



Guía del desarrollador de SDK v2

AWS SDK para JavaScript



AWS SDK para JavaScript: Guía del desarrollador de SDK v2

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

Las marcas comerciales y la imagen comercial de Amazon no se pueden utilizar en relación con ningún producto o servicio que no sea de Amazon, de ninguna manera que pueda causar confusión entre los clientes y que menosprecie o desacredite a Amazon. Todas las demás marcas registradas que no son propiedad de Amazon son propiedad de sus respectivos propietarios, que pueden o no estar afiliados, conectados o patrocinados por Amazon.

Table of Contents

.....	ix
¿Qué es la AWS SDK para JavaScript?	1
Mantenimiento y compatibilidad de las versiones principales del SDK	1
Uso del SDK con Node.js	2
Uso del SDK con AWS Amplify	2
Uso del SDK con navegadores web	2
Casos de uso común	3
Acerca de los ejemplos	3
Introducción	4
Introducción a los script de navegador	4
El escenario	4
Paso 1: Creación de un grupo de identidades en Amazon Cognito	5
Paso 2: Añadir una política al rol de IAM creado	6
Paso 3: Creación de la página HTML	7
Paso 4: Escritura del script de navegador	8
Paso 5: Ejecución de la muestra	10
Muestra completa	10
Mejoras posibles	11
Introducción a Node.js	12
El escenario	12
Tareas previas necesarias	12
Paso 1: Instalar el SDK y las dependencias	13
Paso 2: Configurar sus credenciales	14
Paso 3: Crear el JSON del paquete para el proyecto	14
Paso 4: Escriba el código de Node.js	15
Paso 5: Ejecución de la muestra	16
Configuración del SDK para JavaScript	18
Requisitos previos	18
Configuración de un entorno Node.js de AWS	19
Navegadores web compatibles	19
Instalación del SDK	20
Instalación mediante Bower	21
Carga del SDK	21
Actualización desde la versión 1	23

Conversión automática de tipos de marca temporal y de Base64 en la entrada y la salida	23
Cambio de response.data.RequestId a response.requestId	24
Elementos de encapsulador expuestos	24
Propiedades de cliente eliminadas	29
Configurar el SDK para JavaScript	30
Uso del objeto de configuración global	30
Ajuste de la configuración global	31
Configuración según el servicio	33
Datos de configuración inmutables	33
Configuración de la región de AWS	34
En un constructor de clase de cliente	34
Uso del objeto de configuración global	34
Uso de una variable de entorno	34
Uso de un archivo de configuración compartido	34
Orden de prioridad para establecer la región	35
Especificación de puntos de enlace personalizados	36
Formato de cadena de punto de enlace	36
Puntos de enlace para la región ap-noreste-3	36
Puntos de conexión para MediaConvert	36
Autenticación de SDK con AWS	37
Iniciar una sesión en el portal de acceso a AWS	38
Información adicional de autenticación	39
Configuración de las credenciales	40
Prácticas recomendadas para las credenciales	40
Configuración de credenciales en Node.js	41
Configuración de credenciales en un navegador web	46
Bloqueo de versiones de la API	56
Obtener versiones de la API	57
Consideraciones de Node.js	57
Uso de módulos Node.js integrados	57
Uso de paquetes NPM	58
Configuración de maxSockets en Node.js	59
Reutilización de conexiones con Keep-Alive en Node.js	60
Configuración de proxies para Node.js	61
Registro de paquetes de certificados en Node.js	62
Consideraciones sobre los scripts de navegador	62

Creación del SDK para navegadores	63
Cross-Origin Resource Sharing (CORS, Uso compartido de recursos entre orígenes)	66
Agrupación con Webpack	70
Instalación de Webpack	70
Configuración de Webpack	71
Ejecución de Webpack	72
Uso del grupo de Webpack	73
Importación de servicios individuales	73
Agrupación para Node.js	74
Trabajar con los servicios	76
Creación y llamada a objetos de servicio	77
Necesidad de uso de servicios individuales	78
Creación de objetos de servicio	79
Bloqueo de la versión de API de un objeto de servicio	80
Especificación de parámetros de objetos de servicio	80
Registro de llamadas del AWS SDK para JavaScript	81
Uso de registradores de terceros	81
Llamadas asíncronas a servicios	82
Administración de llamadas asíncronas	83
Uso de una función de devolución de llamada	84
Uso de un objeto de un agente de escucha de eventos de objetos de solicitud	86
Uso de async/await	91
Uso de promesas	92
Uso del objeto de respuesta	94
Acceso a los datos devueltos en el objeto de respuesta	95
Paginación por los datos devueltos	96
Acceso a información de error desde un objeto de respuesta	96
Acceso al objeto de solicitud de origen	97
Uso de JSON	97
Parámetros de JSON como objeto de servicio	98
Devolución de datos como JSON	99
Reintentos	100
Comportamiento de reintentos basado en el retroceso exponencial	100
Ejemplos de código SDK para JavaScript	103
Ejemplos de Amazon CloudWatch	103
Creación de alarmas de Amazon CloudWatch	104

Uso de acciones de alarma en Amazon CloudWatch	108
Obtención de métricas de Amazon CloudWatch	113
Envío de eventos a Amazon CloudWatch Events	116
Uso de filtros de suscripción en Registros de Amazon CloudWatch	121
Ejemplos de Amazon DynamoDB	126
Creación y uso de tablas en DynamoDB	127
Lectura y escritura de un único elemento en DynamoDB	132
Lectura y escritura de elementos en lotes en DynamoDB	136
Consulta y examen de una tabla de DynamoDB	139
Uso del cliente de documentos de DynamoDB	143
Ejemplos de Amazon EC2	149
Creación de una instancia de Amazon EC2	150
Administración de instancias de Amazon EC2	152
Uso de pares de claves de Amazon EC2	159
Uso de las regiones y las zonas de disponibilidad de Amazon EC2	162
Trabajar con grupos de seguridad en Amazon EC2	164
Uso de direcciones IP elásticas en Amazon EC2	169
Ejemplos de MediaConvert	173
Creación y administración de trabajos	174
Uso de plantillas de trabajos	181
Ejemplos de AWS IAM	190
Administración de usuarios de IAM	191
Uso de políticas de IAM	196
Administración de las claves de acceso de IAM	202
Uso de certificados de servidor de IAM	207
Administración de alias de cuenta de IAM	211
Ejemplo de Amazon Kinesis	214
Captura de la progresión del desplazamiento en una página web con Amazon Kinesis	215
Ejemplos de Amazon S3	222
Ejemplos de Amazon S3 en un navegador	223
Ejemplos de Node.js de Amazon S3	252
Ejemplos de Amazon SES	273
Administración de identidades	274
Uso de plantillas de correo electrónico	279
Envío de correo electrónico con Amazon SES	285
Uso de filtros de direcciones IP	292

