o URL da fila. Chame o método `receiveMessage`.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the SQS service object
var sqs = new AWS.SQS({ apiVersion: "2012-11-05" });

var queueURL = "SQS_QUEUE_URL";

var params = {
  AttributeNames: ["SentTimestamp"],
  MaxNumberOfMessages: 1,
  MessageAttributeNames: ["All"],
  QueueUrl: queueURL,
  WaitTimeSeconds: 20,
};

sqs.receiveMessage(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

Para executar o exemplo, digite o seguinte na linha de comando.

```
node sqs_longpolling_receivemessage.js
```

Este código de exemplo pode ser encontrado aqui no GitHub.

Usar dead letter queues no Amazon SQS

Este exemplo de código Node.js mostra:

- Como usar uma fila para receber e guardar mensagens de outras filas que as filas não consigam processar.

O cenário

Uma dead letter queue é aquela fila que outras filas (de origem) podem direcionar para mensagens que não podem ser processadas com sucesso. Você pode separar e isolar essas mensagens na dead letter queue para determinar por que o processamento não teve sucesso. Você deve configurar individualmente cada fila de origem que envia mensagens para uma dead letter queue. Várias filas podem visar uma única dead letter queue.

Neste exemplo, o módulo Node.js é usado para rotear mensagens para uma fila de mensagens mortas. O módulo Node.js usa o SDK para JavaScript para usar filas de mensagens mortas com este método da classe de cliente `AWS.SQS`:

- `setQueueAttributes`

Para obter mais informações sobre a fila de mensagens não entregues do Amazon SQS, consulte Usar filas de mensagens não entregues do Amazon SQS no Guia do desenvolvedor do Amazon Simple Queue Service.

Tarefas de pré-requisito

Para configurar e executar este exemplo, você deve primeiro concluir estas tarefas:

- Instale o Node.js. Para obter mais informações sobre como instalar Node.js, consulte o website de Node.js.
- Crie um arquivo de configurações compartilhado com as credenciais de usuário. Para obter mais informações sobre como fornecer um arquivo de credenciais compartilhadas, consulte Carregar credenciais em Node.js do arquivo de credenciais compartilhado.
- Criar uma fila do Amazon SQS; para servir como uma dead letter queue. Para ver um exemplo de como criar uma fila, consulte Uso de filas no Amazon SQS.

Configurar filas de origem

Depois ter criado uma fila para atuar como uma dead letter queue, você deve configurar outras filas que roteiam mensagens não processadas para a dead letter queue. Para isso, especifique a política

de redrive que identifica a fila a ser usada como dead letter queue e o número máximo que recebe por mensagens individuais antes de roteada para a dead letter queue.

Crie um módulo do Node.js com o nome de arquivo `sqs_deadletterqueue.js`. Não se esqueça de configurar o SDK conforme mostrado anteriormente. Para acessar o Amazon SQS, crie um objeto de serviço do `AWS.SQS`. Crie um objeto JSON contendo os parâmetros necessários para atualizar os atributos da fila, incluindo o parâmetro `RedrivePolicy`, que especifica tanto o ARN quanto a fila de mensagens mortas e o valor de `maxReceiveCount`. Especifique também a fila de origem do URL que você deseja configurar. Chame o método `setQueueAttributes`.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the SQS service object
var sqs = new AWS.SQS({ apiVersion: "2012-11-05" });

var params = {
  Attributes: {
    RedrivePolicy:
      '{"deadLetterTargetArn":"DEAD_LETTER_QUEUE_ARN","maxReceiveCount":"10"}',
  },
  QueueUrl: "SOURCE_QUEUE_URL",
};

sqs.setQueueAttributes(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

Para executar o exemplo, digite o seguinte na linha de comando.

```
node sqs_deadletterqueue.js
```

Este código de exemplo pode ser encontrado aqui no GitHub.

Tutoriais

Os tutoriais a seguir mostram como executar tarefas diferentes relacionadas ao uso do AWS SDK for JavaScript.

Tópicos

- Tutorial: Configuração do Node.js em uma instância do Amazon EC2

Tutorial: Configuração do Node.js em uma instância do Amazon EC2

Um cenário comum para usar Node.js com o SDK para JavaScript é configurar e executar um aplicativo web Node.js em uma instância do Amazon Elastic Compute Cloud (Amazon EC2). Neste tutorial, você criará uma instância do Linux; conecte-se a ela usando o SSH e instale o Node.js para rodar nessa instância.

Pré-requisitos

Este tutorial pressupõe que você já tenha aberto uma instância do Linux com um nome DNS público que possa ser acessado na Internet e para o qual você possa se conectar usando o SSH. Para obter mais informações, consulte Step 1: Launch an Instance no Guia do usuário do Amazon EC2.

> **Important**
>
> Use a imagem de máquina da Amazon (AMI) do Amazon Linux 2023 ao iniciar uma nova instância do Amazon EC2.

Você também precisa ter configurado o grupo de segurança para permitir as conexões SSH (porta 22), HTTP (porta 80) e HTTPS (porta 443). Para obter informações sobre esses pré-requisitos, consulte Setting Up with Amazon Amazon EC2 no Guia do usuário do Amazon EC2.

Procedimento

O procedimento a seguir ajuda você a instalar o Node.js em uma instância do Amazon Linux. Você pode usar esse servidor para hospedar um aplicativo web do Node.js.

Para configurar o Node.js em sua instância do Linux:

1. Conecte-se à sua instância do Linux como `ec2-user` usando SSH.
2. Instale o gerenciador de versão do nó (nvm, do inglês "node version manager") digitando o seguinte na linha de comando.

   > **Warning**
   >
   > AWS não controla o seguinte código. Antes de executar, certifique-se de verificar sua autenticidade e integridade. Mais informações sobre esse código podem ser encontradas no repositório GitHub do nvm.

   ```
   curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.39.7/install.sh | bash
   ```

   Usaremos o nvm para instalar o Node.js, pois o nvm pode instalar várias versões do Node.js e permitir que você alterne entre elas.
3. Para carregar o `nvm`, digite o seguinte na linha de comando:

   ```
   source ~/.bashrc
   ```
4. Use o nvm para instalar a versão do LTS mais recente do Node.js digitando o seguinte na linha de comando.

   ```
   nvm install --lts
   ```

   Instalar Node.js também instala o gerenciador de pacotes do nó (npm, do inglês "node package manager") para que você possa instalar módulos adicionais, conforme necessário.
5. Verifique se o Node.js está instalado e funcionando corretamente ao digitar o seguinte na linha de comando.

   ```
   node -e "console.log('Running Node.js ' + process.version)"
   ```

   Isso exibe a seguinte mensagem que mostra a versão do Node.js em execução.

   ```
   Running Node.js VERSION
   ```

> **Note**
>
> A instalação do nó se aplica somente à sessão atual do Amazon EC2. Se você reiniciar sua sessão de CLI, precisará usar o nvm para habilitar a versão do nó instalado. Se a instância for encerrada, é necessário instalar o nó novamente. A alternativa é criar uma imagem de máquina da Amazon (AMI) da instância do Amazon EC2 assim que você tiver a configuração que deseja manter, conforme descrito no tópico a seguir.