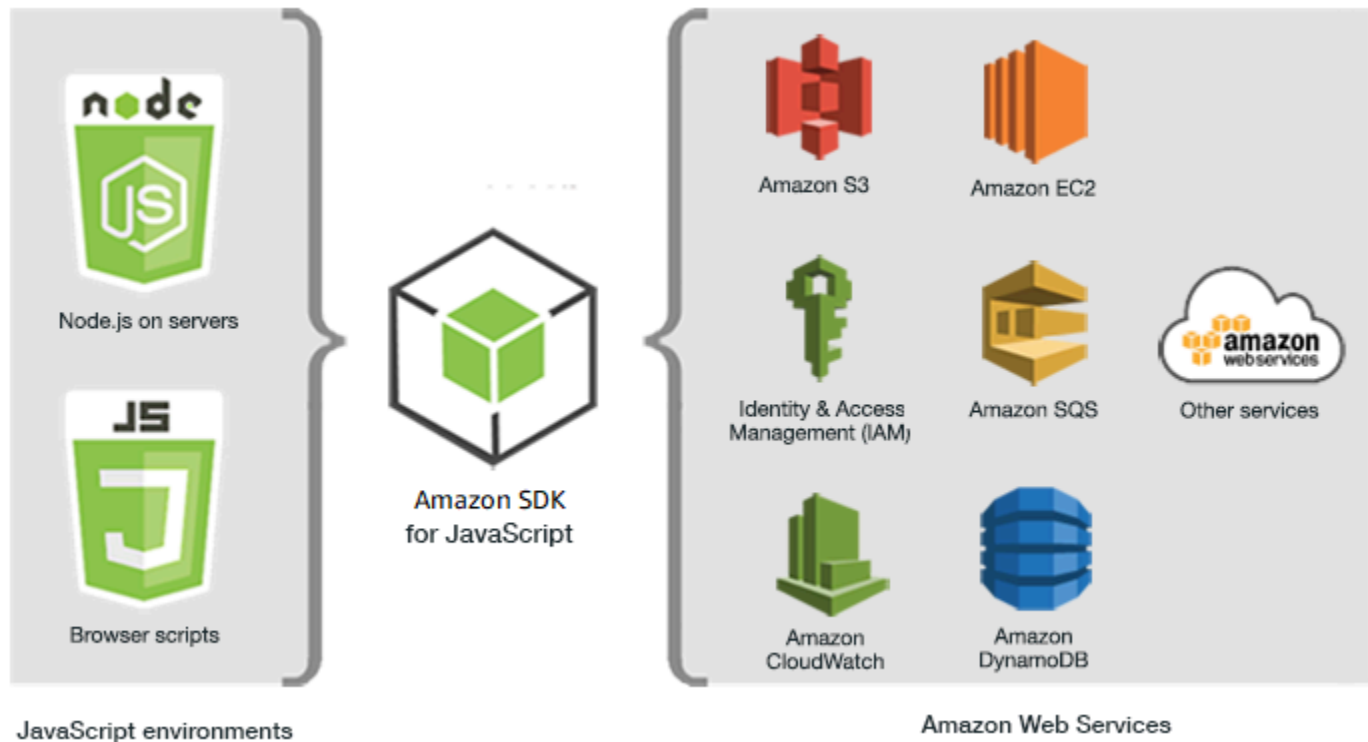
Uso de reglas de recepción	296
Ejemplos de Amazon SNS	302
Administración de temas	303
Publicación de mensajes en un tema	308
Administración de suscripciones	310
Envío de mensajes SMS	317
Ejemplos de Amazon SQS	324
Uso de colas en Amazon SQS	324
Envío y recepción de mensajes en Amazon SQS	329
Administración del tiempo de espera de visibilidad en Amazon SQS	333
Habilitación del sondeo largo en Amazon SQS	335
Uso de colas de mensajes fallidos en Amazon SQS	339
Tutoriales	342
Tutorial: Configuración de Node.js en una instancia de Amazon EC2	342
Requisitos previos	342
Procedimiento	342
Creación de una imagen de Amazon Machine (AMI)	344
Activos relacionados	344
La referencia de la API y el registro de cambios	345
Registro de cambios del SDK en GitHub	345
Migración a la versión 3	346
Seguridad	347
Protección de datos	347
Gestión de identidad y acceso	349
Público	349
Autenticación con identidades	350
Administración del acceso con políticas	351
Cómo funcionan los Servicios de AWS con IAM	353
Solución de problemas de identidades y accesos en AWS	353
Validación de la conformidad	355
Resiliencia	356
Seguridad de infraestructuras	356
Aplicación de una versión mínima de TLS	357
Verificar y aplicar TLS en Node.js	358
Verificar y aplicar TLS en un script de navegador	360
Recursos adicionales	363

Guía de referencia de las herramientas y los SDK de AWS	363
Foro de SDK para JavaScript	363
SDK para JavaScript y guía para desarrolladores en GitHub	363
SDK de JavaScript en Gitter	363
Historial de documento	364
Historial de documentos	364
Actualizaciones anteriores	366

La versión 2 de AWS SDK para JavaScript ha llegado al final del soporte. Se recomienda que migre a [AWS SDK para JavaScript v3](#). Para ver detalles e información adicionales sobre cómo realizar la migración, consulte este [anuncio](#).

¿Qué es la AWS SDK para JavaScript?

[AWS SDK para JavaScript](#) proporciona una API de JavaScript para los servicios de AWS. Puede usar la API de JavaScript para crear bibliotecas o aplicaciones para [Node.js](#) o el navegador.



No todos los servicios están disponibles de forma inmediata en el SDK. Para informarse de qué servicios son actualmente compatibles con AWS SDK para JavaScript, consulte <https://github.com/aws/aws-sdk-js/blob/master/SERVICES.md>. Para obtener más información acerca del SDK para JavaScript en GitHub, consulte [Recursos adicionales](#).

Mantenimiento y compatibilidad de las versiones principales del SDK

Para obtener información sobre el mantenimiento y la compatibilidad con las principales versiones del SDK y sus dependencias subyacentes, consulte lo siguiente en la [Guía de Referencia de SDK y herramientas de AWS](#):

- [Política de mantenimiento de SDK y herramientas de AWS](#)
- [Matriz de compatibilidad para versiones de SDK y herramientas de AWS](#)

Uso del SDK con Node.js

Node.js es una ejecución entre plataformas que permite ejecutar aplicaciones JavaScript de lado de servidor. Puede configurar Node.js en una instancia de Amazon EC2 para que se ejecute en un servidor. También puede utilizar Node.js para escribir funciones de Lambda bajo demanda.

El uso del SDK con Node.js difiere de la forma en que se usa con JavaScript en un navegador web. La diferencia estriba en la forma en que carga el SDK y cómo obtiene las credenciales necesarias para tener acceso a servicios web específicos. Cuando el uso de determinadas API concretas varía según si se trabaja con Node.js o el navegador, dichas diferencias se destacan.

Uso del SDK con AWS Amplify

Para las aplicaciones web, móviles e híbridas basadas en navegador, también puede utilizar la [biblioteca de AWS Amplify en GitHub](#), que amplía el SDK para JavaScript al proporcionar una interfaz declarativa.

Note

Es posible que algunos marcos, como AWS Amplify, no ofrezcan la misma compatibilidad de navegadores que el SDK para JavaScript. Consulte la documentación de los marcos en cuestión para ver información detallada.

Uso del SDK con navegadores web

Todos los principales navegadores web son compatibles con la ejecución de JavaScript. El código JavaScript que se ejecuta en un navegador web suele denominarse JavaScript de lado de cliente.

El uso del SDK para JavaScript con un navegador web difiere de la forma en que se usa para Node.js. La diferencia estriba en la forma en que carga el SDK y cómo obtiene las credenciales necesarias para tener acceso a servicios web específicos. Cuando el uso de determinadas API concretas varía según si se trabaja con Node.js o el navegador, dichas diferencias se destacan.

Para obtener una lista de los navegadores compatibles con AWS SDK for JavaScript, consulte [Navegadores web compatibles](#).

Casos de uso común

El uso de SDK para Javascript en scripts de navegador permite observar una serie de casos de uso convincentes. A continuación se muestran varias ideas de cosas que puede crear en una aplicación de navegador usando el SDK para JavaScript para obtener acceso a diferentes servicios web.

- Crear una consola personalizada a los servicios de AWS en la que tenga acceso y pueda combinar características entre regiones y servicios para atender mejor las necesidades de su organización o proyecto.
- Usar Amazon Cognito para habilitar el acceso de usuarios autenticados a sus aplicaciones y sitios web de navegador, incluido el uso de la autenticación de terceros de Facebook y otros.
- Usar Amazon Kinesis para procesar flujos de clics u otros datos de marketing en tiempo real.
- Usar Amazon DynamoDB para la persistencia de datos sin servidor, como las preferencias de usuarios individuales para los visitantes de su sitio web o usuarios de la aplicación.
- Usar Lambda para encapsular la lógica de propietario que puede invocar desde scripts de navegador sin tener que descargar ni revelar su propiedad intelectual a los usuarios.

Acerca de los ejemplos

Puede buscar ejemplos del SDK para JavaScript en la [biblioteca de ejemplos de código de AWS](#).

Introducción a la AWS SDK para JavaScript

El AWS SDK para JavaScript proporciona acceso a servicios web en scripts de navegador o en Node.js. Esta sección dispone de dos ejercicios de introducción donde se muestra cómo trabajar con el SDK para JavaScript en cada uno de estos entornos de JavaScript.

Temas

- [Introducción a los script de navegador](#)
- [Introducción a Node.js](#)

Introducción a los script de navegador



Este ejemplo de script de navegador le muestra:

- Cómo obtener acceso a los servicios de AWS desde un script de navegador utilizando identidades de Amazon Cognito.
- Cómo convertir texto en voz sintetizada utilizando Amazon Polly.
- Cómo utilizar un objeto prefirmado para crear una URL prefirmada.

El escenario

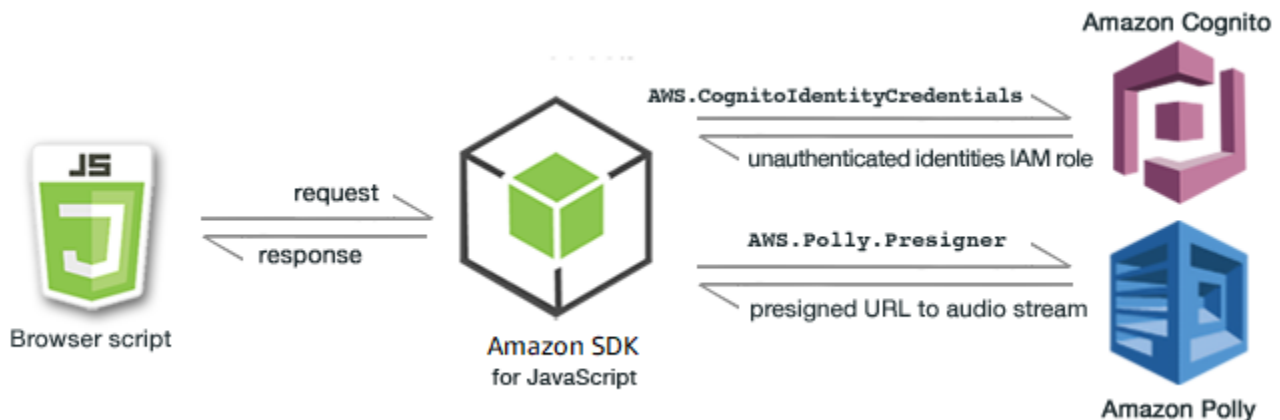
Amazon Polly es un servicio en la nube que convierte el texto en un segmento hablado muy realista. Puede utilizar Amazon Polly para desarrollar aplicaciones que aumenten la participación y mejoren la accesibilidad. Amazon Polly está disponible en varios idiomas e incluye una variedad de voces realistas. Para obtener más información sobre Amazon Polly, consulte la [Guía para desarrolladores de Amazon Polly](#).

En este ejemplo se muestra cómo configurar y ejecutar un sencillo script de navegador que toma texto el que se introduce, envía dicho texto a Amazon Polly y luego devuelve la URL del audio sintetizado del texto para que se reproduzca. El script de navegador utiliza Identidad de Amazon

Cognito para proporcionar las credenciales necesarias que dan acceso a los servicios de AWS. Verá los patrones básicos de carga y uso del SDK para JavaScript en scripts de navegador.

Note

La reproducción de voz sintetizada de este ejemplo depende de si se ejecuta en navegador compatible con audio HTML 5.



El script de navegador usa el SDK para JavaScript para sintetizar texto mediante las API siguientes:

- [AWS.CognitoIdentityCredentials](#) Constructor de
- [AWS.Polly.Presigner](#) Constructor de
- [getSynthesizeSpeechUrl](#)

Paso 1: Creación de un grupo de identidades en Amazon Cognito

En este ejercicio, deberá crear y utilizar un grupo de identidades de Amazon Cognito para proporcionar acceso sin autenticar a su script de navegador para el servicio de Amazon Polly. Al crear un grupo de identidades también creará dos roles de IAM, uno para admitir usuarios autenticados mediante un proveedor de identidades y otro para admitir usuarios invitados no autenticados.

En este ejercicio, vamos a trabajar con el rol de usuario sin autenticar para concentrarnos en la tarea. Puede integrar la compatibilidad con un proveedor de identidades y los usuarios autenticados más adelante. Para obtener más información sobre cómo añadir un grupo de identidades de Amazon

Cognito, consulte [Tutorial: Crear un grupo de identidades](#) en la Guía para desarrolladores de Amazon Cognito.

Creación de un grupo de identidades en Amazon Cognito

1. Inicie sesión en Consola de administración de AWS y abra la consola de Amazon Cognito desde <https://console.aws.amazon.com/cognito/>.
2. En el panel de navegación de la izquierda, elija Grupos de identidades.
3. Elija Crear grupo de identidades.
4. En Configurar la confianza del grupo de identidades, elija Acceso de invitado para la autenticación de usuario.
5. En Configurar permisos, elija Crear un nuevo rol de IAM e introduzca un nombre (por ejemplo, GetStartedRole) en el nombre del rol de IAM.
6. En Configurar propiedades, introduzca un nombre (por ejemplo, getStartedPool) en el nombre del grupo de identidades.
7. En Revisar y crear, confirme las selecciones que realizó para el nuevo grupo de identidades. Seleccione Editar para volver al asistente y cambiar cualquier configuración. Cuando haya acabado, seleccione Crear grupo de identidades.
8. Tenga en cuenta el ID del grupo de identidades y la Región del grupo de identidades de Amazon Cognito recién creado. Necesitará estos valores para sustituir *IDENTITY_POOL_ID* y *REGION* en [Paso 4: Escritura del script de navegador](#).

Después de crear el grupo de identidades de Amazon Cognito, ya está todo listo para añadir los permisos de Amazon Polly que necesita su script del navegador.


Paso 2: Añadir una política al rol de IAM creado

Para habilitar el acceso del script del navegador a Amazon Polly para el proceso de síntesis de voz, utilice el rol de IAM sin autenticar para su grupo de identidades de Amazon Cognito. Para ello tiene que añadir una política de IAM al rol. Para obtener más información acerca de cómo modificar roles de IAM, consulte la política [Modificación de los permisos de un rol](#) en la Guía del usuario de IAM.

Para añadir una política de Amazon Polly al rol de IAM asociado a usuarios sin autenticar

1. Inicie sesión en Consola de administración de AWS y abra la consola IAM en <https://console.aws.amazon.com/iam/>.

2. En el panel de navegación izquierdo, elija Roles.
3. Elija el nombre del rol que desea modificar (por ejemplo, getStartedRole) y, a continuación, seleccione la pestaña Permisos.
4. Elija Agregar permisos y luego Adjuntar políticas.
5. En la página Agregar permisos de este rol, busque y luego seleccione la casilla de verificación de AmazonPollyReadOnly.

 Note

Puede utilizar este proceso para habilitar el acceso a cualquier servicio de AWS.

6. Elija Añadir permisos.

Después de crear el grupo de identidades de Amazon Cognito y de añadir permisos para Amazon Polly a su rol de IAM para usuarios sin autenticar, tendrá todo listo para crear la página web y el script de navegador.

Paso 3: Creación de la página HTML

La aplicación de muestra se compone de una sola página HTML que contiene la interfaz de usuario y el script de navegador. Para empezar cree un documento HTML y copie el siguiente contenido en él. La página contiene un campo de entrada y un botón, un elemento `<audio>` para reproducir la voz sintetizada y un elemento `<p>` para mostrar mensajes. (Tenga en cuenta que el ejemplo completo se muestra en la parte inferior de esta página).

Para obtener más información acerca del elemento `<audio>`, consulte [audio](#).

```
<!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>
```

Guarde el archivo HTML y asígnele el nombre `polly.html`. Una vez que haya creado la interfaz de usuario para la aplicación, ya podrá agregar el código del script de navegador que ejecuta la aplicación.

Paso 4: Escritura del script de navegador

Lo primero que debe hacer al crear el script de navegador es incluir el SDK para JavaScript añadiendo un elemento `<script>` después del elemento `<audio>` de la página. Para encontrar el `SDK_VERSION_NUMBER` actual, consulte la referencia de API para el SDK para JavaScript en la [Guía de referencia de la API de AWS SDK para JavaScript](#).

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

A continuación, añada un elemento `<script type="text/javascript">` después de la entrada del SDK. Añadirá el script de navegador a este elemento. Defina la región de AWS y las credenciales del SDK. A continuación, cree una función denominada `speakText()` que el botón invocará como controlador de eventos.