Criar uma imagem de máquina da Amazon

Depois de instalar o Node.js em uma instância do Amazon EC2, você pode criar uma Imagem de Máquina da Amazon (AMI) para essa instância. A criação de uma AMI facilita o provisionamento de várias instâncias do Amazon EC2 com a mesma instalação do Node.js. Para obter mais informações sobre como criar uma AMI de uma instância existente, consulte Creating an Amazon EBS-Backed Linux AMI no Guia de usuário do Amazon EC2.

Recursos relacionados

Para obter mais informações sobre os comandos e o software usados neste tópico, consulte as seguintes páginas da web:

- gerenciador de versão do nó (nvm): consulte repositório de nvm no GitHub.
- gerenciador de pacotes do nó (npm): consulte o site do npm.

Referência de API do JavaScript

Os tópicos da referência da API para a versão mais recente do SDK para JavaScript estão em:

AWS SDK for JavaScriptGuia de referência de APIs

Changelog do SDK no GitHub

O log de alterações para versões da versão 2.4.8 e posteriores está em:

Log de alterações

Segurança para este produto ou serviço da AWS

A segurança da nuvem na Amazon Web Services (AWS) é a nossa maior prioridade. Como cliente da AWS, você contará com um datacenter e uma arquitetura de rede criados para atender aos requisitos das organizações com as maiores exigências de segurança. A segurança é uma responsabilidade compartilhada entre a AWS e você. O modelo de responsabilidade compartilhada descreve isso como a Segurança da nuvem e a Segurança na nuvem.

Segurança da nuvem: a AWS é responsável pela proteção da infraestrutura que executa todos os serviços oferecidos na Nuvem AWS e por fornecer serviços que você pode usar com segurança. Nossa responsabilidade de segurança é a maior prioridade na AWS, e a eficácia da nossa segurança é regularmente testada e verificada por auditores terceirizados como parte dos Programas de Compatibilidade da AWS.

Segurança na nuvem: sua responsabilidade é determinada pelo serviço da AWS que você estiver usando e por outros fatores, incluindo a confidencialidade dos dados, os requisitos da organização e as leis e regulamentos aplicáveis.

Esse produto ou serviço da AWS segue o modelo de responsabilidade compartilhada por meio dos serviços específicos da Amazon Web Services (AWS) compatíveis. Para obter informações de segurança sobre o serviço da AWS, consulte a página de documentação de segurança do serviço da AWS e Serviços da AWS que estão no escopo dos esforços de conformidade da AWS por programa de conformidade.

Tópicos

- Proteção de dados neste produto ou serviço da AWS
- Identity and Access Management
- Validação de conformidade para este produto ou serviço da AWS
- Resiliência para este produto ou serviço da AWS
- Segurança da infraestrutura para esse produto ou serviço da AWS
- Aplicar uma versão mínima do TLS

Proteção de dados neste produto ou serviço da AWS

O Modelo de Responsabilidade Compartilhada da AWS se aplica à proteção de dados nesse produto ou serviço da AWS. Conforme descrito nesse modelo, AWS é responsável por proteger a

infraestrutura global que executa todas as Nuvem AWS. Você é responsável por manter o controle sobre seu conteúdo hospedado nessa infraestrutura. Você também é responsável pelas tarefas de configuração e gerenciamento de segurança dos Serviços da AWS que usa. Para obter mais informações sobre a privacidade de dados, consulte as Perguntas Frequentes sobre Privacidade de Dados.. Para obter mais informações sobre a proteção de dados na Europa, consulte a postagem do blog AWS Shared Responsibility Model and GDPR no Blog de segurança da AWS.

Para fins de proteção de dados, recomendamos que você proteja as credenciais da Conta da AWS e configure as contas de usuário individuais com AWS IAM Identity Center ou AWS Identity and Access Management (IAM). Dessa maneira, cada usuário receberá apenas as permissões necessárias para cumprir suas obrigações de trabalho. Recomendamos também que você proteja seus dados das seguintes formas:

- Use uma autenticação multifator (MFA) com cada conta.
- Use SSL/TLS para se comunicar com os recursos da AWS. Exigimos TLS 1.2 e recomendamos TLS 1.3.
- Configure a API e atividade do usuário logando com AWS CloudTrail.
- Use as soluções de criptografia AWS, juntamente com todos os controles de segurança padrão em Serviços da AWS.
- Use serviços gerenciados de segurança avançada, como o Amazon Macie, que ajuda a localizar e proteger dados sigilosos armazenados no Amazon S3.
- Se você precisar de módulos criptográficos validados pelo FIPS 140-2 ao acessar AWS por meio de uma interface de linha de comando ou uma API, use um endpoint FIPS. Para ter mais informações sobre endpoints do FIPS disponíveis, consulte Federal Information Processing Standard (FIPS) 140-2.

É altamente recomendável que nunca sejam colocadas informações de identificação confidenciais, como endereços de e-mail dos seus clientes, em marcações ou campos de formato livre, como um campo Nome. Isso inclui quando você trabalha com esse produto ou serviço da AWS ou outros Serviços da AWS usando o console, a API, a AWS CLI ou os AWS SDKs. Quaisquer dados inseridos em tags ou campos de texto de formato livre usados para nomes podem ser usados para logs de faturamento ou de diagnóstico. Se você fornecer um URL para um servidor externo, recomendemos fortemente que não sejam incluídas informações de credenciais no URL para validar a solicitação a esse servidor.

Identity and Access Management

AWS Identity and Access Management (IAM) é um serviço da AWS service (Serviço da AWS) que ajuda o administrador no controle de segurança de acesso aos recursos da AWS de forma segura. Os administradores do IAM controlam quem pode ser autenticado (fazer login) e autorizado (ter permissões) a usar os recursos do AWS. O IAM é um AWS service (Serviço da AWS) que pode ser usado sem custo adicional.

Tópicos

- Público
- Autenticando com identidades
- Gerenciando acesso usando políticas
- Como Serviços da AWS funcionam com o IAM
- Solução de problemas de identidade e acesso do AWS

Público

O uso do AWS Identity and Access Management (IAM) varia dependendo do trabalho que for realizado no AWS.

Usuário do serviço: se você usar Serviços da AWS para fazer seu trabalho, o administrador fornecerá as credenciais e as permissões necessárias. À medida que usar mais atributos do AWS para fazer seu trabalho, você poderá precisar de permissões adicionais. Entender como o acesso é gerenciado pode ajudá-lo a solicitar as permissões corretas ao seu administrador. Se você não conseguir acessar um atributo na AWS, consulte Solução de problemas de identidade e acesso do AWS ou o guia do usuário do AWS service (Serviço da AWS) que você está usando.

Administrador do serviço: se você for o responsável pelos recursos do AWS na empresa, provavelmente terá acesso total ao AWS. Cabe a você determinar quais atributos e recursos do AWS os usuários do serviço devem acessar. Assim, você deve enviar solicitações ao administrador do IAM para alterar as permissões dos usuários de seu serviço. Revise as informações nesta página para entender os Introdução ao IAM. Para saber mais sobre como sua empresa pode usar o IAM com a AWS, consulte o guia do usuário do AWS service (Serviço da AWS) que você está usando.

Administrador do IAM: Se você for um administrador do IAM, talvez queira saber detalhes sobre como pode gravar políticas para gerenciar acesso ao AWS. Para visualizar exemplos de políticas

baseadas em identidade da AWS que podem ser usadas no IAM, consulte o guia do usuário do AWS service (Serviço da AWS) que você está usando.

Autenticando com identidades

A autenticação é a forma como você faz login na AWS usando suas credenciais de identidade. É necessário ser autenticado (fazer login na AWS) como Usuário raiz da conta da AWS, como usuário do IAM, ou assumindo um perfil do IAM.

Você pode fazer login na AWS como uma identidade federada usando credenciais fornecidas por uma fonte de identidades. AWS IAM Identity Center Os usuários (IAM Identity Center), a autenticação única da empresa e as suas credenciais do Google ou do Facebook são exemplos de identidades federadas. Quando você faz login como identidade federada, o administrador já configurou anteriormente a federação de identidades usando perfis do IAM. Quando você acessa a AWS usando a federação, está indiretamente assumindo um perfil.

A depender do tipo de usuário, você pode fazer login no AWS Management Console ou no portal de acesso AWS. Para obter mais informações sobre como fazer login na AWS, consulte Como fazer login na contaConta da AWS no Início de Sessão da AWS Guia do usuário .

Se você acessar a AWS programaticamente, a AWS fornecerá um kit de desenvolvimento de software (SDK) e uma interface de linha de comando (CLI) para você assinar criptograficamente as solicitações usando as suas credenciais. Se você não utilizar as ferramentas AWS, deverá designar as solicitações por conta própria. Para obter mais informações sobre como usar o método recomendado para designar solicitações por conta própria, consulte Designando solicitações de API AWS no Guia do usuário do IAM.

Independente do método de autenticação usado, também pode ser exigido que você forneça informações adicionais de segurança. Por exemplo, a AWS recomenda o uso da autenticação multifator (MFA) para aumentar a segurança de sua conta. Para saber mais, consulte Autenticação Multifator no Guia do Usuário do AWS IAM Identity Center. Usar a autenticação multifator (MFA) na AWS no Guia do Usuário do IAM.

Usuário raiz Conta da AWS

Ao criar uma Conta da AWS, você começa com uma identidade de login com acesso completo a todos os Serviços da AWS e recursos na conta. Essa identidade, chamada usuário raiz da Conta da AWS, é acessada por login com o endereço de e-mail e a senha usada para criar a conta. É altamente recomendável não usar o usuário raiz para tarefas diárias. Proteja as credenciais do

usuário raiz e use-as para executar as tarefas que somente ele puder executar. Para obter a lista completa das tarefas que exigem login como usuário raiz, consulte Tarefas que exigem credenciais de usuário raiz no Guia do Usuário do IAM.

Identidade federada

Como prática recomendada, exija que os usuários, inclusive os que precisam de acesso de administrador, usem a federação com um provedor de identidades para acessar os Serviços da AWS usando credenciais temporárias.

Identidade federada é um usuário de seu diretório de usuários corporativos, um provedor de identidades da web, o AWS Directory Service, o diretório do Identity Center, ou qualquer usuário que acesse os Serviços da AWS usando credenciais fornecidas por meio de uma fonte de identidade. Quando as identidades federadas acessam Contas da AWS, elas assumem perfis que fornecem credenciais temporárias.

Para o gerenciamento de acesso centralizado, recomendamos usar o AWS IAM Identity Center. Você pode criar usuários e grupos no IAM Identity Center ou conectar-se e sincronizar com um conjunto de usuários e grupos em sua própria fonte de identidade para uso em todas as suas Contas da AWS e aplicações. Para obter mais informações sobre o Centro de Identidade do IAM, consulte O que é o Centro de Identidade do IAM? no Manual do Usuário doAWS IAM Identity Center.

Usuários e grupos do IAM

Um usuário do IAM é uma identidade dentro da Conta da AWS que tem permissões específicas para uma única pessoa ou aplicativo. Sempre que possível, recomendamos contar com credenciais temporárias em vez de criar usuários do IAM com credenciais de longo prazo, como senhas e chaves de acesso. No entanto, se você tiver casos de uso específicos que exijam credenciais de longo prazo com usuários do IAM, recomendamos alternar as chaves de acesso. Para obter mais informações, consulte Alterne as chaves de acesso regularmente para casos de uso que exijam credenciais de longo prazo no Guia do Usuário do IAM.

Um grupo do IAM é uma identidade que especifica uma coleção de usuários do IAM. Não é possível fazer login como um grupo. É possível usar grupos para especificar permissões para vários usuários de uma vez. Os grupos facilitam o gerenciamento de permissões para grandes conjuntos de usuários. Por exemplo, você pode ter um grupo chamado IAMAdmins e conceder a esse grupo permissões para administrar recursos do IAM.

Usuários são diferentes de perfis. Um usuário é exclusivamente associado a uma pessoa ou a uma aplicação, mas um perfil pode ser assumido por qualquer pessoa que precisar dele. Os usuários

têm credenciais permanentes de longo prazo, mas os perfis fornecem credenciais temporárias. Para saber mais, consulte Quando criar um usuário do IAM (em vez de um perfil) no Guia do usuário do IAM.

Perfis do IAM

Um perfil do IAM é uma identidade dentro da Conta da AWS que tem permissões específicas. Ele é semelhante a um usuário do IAM, mas não está associado a uma pessoa específica. É possível presumir temporariamente um perfil do IAM no AWS Management Console alternando perfis. É possível presumir um perfil chamando uma operação de API da AWS CLI ou da AWS, ou usando um URL personalizado. Para obter mais informações sobre métodos para o uso de perfis, consulte Utilizar perfis do IAM no Guia do usuário do IAM.