Para sintetizar voz con Amazon Polly, debe proporcionar una serie de parámetros, como el formato de sonido de la salida, la frecuencia de muestreo, el ID de la voz que se va a utilizar y el texto que debe reproducirse. Al crear inicialmente los parámetros, establezca el parámetro `Text`: en una cadena vacía; el parámetro `Text`: se definirá como el valor que recuperara del elemento `<input>` en la página web. Sustituya `IDENTITY_POOL_ID` y `REGION` en el siguiente código por los valores indicados en [Paso 1: Creación de un grupo de identidades en 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;
```

Amazon Polly devuelve voz sintetizada como una secuencia de audio. La forma más sencilla de reproducir dicho audio en un navegador consiste en hacer que Amazon Polly ponga el audio a disposición del usuario en una URL prefirmada que se puede establecer como el atributo `src` del elemento `<audio>` de la página web.

Cree un nuevo objeto de servicio `AWS.Polly`. A continuación, cree el objeto `AWS.Polly.Presigner` que utilizará para crear la URL prefirmada de la que se podrá recuperar el audio de la voz sintetizada. Debe transferir los parámetros de voz que ha definido, así como el objeto de servicio `AWS.Polly` que ha creado al constructor `AWS.Polly.Presigner`.

Después de crear el objeto `presigner`, llame al método `getSynthesizeSpeechUrl` de dicho objeto y pase los parámetros de voz. Si la operación se ejecuta correctamente, este método devuelve la dirección URL de la voz sintetizada, que a continuación puede asignar al elemento `<audio>` para la reproducción.

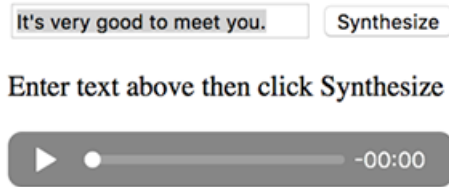
```
// Create the Polly service object and presigner object
var polly = new AWS.Polly({apiVersion: '2016-06-10'});
var signer = new AWS.Polly.Presigner(speechParams, polly)

// 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>
```

Paso 5: Ejecución de la muestra

Para ejecutar la aplicación de muestra, cargue `polly.html` en un navegador web. La presentación del navegador debe tener un aspecto similar al siguiente.



Escriba una frase que desea que se convierta en voz en el cuadro de entrada y, a continuación, elija la Synthesize (Sintetizar). Cuando el audio esté lista para reproducirse, aparecerá un mensaje. Utilice los controles del reproductor de audio para escuchar la voz sintetizada.

Muestra completa

A continuación se muestra la página HTML completa con el script de navegador. También está disponible [aquí, en 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>
```

Mejoras posibles

A continuación, se muestran diversas variaciones de esta aplicación que puede utilizar para profundizar en el uso del SDK para JavaScript en un script de navegador.

- Experimentar con otros formatos de salida de sonido.
- Añadir la opción de seleccionar cualquiera de las distintas voces que ofrece Amazon Polly.
- Integrar un proveedor de identidades como Facebook o Amazon para utilizarlo con el rol de IAM autenticado.

Introducción a Node.js



Este ejemplo de código de Node.js muestra:

- Cómo crear el manifiesto `package.json` para su proyecto.
- Cómo instalar e incluir los módulos que su proyecto utiliza.
- Cómo crear un objeto de servicio de Amazon Simple Storage Service (Amazon S3) a partir de la clase de cliente `AWS.S3`.
- Cómo crear un bucket de Amazon S3 y cargar un objeto en dicho bucket.

El escenario

En el ejemplo se muestra cómo configurar y ejecutar un módulo de Node.js sencillo que crea un bucket de Amazon S3 y luego le añade un objeto de texto.

Dado que los nombres en Amazon S3 deben ser únicos a nivel global, este ejemplo incluye un módulo de Node.js de terceros que genera un valor de ID único que puede incorporar en el nombre de bucket. Este módulo adicional se llama `uuid`.

Tareas previas necesarias

Para configurar y ejecutar este ejemplo, primero debe completar estas tareas:

- Crear un directorio de trabajo para desarrollar su módulo Node.js. Llame a este directorio `awsnodesample`. Tenga en cuenta que el directorio se debe crear en una ubicación donde las aplicaciones lo puedan actualizar. Por ejemplo, en Windows, no cree el directorio en la sección "C:\Archivos de programa".

- Instale Node.js. Para obtener más información, consulte el sitio web de [Node.js](https://nodejs.org/en/download/current/). Puede encontrar descargas de las versiones LTS y actual de Node.js para una serie de sistemas operativos en <https://nodejs.org/en/download/current/>.

Contenido

- [Paso 1: Instalar el SDK y las dependencias](#)
- [Paso 2: Configurar sus credenciales](#)
- [Paso 3: Crear el JSON del paquete para el proyecto](#)
- [Paso 4: Escriba el código de Node.js](#)
- [Paso 5: Ejecución de la muestra](#)

Paso 1: Instalar el SDK y las dependencias

El paquete del SDK para JavaScript se instala usando [npm \(el administrador de paquetes de Node.js\)](#).

En el directorio `awsnodesample` del paquete, escriba lo siguiente en la línea de comandos.

```
npm install aws-sdk
```

Este comando instala el SDK para JavaScript en su proyecto y actualiza `package.json` para incluir el SDK como dependencia de un proyecto. Puede encontrar información sobre este paquete buscando "aws-sdk" en el [sitio web de npm](#).

A continuación, instale el módulo `uuid` en el proyecto escribiendo lo siguiente en la línea de comandos, para instalar el módulo y las actualizaciones `package.json`. Para obtener más información acerca de `uuid`, consulte la página del módulo en <https://www.npmjs.com/package/uuid>.

```
npm install uuid
```

Estos paquetes y su código asociado se instalan en el subdirectorio `node_modules` del proyecto.

Para obtener más información acerca de la instalación de paquetes Node.js, consulte las secciones sobre [cómo descargar e instalar paquetes locales](#) y [cómo crear módulos Node.js](#) en el [sitio web de npm \(administrador de paquetes de Node.js\)](#). Para obtener información acerca del cómo descargar e instalar el AWS SDK para JavaScript, consulte [Instalación del SDK para JavaScript](#).

Paso 2: Configurar sus credenciales

Debe proporcionar credenciales para AWS, de modo que el SDK solo obtenga acceso a su cuenta y sus recursos. Para obtener más información sobre cómo obtener sus credenciales de cuenta, consulte [Autenticación de SDK con AWS](#).

Para almacenar esta información, le recomendamos que cree un archivo de credenciales compartidas. Para aprender a hacerlo, consulte [Carga de credenciales en Node.js desde el archivo de credenciales compartidas](#). Su archivo de credenciales debe parecerse al siguiente ejemplo.

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

Puede determinar si ha configurado las credenciales correctamente ejecutando el siguiente código con 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);
  }
});
```

Del mismo modo, si ha definido la región correctamente en el archivo `config`, puede mostrar ese valor estableciendo la variable de entorno `AWS_SDK_LOAD_CONFIG` en cualquier valor y utilizando el siguiente código:

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

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

Paso 3: Crear el JSON del paquete para el proyecto

Después de crear el directorio del proyecto `awsnodesample`, debe crear y añadir un archivo `package.json` para guardar los metadatos de su proyecto de Node.js. Para obtener información

detallada acerca de cómo utilizar `package.json` en un proyecto Node.js, consulte la sección sobre [creación de un archivo package.json](#).

En el directorio del proyecto, cree un archivo nuevo denominado `package.json`. Luego añada este JSON al archivo.

```
{
  "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"
}
```

Guarde el archivo. A medida que instale los módulos que necesita, la parte `dependencies` del archivo se completará. Puede encontrar un archivo JSON que muestra un ejemplo de estas dependencias [aquí, en GitHub](#).

Paso 4: Escriba el código de Node.js

Cree un archivo nuevo denominado `sample.js` para que contenga el código de ejemplo. Para empezar, añada que las llamadas a la función `require` deben incluir el SDK para JavaScript y los módulos `uuid`, de modo que estén disponibles para que los use.

Cree un nombre de bucket único que se utilice para crear un bucket de Amazon S3 añadiendo un valor de ID único a un prefijo fácil de reconocer, en este caso `'node-sdk-sample-'`. Puede generar el ID único llamando al módulo `uuid`. A continuación, cree un nombre para el parámetro `Key` utilizado para cargar un objeto en el bucket.

Cree un objeto `promise` para llamar al método `createBucket` del objeto de servicio `AWS.S3`. Si obtiene una respuesta correcta, cree los parámetros necesarios para cargar texto al bucket que acaba de crear. Usando otra promesa, llame al método `putObject` para cargar el objeto de texto en el 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 ejemplo se puede encontrar [aquí en GitHub](#).

Paso 5: Ejecución de la muestra

Escriba el comando siguiente para ejecutar el ejemplo.

```
node sample.js
```

Si la carga se realiza correctamente, verá un mensaje de confirmación en la línea de comandos. También puede encontrar el bucket y el objeto de texto cargado en la [consola de Amazon S3](#).

Configuración del SDK para JavaScript

En los temas de esta sección se explica cómo instalar el SDK para JavaScript para usarlo en navegadores web y con Node.js. También se muestra cómo cargar el SDK para que pueda obtener acceso a los servicios web que son compatibles con él.

Note

Los desarrolladores de React Native deben usar AWS Amplify para crear nuevos proyectos en AWS. Consulte el archivo [aws-sdk-react-native](#) para obtener más detalles.

Temas

- [Requisitos previos](#)
- [Instalación del SDK para JavaScript](#)
- [Carga del SDK para JavaScript](#)
- [Actualización desde la versión 1 del SDK para JavaScript](#)

Requisitos previos

Antes de usar AWS SDK para JavaScript, establezca si el código necesita ejecutarse en Node.js o bien en navegadores web. Después realice la siguiente operación:

- En el caso de Node.js, instale Node.js en los servidores si todavía no lo tiene instalado.
- En el caso de los navegadores web, determine cuáles son las versiones de navegador que debe admitir.

Temas

- [Configuración de un entorno Node.js de AWS](#)
- [Navegadores web compatibles](#)

Configuración de un entorno Node.js de AWS

Para configurar un entorno Node.js de AWS en el que pueda ejecutar su aplicación, utilice uno de los siguientes métodos:

- Elija una imagen de máquina de Amazon (AMI) que tenga Node.js preinstalado y cree una instancia de Amazon EC2 mediante dicha AMI. Cuando cree la instancia de Amazon EC2, seleccione su AMI en AWS Marketplace. Busque Node.js en AWS Marketplace y elija una opción de AMI que contenga una versión de Node.js (32 bits o 64 bits) preinstalada.
- Cree una instancia de Amazon EC2 e instale en ella Node.js. Para obtener más información acerca de cómo instalar Node.js en una instancia de Amazon Linux consulte [Tutorial: Configuración de Node.js en una instancia de Amazon EC2](#).
- Cree con Lambda un entorno sin servidor para ejecutar Node.js como función de Lambda. Para obtener más información acerca de cómo utilizar Node.js dentro de una función de Lambda, consulte [Modelos de programación \(Node.js\)](#) en la Guía para desarrolladores de AWS Lambda.
- Implemente su aplicación de Node.js en AWS Elastic Beanstalk. Para obtener más información acerca de cómo usar Node.js con Elastic Beanstalk, consulte [Implementación de aplicaciones Node.js en AWS Elastic Beanstalk](#) en la Guía para desarrolladores de AWS Elastic Beanstalk.

Navegadores web compatibles

El SDK para JavaScript es compatible con todos los navegadores web modernos, incluidas las versiones mínimas siguientes:

Navegador	Versión
Google Chrome	28.0+
Mozilla Firefox	26.0+
Opera	17.0+
Microsoft Edge	25.10+
Windows Internet Explorer	N/A
Apple Safari	5+

Navegador	Versión
Navegador Android	4.3+

Note

Es posible que algunos marcos, como AWS Amplify, no ofrezcan la misma compatibilidad de navegadores que el SDK para JavaScript. Consulte la documentación de los marcos en cuestión para ver información detallada.

Instalación del SDK para JavaScript

La instalación de AWS SDK para JavaScript y cómo lo instala depende de si el código se va a ejecutar en módulos de Node.js o en scripts de navegador.

No todos los servicios están disponibles de forma inmediata en el SDK. Para informarse de qué servicios son actualmente compatibles con AWS SDK para JavaScript, consulte <https://github.com/aws/aws-sdk-js/blob/master/SERVICES.md>.

Node

El método preferido de instalación de AWS SDK para JavaScript para Node.js es utilizar [npm](#), [el administrador de paquetes de Node.js](#). Para hacerlo, escriba lo siguiente en la línea de comandos:

```
npm install aws-sdk
```

En el caso de que vea el mensaje de error siguiente:

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

Escriba estos comandos en la línea de comandos:

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

Browser

No es preciso que instale el SDK para poder utilizarlo en scripts de navegador. Puede cargar el paquete del SDK alojado directamente desde Amazon Web Services con un script dentro de sus páginas HTML. El paquete del SDK alojado es compatible con el subconjunto de servicios de AWS que aplican el uso compartido de recursos entre orígenes (CORS). Para obtener más información, consulte [Carga del SDK para JavaScript](#).

Puede crear una compilación personalizada del SDK en la que se pueden seleccionar las versiones y los servicios web concretos que desea utilizar. Después deberá descargar el paquete del SDK personalizado para desarrollarlo localmente y alojarlo para que su aplicación lo utilice. Para obtener más información acerca de cómo crear una compilación personalizada del SDK, consulte [Creación del SDK para navegadores](#).

Puede descargar versiones distribuibles minimizadas y sin minimizar de la versión de AWS SDK para JavaScript actual desde GitHub en:

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

Instalación mediante Bower

[Bower](#) es un administrador de paquetes para la web. Una vez que haya instalado Bower podrá utilizarlo para instalar el SDK. Para instalar el SDK con Bower, escriba lo siguiente en una ventana del terminal:

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

Carga del SDK para JavaScript

El modo de cargar el SDK para JavaScript dependerá de si lo hace para ejecutarlo en un navegador web o en Node.js.

No todos los servicios están disponibles de forma inmediata en el SDK. Para informarse de qué servicios son actualmente compatibles con AWS SDK para JavaScript, consulte <https://github.com/aws/aws-sdk-js/blob/master/SERVICES.md>.

Node.js

Después de instalar el SDK, puede cargar el paquete de AWS en su aplicación de nodo usando `require`.

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

React Native

Para utilizar el SDK en un proyecto React Native, primero tiene que instalar el SDK mediante `npm`:

```
npm install aws-sdk
```

En la aplicación, haga referencia a la versión compatible con React Native del SDK con el siguiente código:

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

Browser

Para comenzar a utilizar el SDK lo antes posible, cargue directamente desde Amazon Web Services el paquete del SDK alojado. Para ello, añada un elemento `<script>` a las páginas de código HTML en la forma siguiente:

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

Para encontrar el `SDK_VERSION_NUMBER` actual, consulte la referencia de API para el SDK para JavaScript en la [Guía de referencia de la API de AWS SDK para JavaScript](#).

Una vez que el SDK se haya cargado en su página, estará disponible desde la variable global `AWS` (o `window.AWS`).

Si usa [browserify](#) para agrupar el código y las dependencias del módulo, deberá cargar el SDK con `require`, igual que en Node.js.

Actualización desde la versión 1 del SDK para JavaScript

Las siguientes notas le serán útiles para actualizar el SDK para JavaScript de la versión 1 a la versión 2.

Conversión automática de tipos de marca temporal y de Base64 en la entrada y la salida

Ahora el SDK codifica y descodifica automáticamente los valores con codificación base64, así como los valores de marca temporal, en lugar del usuario. Este cambio afecta a todas las operaciones en las que se envían valores con codificación base64 o de marca temporal a través de solicitudes o a las operaciones en que dichos valores se devuelven en una respuesta en la que se pueden usar valores con codificación base64.

Ahora ya no se necesita el código de usuario que anteriormente convertía valores con codificación base64. Ahora los valores con codificación base64 se devuelven como objetos de búfer desde respuestas de servidor y también se pueden transferir como entradas de búfer. Por ejemplo, los siguientes parámetros `SQS.sendMessage` de la versión 1:

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

Se pueden reescribir como se indica a continuación.

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

Aquí vemos cómo se lee el mensaje.