Funções do IAM com credenciais temporárias são úteis nas seguintes situações:

- Acesso de usuário federado: para atribuir permissões a identidades federadas, você pode criar um perfil e definir permissões para ele. Quando uma identidade federada é autenticada, essa identidade é associada ao perfil e recebe as permissões definidas pelo mesmo. Para obter mais informações sobre perfis para federação, consulte Criar um perfil para um provedor de identidades de terceiros no Guia do Usuário do IAM. Se você usar o Centro de identidade do IAM, configure um conjunto de permissões. Para controlar o que suas identidades podem acessar após a autenticação, o Centro de identidade do IAM correlaciona o conjunto de permissões a um perfil no IAM. Para obter informações sobre conjuntos de permissões, consulte Conjuntos de Permissões no Manual do Usuário do AWS IAM Identity Center.
- Permissões temporárias para usuários do IAM –- um usuário ou um perfil do IAM pode presumir um perfil do IAM para obter temporariamente permissões diferentes para uma tarefa específica.
- Acesso entre contas –- é possível usar um perfil do IAM para permitir que alguém (uma entidade principal confiável) em outra conta acesse recursos em sua conta. Os perfis são a principal forma de conceder acesso entre contas. No entanto, alguns Serviços da AWS permitem que você anexe uma política diretamente a um recurso (em vez de usar um perfil como proxy). Para saber a diferença entre perfis e políticas baseadas em recurso para acesso entre contas, consulte Acesso a recursos entre contas no IAM no Guia do usuário do IAM.
- Acesso entre serviços: alguns Serviços da AWS usam atributos em outros Serviços da AWS. Por exemplo, quando você faz uma chamada em um serviço, é comum que esse serviço execute aplicativos no Amazon EC2 ou armazene objetos no Amazon S3. Um serviço pode fazer isso usando as permissões do principal de chamada, usando um perfil de serviço ou um perfil vinculado a serviço.

  - Encaminhamento de sessões de acesso (FAS): qualquer pessoa que utilizar uma função ou usuário do IAM para realizar ações na AWS é considerada uma entidade principal. Ao usar alguns serviços, você pode executar uma ação que inicia outra ação em um serviço diferente. O recurso FAS utiliza as permissões da entidade principal que chama um AWS service (Serviço da AWS), combinadas às permissões do AWS service (Serviço da AWS) solicitante, para realizar solicitações para serviços downstream. As solicitações de FAS só são feitas quando um serviço recebe uma solicitação que exige interações com outros Serviços da AWS ou com recursos para serem concluídas. Nesse caso, você precisa ter permissões para executar ambas as ações. Para obter detalhes da política ao fazer solicitações de FAS, consulte Encaminhar sessões de acesso.
  - Função de serviço: um perfil de serviço é um perfil do IAM que um serviço assume para realizar ações em seu nome. Um administrador do IAM pode criar, modificar e excluir um perfil de serviço do IAM. Para obter mais informações, consulte Criar um perfil para delegar permissões a um AWS service (Serviço da AWS) no Guia do Usuário do IAM.
  - Perfil vinculado a serviço: um perfil vinculado a serviço é um tipo de perfil de serviço vinculado a um AWS service (Serviço da AWS). O serviço pode presumir a função de executar uma ação em seu nome. Funções vinculadas ao serviço aparecem em sua Conta da AWS e são de propriedade do serviço. Um administrador do IAM pode visualizar, mas não editar as permissões para funções vinculadas ao serviço.
- Aplicações em execução no Amazon EC2: é possível usar um perfil do IAM para gerenciar credenciais temporárias para aplicações em execução em uma instância do EC2 e fazer solicitações da AWS CLI ou da AWS API. É preferível fazer isso a armazenar chaves de acesso na instância do EC2. Para atribuir um perfil da AWS a uma instância do EC2 e disponibilizá-la para todas as suas aplicações, crie um perfil de instância que esteja anexado a ela. Um perfil de instância contém o perfil e permite que os programas em execução na instância do EC2 obtenham credenciais temporárias. Para mais informações, consulte Utilizar um perfil do IAM para conceder permissões a aplicações em execução nas instâncias do Amazon EC2 no Guia do usuário do IAM.

Para saber se deseja usar perfis do IAM, consulte Quando criar um perfil do IAM (em vez de um usuário) no Guia do usuário do IAM.

Gerenciando acesso usando políticas

Você controla o acesso na AWS criando políticas e anexando-as a identidades ou atributos da AWS. Uma política é um objeto na AWS que, quando associado a uma identidade ou recurso, define suas permissões. A AWS avalia essas políticas quando uma entidade principal (usuário, usuário raiz ou

sessão de perfil) faz uma solicitação. As permissões nas políticas determinam se a solicitação será permitida ou negada. A maioria das políticas é armazenada na AWS como documentos JSON. Para obter mais informações sobre a estrutura e o conteúdo de documentos de políticas JSON, consulte Visão geral das políticas JSON no Guia do Usuário do IAM.

Os administradores podem usar as políticas JSON da AWS para especificar quem tem acesso a o quê. Ou seja, qual entidade principal pode executar ações em quais recursos e em que condições.

Por padrão, usuários e funções não têm permissões. Para conceder aos usuários permissões para executar ações nos recursos que eles precisam, um administrador do IAM pode criar políticas do IAM. O administrador pode então adicionar as políticas do IAM aos perfis e os usuários podem presumir os perfis.

As políticas do IAM definem permissões para uma ação independente do método usado para executar a operação. Por exemplo, suponha que você tenha uma política que permite a ação `iam:GetRole`. Um usuário com essa política pode obter informações de perfis do AWS Management Console, da AWS CLI ou da API da AWS.

Políticas baseadas em identidade

As políticas baseadas em identidade são documentos de políticas de permissões JSON que você pode anexar a uma identidade, como usuário do IAM, grupo de usuários ou perfil. Essas políticas controlam quais ações os usuários e perfis podem realizar, em quais recursos e em que condições. Para saber como criar uma política baseada em identidade, consulte Criando políticas do IAM no Guia do Usuário do IAM.

As políticas baseadas em identidade podem ser categorizadas ainda adicionalmente como políticas em linha ou políticas gerenciadas. As políticas em linha são anexadas diretamente a um único usuário, grupo ou perfil. As políticas gerenciadas são políticas independentes que podem ser anexadas a vários usuários, grupos e perfis na Conta da AWS. As políticas gerenciadas incluem políticas gerenciadas pela AWS e políticas gerenciadas pelo cliente. Para saber como escolher entre uma política gerenciada ou uma política em linha, consulte Escolher entre políticas gerenciadas e políticas em linha no Guia do Usuário do IAM.

Políticas baseadas em recursos

Políticas baseadas em recursos são documentos de políticas JSON que você anexa a um recurso. São exemplos de políticas baseadas em recursos as políticas de confiança de perfil do IAM e as políticas de bucket do Amazon S3. Em serviços que suportem políticas baseadas em recursos, os

administradores de serviço podem usá-las para controlar o acesso a um recurso específico. Para o recurso ao qual a política está anexada, a política define quais ações um principal especificado pode executar nesse recurso e em que condições. Você deve especificar uma entidade principal em uma política baseada em recursos. As entidades principais podem incluir contas, usuários, perfis, usuários federados ou Serviços da AWS.

Políticas baseadas em recursos são políticas em linha localizadas nesse serviço. Não é possível usar as políticas gerenciadas da AWS do IAM em uma política baseada em atributos.

Listas de controle de acesso (ACLs)

As listas de controle de acesso (ACLs) controlam quais entidades principais (membros, usuários ou perfis da conta) têm permissões para acessar um recurso. As ACLs são semelhantes as políticas baseadas em recursos, embora não usem o formato de documento de política JSON.