```
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"
});
```

Cambio de `response.data.RequestId` a `response.requestId`

Ahora el SDK almacena los ID de solicitudes de todos los servicios en un lugar coherente, en el objeto `response`, en lugar de almacenarlos dentro de la propiedad `response.data`. Esto mejora la coherencia entre los servicios que exponen los ID de las solicitudes de diferentes maneras. También es una modificación importante el hecho de cambiar el nombre de la propiedad `response.data.RequestId` por `response.requestId` (`this.requestId` dentro de una función de devolución de llamada).

En el código, cambie lo siguiente:

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

A lo siguiente:

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

Elementos de encapsulador expuestos

Si utiliza `AWS.ElasticCache`, `AWS.RDS` o `AWS.Redshift`, debe obtener acceso a la respuesta a través de la propiedad de salida de nivel superior en la respuesta a algunas operaciones.

Por ejemplo, el método `RDS.describeEngineDefaultParameters` que se utiliza para devolver lo siguiente:

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

Ahora devuelve lo siguiente:

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

En la siguiente tabla se muestra la lista de operaciones afectadas por cada servicio:

Clase de cliente	Operaciones
AWS.ElastiCache	authorizeCacheSecurityGroup Ingress createCacheCluster createCacheParameterGroup createCacheSecurityGroup createCacheSubnetGroup createReplicationGroup deleteCacheCluster deleteReplicationGroup describeEngineDefaultParameters modifyCacheCluster modifyCacheSubnetGroup modifyReplicationGroup purchaseReservedCacheNodesO ffering rebootCacheCluster revokeCacheSecurityGroupIngress
AWS.RDS	addSourceIdentifierToSubscr iption

Clase de cliente	Operaciones
	<code>authorizeDBSecurityGroupIngress</code> <code>copyDBSnapshot</code> <code>createDBInstance</code> <code>createDBInstanceReadReplica</code> <code>createDBParameterGroup</code> <code>createDBSecurityGroup</code> <code>createDBSnapshot</code> <code>createDBSubnetGroup</code> <code>createEventSubscription</code> <code>createOptionGroup</code> <code>deleteDBInstance</code> <code>deleteDBSnapshot</code> <code>deleteEventSubscription</code> <code>describeEngineDefaultParameters</code> <code>modifyDBInstance</code> <code>modifyDBSubnetGroup</code> <code>modifyEventSubscription</code> <code>modifyOptionGroup</code> <code>promoteReadReplica</code> <code>purchaseReservedDBInstances</code> <code>Offering</code> <code>rebootDBInstance</code>

Clase de cliente	Operaciones
	<code>removeSourceIdentifierFromSubscription</code> <code>restoreDBInstanceFromDBSnapshot</code> <code>restoreDBInstanceToPointInTime</code> <code>revokeDBSecurityGroupIngress</code>

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

Clase de cliente	Operaciones
	<code>purchaseReservedNodeOffering</code>
	<code>rebootCluster</code>
	<code>restoreFromClusterSnapshot</code>
	<code>revokeClusterSecurityGroupIngress</code>
	<code>revokeSnapshotAccess</code>
	<code>rotateEncryptionKey</code>

Propiedades de cliente eliminadas

Las propiedades `.Client` y `.client` se han eliminado de los objetos de servicio. Si utiliza la propiedad `.Client` en una clase de servicio o una propiedad `.client` en una instancia de objeto de servicio, elimine estas propiedades de su código.

El código siguiente que se utiliza con la versión 1 del SDK para JavaScript:

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

Debe cambiarse por el siguiente código:

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

Configurar el SDK para JavaScript

Para poder usar el SDK para JavaScript para invocar los servicios web con la API, primero tiene que configurar el SDK. Como mínimo, debe configurar los valores siguientes:

- La región en la que solicitará servicios.
- Las credenciales que autoricen su acceso a los recursos del SDK.

Además de configurar estos valores, es posible que también tenga que configurar permisos para sus recursos de AWS. Por ejemplo, puede limitar el acceso a un bucket de Amazon S3 o restringir una tabla de Amazon DynamoDB para acceso de solo lectura.

La [Guía de referencia de las herramientas y los SDK de AWS](#) también contiene configuraciones, características y otros conceptos fundamentales comunes a muchos de los SDK de AWS.

En los temas de esta sección se describen diversas formas de configurar el SDK para Node.js y JavaScript que se ejecuta en un navegador web.

Temas

- [Uso del objeto de configuración global](#)
- [Configuración de la región de AWS](#)
- [Especificación de puntos de enlace personalizados](#)
- [Autenticación de SDK con AWS](#)
- [Configuración de las credenciales](#)
- [Bloqueo de versiones de la API](#)
- [Consideraciones de Node.js](#)
- [Consideraciones sobre los scripts de navegador](#)
- [Agrupación de aplicaciones con Webpack](#)

Uso del objeto de configuración global

Hay dos formas de configurar el SDK:

- Establecer la configuración global con `AWS.Config`.

- Transferir información de configuración adicional a un objeto de servicio.

Normalmente es más fácil comenzar con la configuración global de valores mediante `AWS.Config`, pero la configuración de nivel de servicio puede aportar más control sobre los servicios individuales. La configuración global que `AWS.Config` especifica proporciona la configuración predeterminada para objetos de servicio que crea posteriormente, lo que simplifica su configuración. Sin embargo, tiene la posibilidad de actualizar la configuración de objetos de servicio individuales cuando sus necesidades varían de la configuración global.

Ajuste de la configuración global

Después de cargar el paquete `aws-sdk` en el código, puede utilizar la variable global `AWS` para acceder a las clases del SDK e interactuar con servicios individuales. El SDK incluye un objeto de configuración global, `AWS.Config`, que se utiliza para especificar los ajustes de configuración del SDK que la aplicación necesita.

Configure el SDK definiendo las propiedades `AWS.Config` de acuerdo con las necesidades de la aplicación. En la tabla siguiente se resumen las propiedades `AWS.Config` que se suelen usar para establecer la configuración del SDK.

Opciones de configuración	Descripción
<code>credentials</code>	Obligatorio. Especifica las credenciales que se usan para determinar el acceso a los servicios y los recursos.
<code>region</code>	Obligatorio. Especifica la región en la que se realizan las solicitudes de servicios.
<code>maxRetries</code>	Opcional. Especifica el número máximo de veces que una solicitud determinada se reintenta.
<code>logger</code>	Opcional. Especifica un objeto registrador en el que se escribe información de depuración.
<code>update</code>	Opcional. Actualiza la configuración actual con nuevos valores.

Para obtener más información acerca del objeto de configuración, consulte [Class: AWS.Config](#) en la referencia de la API.

Ejemplos de configuración global

Tiene que establecer la región y las credenciales en `AWS.Config`. Puede configurar estas propiedades como parte del constructor `AWS.Config`, tal y como se muestra en el siguiente ejemplo de script de navegador:

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

También puede definir estas propiedades después de crear `AWS.Config` utilizando el método `update`, tal y como se muestra en el siguiente ejemplo que actualiza la región:

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

Puede obtener sus credenciales predeterminadas llamando al método `getCredentials` estático 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);
  }
});
```

Del mismo modo, si ha definido la región correctamente en el archivo `config`, obtendrá ese valor definiendo la variable de entorno `AWS_SDK_LOAD_CONFIG` en cualquier valor y llamando a la propiedad `region` estática de `AWS.config`:

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

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

Configuración según el servicio

El acceso a cada servicio que se utiliza en el SDK de JavaScript se realiza a través de un objeto de servicio que forma parte de la API de dicho servicio. Por ejemplo, para acceder al servicio de Amazon S3, tiene que crear el objeto de servicio de Amazon S3. Puede especificar los valores de configuración que son específicos de un servicio como parte del constructor para dicho objeto de servicio. Cuando establece valores de configuración en un objeto de servicio, el constructor toma todos los valores de configuración que `AWS.Config` utiliza, incluidas las credenciales.

Por ejemplo, si necesita obtener acceso a objetos de Amazon EC2 en varias regiones, cree un objeto de servicio de Amazon EC2 para cada región y, a continuación, defina la configuración de región de cada objeto de servicio en consecuencia.

```
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'});
```

También puede establecer valores de configuración específicos de un servicio al configurar el SDK con `AWS.Config`. El objeto de configuración global admite muchas opciones de configuración específicas de servicios. Para obtener más información acerca de la configuración específica de servicios, consulte [Class: AWS.Config](#) en la referencia de la API de AWS SDK para JavaScript.