Amazon S3, AWS WAF e Amazon VPC são exemplos de serviços que oferecem compatibilidade com ACLs. Para saber mais sobre ACLs, consulte Visão geral da lista de controle de acesso (ACL) no Guia do Desenvolvedor do Amazon Simple Storage Service.

Outros tipos de política

A AWS oferece compatibilidade com tipos de política menos comuns. Esses tipos de política podem definir o máximo de permissões concedidas a você pelos tipos de política mais comuns.

- Limites de permissões: um limite de permissões é um atributo avançado no qual você define o máximo de permissões que uma política baseada em identidade pode conceder a uma entidade do IAM (usuário ou perfil do IAM). É possível definir um limite de permissões para uma entidade. As permissões resultantes são a interseção das políticas baseadas em identidade de uma entidade com seus limites de permissões. As políticas baseadas em recurso que especificam o usuário ou o perfil no campo `Principal` não são limitadas pelo limite de permissões. Uma negação explícita em qualquer uma dessas políticas substitui a permissão. Para obter mais informações sobre limites de permissões, consulte Limites de permissões para identidades do IAM no Guia do Usuário do IAM.
- Políticas de controle de serviço (SCPs) –- SCPs são políticas JSON que especificam as permissões máximas para uma organização ou unidade organizacional (UO) no AWS Organizations. O AWS Organizations é um serviço que agrupa e gerencia centralmente várias Contas da AWS pertencentes a sua empresa. Se você habilitar todos os atributos em uma organização, poderá aplicar políticas de controle de serviço (SCPs) a qualquer uma ou a todas as

contas. O SCP limita as permissões para entidades em contas membro, o que inclui cada Usuário raiz da conta da AWS. Para obter mais informações sobre o Organizações e SCPs, consulte Como os SCPs Funcionam no Manual do Usuário do AWS Organizations.

- Políticas de sessão: são políticas avançadas que você transmite como um parâmetro quando cria de forma programática uma sessão temporária para um perfil ou um usuário federado. As permissões da sessão resultante são a interseção das políticas baseadas em identidade do usuário ou do perfil e das políticas de sessão. As permissões também podem ser provenientes de uma política baseada em atributo. Uma negação explícita em qualquer uma dessas políticas substitui a permissão. Para obter mais informações, consulte Políticas de sessão no Guia do Usuário do IAM.

Vários tipos de política

Quando vários tipos de política são aplicáveis a uma solicitação, é mais complicado compreender as permissões resultantes. Para saber como a AWS determina permitir ou não uma solicitação quando há vários tipos de política envolvidos, consulte Lógica da avaliação de políticas no Guia do Usuário do IAM.

Como Serviços da AWS funcionam com o IAM

Para obter uma visão geral de como Serviços da AWS funcionam com a maioria dos atributos do IAM, consulte Serviços da AWS compatíveis com o IAM no Guia do usuário do IAM.

Para saber como usar um AWS service (Serviço da AWS) específico com o IAM, consulte a seção de segurança do Guia do usuário do serviço relevante.

Solução de problemas de identidade e acesso do AWS

Use as seguintes informações para ajudar a diagnosticar e corrigir problemas comuns que podem ser encontrados ao trabalhar com o e o IAM.AWS

Tópicos

- Não tenho autorização para executar uma ação no AWS
- Não estou autorizado a executar iam:PassRole
- Quero permitir que as pessoas fora da minha Conta da AWS acessem meus recursos do AWS

Não tenho autorização para executar uma ação no AWS

Se você receber uma mensagem de erro informando que não tem autorização para executar uma ação, suas políticas deverão ser atualizadas para permitir que você realize a ação.