Datos de configuración inmutables

Los cambios de la configuración global se aplican a las solicitudes de todos los objetos de servicio que se acaban de crear. Los objetos de servicio recién creados se configuran primero con los datos de la configuración global actual y después con las opciones de configuración local. Las actualizaciones que se introducen en el objeto `AWS.config` global no se aplican a los objetos de servicio que se ha creado anteriormente.

Los objetos de servicio ya existentes deben actualizarse manualmente con datos de la nueva configuración o bien debe crear y utilizar un objeto de servicio nuevo que tenga los nuevos datos de configuración. En el siguiente ejemplo, se crea un objeto de servicio de Amazon S3 con datos de configuración nuevos:

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

Configuración de la región de AWS

Una región es un conjunto designado de recursos de AWS que están en la misma área geográfica. Un ejemplo de región es `us-east-1`, que es la región Este de EE. UU. (Norte de Virginia). La región se especifica al configurar el SDK para JavaScript para que el SDK obtenga acceso a los recursos de dicha región. Algunos servicios solo están disponibles en regiones específicas.

El SDK para JavaScript no selecciona una región de forma predeterminada. Sin embargo, puede configurar la región mediante una variable de entorno, un archivo `config` compartido o el objeto de configuración global.

En un constructor de clase de cliente

Cuando cree una instancia de un objeto de servicio, puede especificar la región de AWS de dicho recurso como parte del constructor de clases de cliente, tal y como se muestra aquí.

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

Uso del objeto de configuración global

Para establecer la región en su código JavaScript, actualice el objeto de configuración global `AWS.Config` tal y como se muestra aquí.

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

Para obtener más información acerca de las regiones y los servicios disponibles actualmente en cada región, consulte [Regiones y puntos de conexión de AWS](#) en la Referencia general de AWS.

Uso de una variable de entorno

Puede establecer la región mediante la variable de entorno `AWS_REGION`. Si define esta variable, el SDK para JavaScript la lee y la utiliza.

Uso de un archivo de configuración compartido

Al igual que el archivo de credenciales compartidas le permite almacenar credenciales para que el SDK las use, puede mantener su región y otras opciones de configuración en un archivo compartido denominado `config` que el SDK utiliza. Si la variable de entorno `AWS_SDK_LOAD_CONFIG` se

establece en cualquier valor, el SDK para JavaScript busca automáticamente un archivo `config` cuando se carga. La ubicación donde guarde el archivo `config` depende de su sistema operativo:

- Usuarios de Linux, macOS o Unix: `~/.aws/config`
- Usuarios de Windows: `C:\Users\USER_NAME\.aws\config`

Si todavía no tiene un archivo `config` compartido, puede crear uno en el directorio designado. En el siguiente ejemplo, el archivo `config` establece la región y el formato de salida.

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

Para obtener más información acerca del uso de archivos de configuración y de credenciales compartidos, consulte [Carga de credenciales en Node.js desde el archivo de credenciales compartidas](#) o [Archivos de configuración y credenciales](#) en la Guía del usuario de AWS Command Line Interface.

Orden de prioridad para establecer la región

El orden de prioridad para la configuración de una región es el siguiente:

- Si se transfiere una región a un constructor de clase de cliente, se usa dicha región. De lo contrario, entonces...
- Si se establece una región en el objeto de configuración global, se usa dicha región. De lo contrario, entonces...
- Si la variable de entorno `AWS_REGION` es un valor [truthy](#), se usa esa región. De lo contrario, entonces...
- Si la variable de entorno `AMAZON_REGION` es un valor `truthy`, se usa esa región. De lo contrario, entonces...
- Si la variable de entorno `AWS_SDK_LOAD_CONFIG` se establece en cualquier valor y el archivo de credenciales compartidas (`~/.aws/credentials` o la ruta indicada por `AWS_SHARED_CREDENTIALS_FILE`) contiene una región para el perfil configurado, se usa esa región. De lo contrario, entonces...
- Si la variable de entorno `AWS_SDK_LOAD_CONFIG` se establece en cualquier valor y el archivo de configuración (`~/.aws/config` o la ruta indicada por `AWS_CONFIG_FILE`) contiene una región para el perfil configurado, se usa esa región.

Especificación de puntos de enlace personalizados

Las llamadas a los métodos de la API en el SDK para JavaScript se realizan a los URI de punto de conexión de servicio. De forma predeterminada, estos puntos de enlace se crean a partir de la región que ha configurado para su código. Sin embargo, hay situaciones en las que es preciso especificar un punto de enlace personalizado para las llamadas a la API.

Formato de cadena de punto de enlace

Los valores de los puntos de enlace tienen que ser una cadena con el formato:

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

Puntos de enlace para la región ap-noreste-3

La región ap-northeast-3 de Japón no se devuelve mediante API de enumeración de regiones como [EC2.describeRegions](#). Para definir puntos de enlace para esta región, siga el formato que se ha descrito anteriormente. Por lo tanto, el punto de conexión de Amazon EC2 para esta región sería

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

Puntos de conexión para MediaConvert

Debe crear un punto de conexión personalizado para usarlo con MediaConvert. A cada cuenta de cliente se le asigna su propio punto de enlace, que debe utilizar. A continuación, se muestra un ejemplo de cómo utilizar un punto de conexión personalizado con 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 obtener el punto de enlace de la API de su cuenta, consulte [MediaConvert.describeEndpoints](#) en la referencia de la API.

Asegúrese de especificar en el código la misma región que en el URI del punto de enlace personalizado. Una discrepancia entre la configuración de la región y el URI del punto de enlace personalizado puede hacer que las llamadas a la API generen un error.

Para obtener más información sobre MediaConvert, consulte la clase [AWS.MediaConvert](#) en la Referencia de la API o en la [Guía del usuario de AWS Elemental MediaConvert](#).

Autenticación de SDK con AWS

Debe establecer cómo se autentica el código con AWS cuando se desarrolla con servicios de AWS. Puede configurar el acceso mediante programación a los recursos de AWS de diferentes maneras, según el entorno y el acceso de AWS disponibles.

Para elegir el método de autenticación y configurarlo para el SDK, consulte [Autenticación y acceso](#) en la Guía de referencia de herramientas y SDK de AWS.

Recomendamos que los nuevos usuarios que se desarrollen localmente y no dispongan de un método de autenticación por parte de su empleador configuren AWS IAM Identity Center. Este método incluye la instalación de AWS CLI para facilitar la configuración y para iniciar sesión con regularidad en el portal de acceso a AWS. Si elige este método, su entorno debería contener los siguientes elementos después de completar el procedimiento de [Autenticación del IAM Identity Center](#) descrito en la Guía de referencia de las herramientas y los AWS SDK:

- AWS CLI, que se utiliza para iniciar una sesión en el portal de acceso a AWS antes de ejecutar la aplicación.
- Un [archivo config compartido de AWS](#) que tiene un perfil [default] con un conjunto de valores de configuración a los que se puede hacer referencia desde el SDK. Para encontrar la ubicación de este archivo, consulte [Ubicación de los archivos compartidos](#) en la Guía de referencia de herramientas y AWS SDK.
- El archivo compartido de config establece la configuración de [region](#). Esto establece la región de AWS predeterminada que el SDK usa para las solicitudes de AWS. Esta región se usa para las solicitudes de servicio del SDK que no tienen especificadas una región.
- El SDK utiliza la [configuración de proveedor de token de SSO](#) del perfil para adquirir las credenciales antes de enviar las solicitudes a AWS. El valor `sso_role_name`, que es un rol de IAM conectado a un conjunto de permisos del Centro de identidades de IAM, permite el acceso a los servicios de AWS utilizados en la aplicación.