O erro do exemplo a seguir ocorre quando o usuário do IAM `mateojackson` tenta usar o console para visualizar detalhes sobre um atributo *`my-example-widget`* fictício, mas não tem as permissões `awes:`*`GetWidget`* fictícias.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 awes:GetWidget on resource: my-example-widget
```

Nesse caso, a política do usuário `mateojackson` deve ser atualizada para permitir o acesso ao recurso *`my-example-widget`* usando a ação `awes:`*`GetWidget`*.

Se você precisar de ajuda, entre em contato com seu administrador AWS. Seu administrador é a pessoa que forneceu suas credenciais de login.

Não estou autorizado a executar iam:PassRole

Se você receber uma mensagem de erro informando que não está autorizado a executar a ação `iam:PassRole`, as suas políticas devem ser atualizadas para permitir que você passe uma função para o AWS.

Alguns Serviços da AWS permitem que você passe uma função existente para o serviço, em vez de criar uma nova função de serviço ou função vinculada ao serviço. Para fazê-lo, você deve ter permissões para passar o perfil para o serviço.

O exemplo de erro a seguir ocorre quando uma usuária do IAM chamada `marymajor` tenta utilizar o console para executar uma ação no AWS. No entanto, a ação exige que o serviço tenha permissões concedidas por um perfil de serviço. Mary não tem permissões para passar o perfil para o serviço.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

Nesse caso, as políticas de Mary devem ser atualizadas para permitir que ela realize a ação `iam:PassRole`.

Se você precisar de ajuda, entre em contato com seu administrador AWS. Seu administrador é a pessoa que forneceu suas credenciais de login.

Quero permitir que as pessoas fora da minha Conta da AWS acessem meus recursos do AWS

Você pode criar uma função que os usuários de outras contas ou pessoas fora da sua organização podem usar para acessar seus recursos. Você pode especificar quem é confiável para assumir o perfil. Para serviços que oferecem compatibilidade com políticas baseadas em recursos ou listas de controle de acesso (ACLs), você pode usar essas políticas para conceder às pessoas acesso aos seus recursos.

Saiba mais consultando o seguinte:

- Para saber se o AWS oferece compatibilidade com esses atributos, consulte Como Serviços da AWS funcionam com o IAM.
- Saiba como conceder acesso a seus recursos em todos os Contas da AWS pertencentes a você, consulte Fornecendo Acesso a um Usuário do IAM em Outro Conta da AWS Pertencente a Você no Guia de Usuário do IAM.
- Para saber como conceder acesso a seus recursos para terceiros Contas da AWS, consulte Fornecimento de acesso a Contas da AWS pertencentes a terceiros no Guia do usuário do IAM.
- Para saber como conceder acesso por meio da federação de identidades, consulte Conceder acesso a usuários autenticados externamente (federação de identidades) no Guia do usuário do IAM.
- Para saber a diferença entre o uso de perfis e políticas baseadas em recurso para acesso entre contas, consulte Acesso a recursos entre contas no IAM no Guia do usuário do IAM.

Validação de conformidade para este produto ou serviço da AWS

Para saber se um AWS service (Serviço da AWS) está no escopo de programas de conformidade específicos, consulte Serviços da AWS no escopo por programa de conformidade e selecione o programa de conformidade em que você está interessado. Para obter informações gerais, consulte Programas de Conformidade da AWS.

É possível fazer download de relatórios de auditoria de terceiros usando o AWS Artifact. Para obter mais informações, consulte Baixar relatórios no AWS Artifact.

Sua responsabilidade de conformidade ao usar o Serviços da AWS é determinada pela confidencialidade dos seus dados, pelos objetivos de conformidade da sua empresa e pelos

regulamentos e leis aplicáveis. A AWS fornece os seguintes recursos para ajudar com a conformidade:

- Guias de início rápido de segurança e conformidade: estes guias de implantação discutem considerações sobre arquitetura e fornecem as etapas para a implantação de ambientes de linha de base focados em segurança e conformidade na AWS.
- Arquitetura para segurança e conformidade com HIPAA no Amazon Web Services: esse whitepaper descreve como as empresas podem usar a AWS para criar aplicações adequadas aos padrões HIPAA.

> **Note**
>
> Nem todos os Serviços da AWS estão qualificados pela HIPAA. Para mais informações, consulte a Referência dos serviços qualificados pela HIPAA.

- Recursos de Conformidade da AWS: essa coleção de manuais e guias pode ser aplicada ao seu setor e local.
- Guias de conformidade do cliente da AWS: entenda o modelo de responsabilidade compartilhada sob a ótica da conformidade. Os guias resumem as práticas recomendadas para proteção de Serviços da AWS e mapeiam as diretrizes para controles de segurança em várias estruturas (incluindo o Instituto Nacional de Padrões e Tecnologia (NIST), o Conselho de Padrões de Segurança do Setor de Cartões de Pagamento (PCI) e a Organização Internacional de Padronização (ISO)).
- Avaliar recursos com regras no Guia do desenvolvedor do AWS Config: o serviço AWS Config avalia como as configurações de recursos estão em conformidade com práticas internas, diretrizes do setor e regulamentos.
- AWS Security Hub: este AWS service (Serviço da AWS) fornece uma visão abrangente do seu estado de segurança na AWS. O Security Hub usa controles de segurança para avaliar os recursos da AWS e verificar a conformidade com os padrões e as práticas recomendadas do setor de segurança. Para obter uma lista dos serviços e controles aceitos, consulte a Referência de controles do Security Hub.
- Amazon GuardDuty: este AWS service (Serviço da AWS) detecta possíveis ameaças às suas Contas da AWS, workloads, contêineres e dados ao monitorar o ambiente em busca de atividades suspeitas e maliciosas. O GuardDuty pode ajudar você a atender a diversos requisitos de conformidade, como o PCI DSS, com o cumprimento dos requisitos de detecção de intrusões requeridos por determinadas estruturas de conformidade.

- AWS Audit Manager: esse AWS service (Serviço da AWS) ajuda a auditar continuamente seu uso da AWS para simplificar a forma como você gerencia os riscos e a conformidade com regulamentos e padrões do setor.

Esse produto ou serviço da AWS segue o modelo de responsabilidade compartilhada por meio dos serviços específicos da Amazon Web Services (AWS) compatíveis. Para obter informações de segurança sobre o serviço da AWS, consulte a página de documentação de segurança do serviço da AWS e Serviços da AWS que estão no escopo dos esforços de conformidade da AWS por programa de conformidade.

Resiliência para este produto ou serviço da AWS

A infraestrutura global da AWS é criada com base em Regiões da AWS e zonas de disponibilidade.

As Regiões da AWS fornecem várias zonas de disponibilidade separadas e isoladas fisicamente, que são conectadas com baixa latência, throughputs elevadas e redes altamente redundantes.

Com as zonas de disponibilidade, é possível projetar e operar aplicativos e bancos de dados que automaticamente executam o failover entre as zonas sem interrupção. As zonas de disponibilidade são altamente disponíveis, tolerantes a falhas e escaláveis que uma ou várias infraestruturas de datacenters tradicionais.

Para mais informações sobre regiões e zonas de disponibilidade da AWS, consulte Infraestrutura global da AWS.

Esse produto ou serviço da AWS segue o modelo de responsabilidade compartilhada por meio dos serviços específicos da Amazon Web Services (AWS) compatíveis. Para obter informações de segurança sobre o serviço da AWS, consulte a página de documentação de segurança do serviço da AWS e Serviços da AWS que estão no escopo dos esforços de conformidade da AWS por programa de conformidade.

Segurança da infraestrutura para esse produto ou serviço da AWS

Esse produto ou serviço da AWS usa serviços gerenciados e, portanto, é protegido pela segurança de rede global da AWS. Para obter informações sobre serviços de segurança da AWS e como a AWS protege a infraestrutura, consulte Segurança na Nuvem AWS. Para projetar seu ambiente da AWS usando as práticas recomendadas de segurança da infraestrutura, consulte Proteção de Infraestrutura em Pilar de Segurança: AWS Well-Architected Framework.

Você usa chamadas de API publicadas pela AWS para acessar esse produto ou serviço da AWS pela rede. Os clientes devem oferecer suporte para:

- Transport Layer Security (TLS). Exigimos TLS 1.2 e recomendamos TLS 1.3.
- Conjuntos de criptografia com perfect forward secrecy (PFS) como DHE (Ephemeral Diffie-Hellman) ou ECDHE (Ephemeral Elliptic Curve Diffie-Hellman). A maioria dos sistemas modernos, como Java 7 e versões posteriores, comporta esses modos.

Além disso, as solicitações devem ser assinadas usando um ID da chave de acesso e uma chave de acesso secreta associada a uma entidade principal do IAM. Ou você pode usar o AWS Security Token Service (AWS STS) para gerar credenciais de segurança temporárias para assinar solicitações.

Esse produto ou serviço da AWS segue o modelo de responsabilidade compartilhada por meio dos serviços específicos da Amazon Web Services (AWS) compatíveis. Para obter informações de segurança sobre o serviço da AWS, consulte a página de documentação de segurança do serviço da AWS e Serviços da AWS que estão no escopo dos esforços de conformidade da AWS por programa de conformidade.

Aplicar uma versão mínima do TLS

> **Important**
>
> O AWS SDK for JavaScript v2 negocia automaticamente a versão TLS de nível mais alto suportada por um determinado endpoint de serviço da AWS. Opcionalmente, você pode aplicar uma versão mínima do TLS exigida pelo seu aplicativo, como o TLS 1.2 ou 1.3, mas observe que o TLS 1.3 não é suportado por alguns endpoints de serviço AWS, portanto, algumas chamadas podem falhar se você aplicar o TLS 1.3.

Para adicionar maior segurança ao se comunicar com serviços da AWS, configure o AWS SDK for JavaScript para usar o TLS 1.2 ou posterior.

O Transport Layer Security (TLS) é um protocolo usado por navegadores da web e outros aplicativos para garantir a privacidade e a integridade dos dados trocados por meio de uma rede.

Verificar e impor o TLS no Node.js

Quando você usa o AWS SDK for JavaScript com Node.js, a camada de segurança subjacente do Node.js é usada para definir a versão do TLS.

O Node.js 12.0.0 e posterior usam uma versão mínima do OpenSSL 1.1.1b, que oferece suporte ao TLS 1.3. O AWS SDK for JavaScript v3 usa como padrão o TLS 1.3 quando disponível, mas usa como padrão uma versão inferior, se necessário.

Verificar a versão do OpenSSL e do TLS

Para obter a versão do OpenSSL usada pelo Node.js no seu computador, execute o comando a seguir.

```
node -p process.versions
```

A versão do OpenSSL na lista é a versão usada pelo Node.js, como mostrado no exemplo a seguir.

```
openssl: '1.1.1b'
```

Para obter a versão do TLS usada pelo Node.js no seu computador, inicie o shell do Node e execute os comandos a seguir, na ordem.

```
> var tls = require("tls");
> var tlsSocket = new tls.TLSSocket();
> tlsSocket.getProtocol();
```

O último comando gera a versão do TLS, como mostrado no exemplo a seguir.

```
'TLSv1.3'
```

O padrão do Node.js é usar essa versão do TLS e tentará negociar outra versão do TLS se uma chamada não for bem-sucedida.

Impor uma versão mínima do TLS

O Node.js negocia uma versão do TLS quando uma chamada falha. Você pode impor a versão mínima permitida do TLS durante essa negociação, seja ao executar um script a partir da linha de comando ou por solicitação em seu código JavaScript.

Para especificar a versão mínima do TLS a partir da linha de comando, você deve usar o Node.js versão 11.0.0 ou posterior. Para instalar uma versão específica do Node.js, primeiro instale o Gerenciador de versão do Node (nvm) usando as etapas encontradas em Instalação e atualização do Gerenciador de versão do Node. Execute os comandos a seguir para instalar e usar uma versão específica do Node.js.

```
nvm install 11
nvm use 11
```

Enforcing TLS 1.2

Para impor que o TLS 1.2 seja a versão mínima permitida, especifique o argumento `--tls-min-v1.2` ao executar o script, como mostrado no exemplo a seguir.

```
node --tls-min-v1.2 yourScript.js
```

Para especificar a versão mínima permitida do TLS para uma solicitação específica em seu código JavaScript, use o parâmetro `httpOptions` para especificar o protocolo, como mostrado no exemplo a seguir.

```
const https = require("https");
const {NodeHttpHandler} = require("@aws-sdk/node-http-handler");
const {DynamoDBClient} = require("@aws-sdk/client-dynamodb");

const client = new DynamoDBClient({
    region: "us-west-2",
    requestHandler: new NodeHttpHandler({
        httpsAgent: new https.Agent(
            {
                secureProtocol: 'TLSv1_2_method'
            }
        )
    })
});
```

Enforcing TLS 1.3

Para impor que o TLS 1.3 seja a versão mínima permitida, especifique o argumento `--tls-min-v1.3` ao executar o script, como mostrado no exemplo a seguir.

```
node --tls-min-v1.3 yourScript.js
```

Para especificar a versão mínima permitida do TLS para uma solicitação específica em seu código JavaScript, use o parâmetro `httpOptions` para especificar o protocolo, como mostrado no exemplo a seguir.

```
const https = require("https");
const {NodeHttpHandler} = require("@aws-sdk/node-http-handler");
const {DynamoDBClient} = require("@aws-sdk/client-dynamodb");

const client = new DynamoDBClient({
    region: "us-west-2",
    requestHandler: new NodeHttpHandler({
        httpsAgent: new https.Agent(
            {
                secureProtocol: 'TLSv1_3_method'
            }
        )
    })
});
```

Verificar e impor o TLS em um script de navegador

Quando você usa o SDK para JavaScript em um script de navegador, as configurações do navegador controlam a versão do TLS que é usada. A versão do TLS usada pelo navegador não pode ser descoberta nem definida por script e deve ser configurada pelo usuário. Para verificar e impor a versão do TLS usada em um script de navegador, consulte as instruções para seu navegador específico.

Microsoft Internet Explorer

1. Abra o Internet Explorer.
2. Na barra de menu, escolha a guia Ferramentas - Opções da Internet - Avançado.
3. Role para baixo até a categoria Segurança e marque manualmente a caixa de opção Usar TLS 1.2.
4. Clique em OK.
5. Feche o navegador e reinicie o Internet Explorer.

Microsoft Edge

1. Na caixa de pesquisa do menu do Windows, digite *Opções da Internet*.
2. Em Melhor correspondência, clique em Opções da Internet.
3. Na janela Propriedades da Internet, na guia Avançado, role para baixo até a seção Segurança.
4. Marque a caixa de seleção Usar TLS 1.2.
5. Clique em OK.

Google Chrome

1. Abra o Google Chrome.
2. Clique em Alt F e selecione Configurações.
3. Role para baixo e selecione Mostrar configurações avançadas....
4. Role para baixo até a seção Sistema e clique em Abrir configurações de proxy....
5. Selecione a guia Avançado.
6. Role para baixo até a categoria Segurança e marque manualmente a caixa de opção Usar TLS 1.2.
7. Clique em OK.
8. Feche seu navegador e reinicie o Google Chrome.

Mozilla Firefox

1. Abra o Firefox.
2. Na barra de endereço, digite about:config e pressione Enter.
3. No campo Pesquisar, digite tls. Localize e clique duas vezes na entrada de security.tls.version.min.
4. Defina o valor inteiro como 3 para forçar o protocolo TLS 1.2 a ser o padrão.
5. Clique em OK.
6. Feche seu navegador e reinicie o Mozilla Firefox.

Apple Safari

Não há opções para ativar os protocolos SSL. Se você estiver usando o Safari versão 7 ou superior, o TLS 1.2 será ativado automaticamente.

Recursos adicionais

Os links a seguir fornecem recursos adicionais que você pode usar com o AWS SDK for JavaScript.

Guia de referência de AWS SDKs e ferramentas

O Guia de referência de ferramentas e SDKs da AWS também contém configurações, recursos e outros conceitos fundamentais comuns entre muitos dos SDKs da AWS.

Fórum do SDK do JavaScript

Encontre perguntas e discussões sobre questões de interesse para usuários do SDK para JavaScript no Fórum do SDK do JavaScript.

SDK do JavaScript e Guia do desenvolvedor no GitHub

Há vários repositórios do SDK para JavaScript no GitHub.

- O SDK para JavaScript atual está disponível no Repositório do SDK.
- O Guia do desenvolvedor do SDK para JavaScript (este documento) está disponível em formato markdown no próprio repositório de documentação.
- Parte do código de exemplo incluído neste guia está disponível no repositório de código de exemplo do SDK.

SDK do JavaScript no Gitter

Também é possível encontrar perguntas e discussões sobre o SDK para JavaScript na Comunidade do SDK do JavaScript no Gitter.

Histórico de documentos do AWS SDK for JavaScript

- Versão do SDK: consulte Referência de API do JavaScript
- Última atualização importante da documentação: 31 de março de 2023

Histórico do documento

A tabela a seguir descreve alterações importantes em cada versão do AWS SDK for JavaScript após maio de 2018. Para receber notificações sobre atualizações dessa documentação, inscreva-se em um feed RSS.

Alteração	Descrição	Data
Aplicar uma versão mínima do TLS	Adição de informações sobre TLS 1.3.	31 de março de 2022
Como visualizar fotos em um bucket do Amazon S3 em um navegador	Adicionado um exemplo para simplesmente visualizar fotos em álbuns de fotos existentes.	13 de maio de 2019
Configuração de credenciais no Node.js, novas opções de carregamento de credenciais	Adicionadas informações sobre credenciais que são carregadas do provedor de credenciais do ECS ou de um processo de credenciais configurado.	25 de abril de 2019
Credenciais usando um processo de credenciais configuradas	Adicionadas informações sobre credenciais que são carregadas por meio de um processo de credenciais configuradas.	25 de abril de 2019
Conceitos básicos novos de um script de navegador	Conceitos básicos de um script de navegador foi reescrito para simplificar o exemplo e acessar o serviço	14 de julho de 2023

	para enviar texto e retornar fala sintetizada que você pode reproduzir no navegador. Consulte Conceitos básicos do script do navegador para consultar o novo conteúdo.	
Novos exemplos de código do Amazon SNS	Foram adicionados quatro novos exemplos de código Node.js para trabalhar com o Amazon SNS. Consulte Exemplos do Amazon SNS para o código de exemplo.	29 de junho de 2018
Conceitos básicos novos de Node.js	Conceitos básicos de Node.js foi reescrito para usar o código de exemplo atualizado e para dar mais detalhes sobre como criar o arquivo `package.json`, bem como o código Node.js propriamente dito. Consulte Conceitos básicos de Node.js para consultar o novo conteúdo.	4 de junho de 2018

Atualizações anteriores

A tabela a seguir descreve alterações importantes em cada versão do AWS SDK for JavaScript antes de junho de 2018.

Alteração	Descrição	Data
Novos exemplos de código do AWS Elemental MediaConvert	Foram adicionados três novos exemplos de código Node.js para trabalhar com o AWS Elemental MediaConvert. Consulte Exemplos do AWS	21 de maio de 2018

Alteração	Descrição	Data
	Elemental MediaConvert para obter o código de exemplo.	
Nova edição no botão do GitHub	O cabeçalho de cada tópico já fornece um botão que leva você à versão markdown do mesmo tópico no GitHub, de maneira que possa fornecer edições para aumentar a precisão e a integridade do guia.	21 de fevereiro de 2018
Novo tópico em endpoints personalizados	As informações foram adicionadas no formato e no uso de endpoints personalizados para fazer chamadas de API. Consulte Especificar endpoints personalizados.	20 de fevereiro de 2018
SDK para JavaScript e Guia do desenvolvedor no GitHub	O Guia do desenvolvedor do SDK para JavaScript está disponível em formato markdown no próprio repositório de documentação. Publique problemas que você gostaria que o guia abordasse ou envie solicitações pull para enviar alterações propostas.	16 de fevereiro de 2018

Alteração	Descrição	Data
Amostra da amostra de código do Amazon DynamoDB	Um novo exemplo de código Node.js para atualizar uma tabela do DynamoDB que usa o cliente de documento foi adicionado. Consulte Uso do cliente de documentos do DynamoDB para obter o código de exemplo.	14 de fevereiro de 2018
Novo tópico sobre AWS Cloud9	Um tópico que descreve como usar o AWS Cloud9 para desenvolver e depurar o navegador e o código Node.js foi adicionado. Consulte Como usar o AWS Cloud9 com o AWS SDK for JavaScript.	5 de fevereiro de 2018
Novo tópico sobre registro em log do SDK	Um tópico que descreve como registrar chamadas de API feitas com o SDK para JavaScript foi adicionado, inclusive informações sobre como usar um registrador de terceiros. Consulte Registrar em log as chamadas a AWS SDK for JavaScript.	5 de fevereiro de 2018
Tópico atualizado sobre a definição da região	O tópico que descreve como definir a região usada com o SDK foi atualizad o e expandido, incluindo informações sobre a ordem de precedência para definir a região. Consulte Definir a região da AWS.	12 de dezembro de 2017

Alteração	Descrição	Data
Novo exemplos de código do Amazon SES	A seção com exemplos de código do SDK foi atualizad a para incluir cinco novos exemplos para trabalhar com o Amazon SES. Para obter mais informações sobre esses exemplos de código, consulte Exemplos do Amazon Simple Email Service.	9 de novembro de 2017

Alteração	Descrição	Data
Melhorias de usabilidade	Com base em testes de usabilidade recentes, uma série de alterações foram feitas para melhorar a usabilidade da documentação. • Os exemplos de código são identificados mais claramente e como segmentados para execução em navegador ou Node.js. • Os links TOC deixam de ir imediatamente para outro conteúdo da web, inclusive a referência da API. • Inclui mais vinculação na seção Conceitos básicos para obter detalhes sobre como obter credenciais da AWS. • Fornece mais informações sobre recursos de Node.js comuns necessários para usar o SDK. Para ter mais informações, consulte Considerações sobre Node.js.	9 de agosto de 2017

Alteração	Descrição	Data
Novos exemplos de código do DynamoDB	A seção com exemplos de código do SDK foi atualizada para reescrever os dois exemplos anteriores, bem como adicionar três novos exemplos para trabalhar com o DynamoDB. Para obter mais informações sobre esses exemplos de código, consulte Exemplos do Amazon DynamoDB.	21 de junho de 2017
Novos exemplos de código do IAM	A seção com exemplos de código do SDK foi atualizada para incluir cinco novos exemplos para trabalhar com o IAM. Para obter mais informações sobre esses exemplos de código, consulte Exemplos do IAM AWS.	23 de dezembro de 2016
Novos exemplos de códigos do CloudWatch e do Amazon SQS	A seção com exemplos de código do SDK foi atualizada para incluir novos exemplos para trabalhar com o CloudWatch e com o Amazon SQS. Para obter mais informações sobre esses exemplos de código, consulte Exemplos do Amazon CloudWatch e Exemplos do Amazon SQS.	20 de dezembro de 2016

Alteração	Descrição	Data
Novos exemplos de código do Amazon EC2	A seção com exemplos de código do SDK foi atualizad a para incluir cinco novos exemplos para trabalhar com o Amazon EC2. Para obter mais informações sobre esses exemplos de código, consulte Exemplos do Amazon EC2.	15 de dezembro de 2016
Lista de navegadores compatíveis mais visível	A lista de navegadores compatíveis com o SDK for JavaScript, que estava no tópico sobre Pré-requisitos, agora tem o próprio tópico para deixá-la mais visível no sumário.	16 de novembro de 2016
Publicação inicial do novo Guia do desenvolvedor	O Guia do desenvolvedor anterior já está obsoleto. O novo Guia do desenvolvedor foi reorganizado para facilitar a localização das informações. Quando cenários de Node.js ou JavaScript de navegador apresentam considerações especiais, eles são identific ados como apropriados. O guia também fornece exemplos de código adicionai s mais bem organizados para facilitar e agilizar a localizaç ão.	28 de outubro de 2016