El siguiente archivo config de ejemplo muestra la configuración de un perfil predeterminado con el proveedor de token de SSO. La configuración `sso_session` del perfil hace referencia a la [sección llamada `sso-session`](#). La sección `sso-session` contiene la configuración para iniciar una sesión en el portal de acceso a 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
```

El SDK para JavaScript no necesita que se agreguen paquetes adicionales (como SSO y SSO0IDC) a la aplicación para utilizar la autenticación de IAM Identity Center.

Iniciar una sesión en el portal de acceso a AWS

Antes de ejecutar una aplicación que accede a Servicios de AWS, necesita una sesión activa en el portal de acceso a AWS para que SDK utilice la autenticación del IAM Identity Center para resolver las credenciales. En función de la duración de las sesiones configuradas, el acceso terminará por caducar y SDK detectará un error de autenticación. Para iniciar sesión en el portal de acceso a AWS, ejecute el siguiente comando en AWS CLI.

```
aws sso login
```

Si sigue la guía y utiliza una configuración de perfil predeterminada, no necesita llamar al comando con una opción `--profile`. Si la configuración del proveedor de token de SSO utiliza un perfil con nombre, el comando es `aws sso login --profile named-profile`.

Para comprobar si ya tiene una sesión activa, ejecute el siguiente comando de AWS CLI.

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

Si su sesión está activa, la respuesta a este comando debe indicar la cuenta y el conjunto de permisos del Centro de identidades de IAM configurados en el archivo `config` compartido.

Note

Si ya tiene una sesión activa en el portal de acceso a AWS y ejecuta `aws sso login`, no tendrá que proporcionar credenciales.

Es posible que el proceso de inicio de sesión le permita el acceso de AWS CLI a los datos. Como AWS CLI se ha creado con el SDK para Python, los mensajes de permiso pueden contener variaciones del nombre `botocore`.

Información adicional de autenticación

Los usuarios humanos, que también reciben el nombre de identidades humanas, son las personas, los administradores, los desarrolladores, los operadores y los consumidores de las aplicaciones. Deben tener una identidad para acceder a los entornos y aplicaciones de AWS. Los usuarios humanos que son miembros de su organización (es decir, usted, el desarrollador) se conocen como identidades de personal.

Utilice credenciales temporales al acceder a AWS. Puede utilizar un proveedor de identidades para los usuarios humanos para proporcionar acceso federado a las cuentas de AWS asumiendo roles, que proporcionan credenciales temporales. Si desea administrar el acceso de manera centralizada, se recomienda utilizar (IAM Identity Center) para administrar el acceso a las cuentas y los permisos de esas cuentas. Para obtener más alternativas, consulte lo siguiente:

- Para obtener más información sobre las prácticas recomendadas, consulte [Prácticas recomendadas de seguridad en IAM](#) en la Guía del usuario de IAM.
- Para crear credenciales de AWS de corta duración, consulte [Credenciales de seguridad temporales](#) en la Guía del usuario de IAM.
- Para obtener más información sobre otros proveedores de credenciales de SDK para JavaScript, consulte los [proveedores de credenciales estandarizados](#) en la Guía de referencia de las herramientas y los SDK de AWS.

Configuración de las credenciales

AWS utiliza credenciales para identificar quién llama a servicios y si el acceso a los recursos solicitados está permitido.

El código JavaScript, sea cual sea su punto de ejecución, un navegador web o un servidor Node.js, debe obtener credenciales válidas para poder obtener acceso a los servicios a través de la API. Las credenciales se pueden configurar en el objeto de configuración, mediante `AWS.Config` o por servicio, por medio de la transferencia directa de las credenciales a un objeto de servicio.

Hay varias formas de configurar credenciales que difieren entre Node.js y JavaScript en navegadores web. En los temas de esta sección se describe cómo configurar credenciales en Node.js o en navegadores web. En cada caso, las opciones se presentan en el orden recomendado.

Prácticas recomendadas para las credenciales

Configurar correctamente las credenciales garantiza que su aplicación o script de navegador pueda obtener acceso a los servicios y los recursos necesarios y, al mismo tiempo, minimiza su exposición a problemas de seguridad que puedan repercutir en aplicaciones críticas o pongan en peligro información confidencial.

Cuando se configuran credenciales es importante aplicar siempre el principio de conceder el privilegio mínimo necesario para llevar a cabo la tarea. Es más seguro proporcionar permisos mínimos sobre sus recursos y añadir más permisos según sea necesario, en lugar de proporcionar permisos que superan el privilegio mínimo y luego tener que solucionar problemas de seguridad que puedan aparecer más tarde. Por ejemplo, a menos que necesite leer y escribir recursos individuales, como objetos en un bucket de Amazon S3 o una tabla de DynamoDB, establezca dichos permisos en solo lectura.

Para obtener más información sobre la concesión de privilegios mínimos, consulte la sección [Conceder privilegios mínimos](#) del tema de prácticas recomendadas de la Guía del usuario de IAM.

Warning

Aunque es posible hacerlo, le recomendamos que no codifique de forma rígida las credenciales dentro de una aplicación o un script de navegador. La codificación rígida de las credenciales conlleva un riesgo de exponer información confidencial.

Para obtener más información acerca de cómo administrar las claves de acceso, consulte [Prácticas recomendadas para administrar las claves de acceso de AWS](#) en la Referencia general de AWS.

Temas

- [Configuración de credenciales en Node.js](#)
- [Configuración de credenciales en un navegador web](#)

Configuración de credenciales en Node.js

En Node.js hay varias formas de proporcionar sus credenciales al SDK. Algunas de ellas son más seguras y otras ofrecen más comodidad mientras se desarrolla una aplicación. Al obtener las credenciales en Node.js, sea especialmente cuidadoso si se basa en más de un origen como, por ejemplo, una variable de entorno y un archivo JSON que carga. Puede cambiar los permisos con los que se ejecuta su código sin darse cuenta de que se ha producido el cambio.

A continuación se muestran las formas de suministro de credenciales según su orden de recomendación:

1. Cargado desde roles de AWS Identity and Access Management (IAM) para Amazon EC2
2. Cargadas desde un archivo de credenciales compartidas (~/.aws/credentials).
3. Cargadas desde variables de entorno.
4. Cargadas desde un archivo JSON en el disco.
5. Otras clases de proveedores de credenciales que proporciona el SDK para JavaScript

Si hay más de un origen de credenciales disponible para el SDK, la prioridad de selección predeterminada es la siguiente:

1. Credenciales que se establecen explícitamente a través del constructor de cliente de servicio.
2. Variables de entorno
3. El archivo de credenciales compartidas.
4. Credenciales cargadas desde el proveedor de credenciales de ECS (si corresponde)
5. Credenciales que se obtienen utilizando un proceso de credenciales especificado en el archivo de configuración de AWS compartido o en el archivo de credenciales compartido. Para obtener más información, consulte [the section called “Credenciales con un proceso de credenciales configurado”](#).

6. Credenciales cargadas desde AWS IAM mediante el proveedor de credenciales de la instancia de Amazon EC2 (si se configura en los metadatos de instancia)

Para obtener más información, consulte [Class: AWS.Credentials](#) y [Class: AWS.CredentialProviderChain](#) en la referencia de la API.

Warning

Aunque es posible hacerlo, no le recomendamos codificar de forma rígida sus credenciales de AWS en la aplicación. Si las codifica de forma rígida, corre el riesgo de que su ID de clave de acceso y la clave de acceso secreta queden expuestos.

En los temas de esta sección se describe cómo cargar credenciales en Node.js.

Temas

- [Carga de credenciales en Node.js desde roles de IAM para Amazon EC2](#)
- [Carga de credenciales para una función Lambda de Node.js](#)
- [Carga de credenciales en Node.js desde el archivo de credenciales compartidas](#)
- [Carga de credenciales en Node.js desde variables de entorno](#)
- [Carga de credenciales en Node.js desde un archivo JSON](#)
- [Loading Credentials en Node.js mediante un Proceso de credenciales configurado](#)

Carga de credenciales en Node.js desde roles de IAM para Amazon EC2

Si ejecuta su aplicación de Node.js en una instancia de Amazon EC2, puede utilizar roles de IAM para Amazon EC2, para suministrar automáticamente credenciales a la instancia. Si configura la instancia para que utilice roles de IAM, el SDK selecciona automáticamente las credenciales de IAM para su aplicación, lo que elimina la necesidad de proporcionar las credenciales manualmente.

Para obtener más información sobre cómo agregar roles de IAM a una instancia de Amazon EC2, consulte [Uso de roles de IAM para instancias de Amazon EC2](#) en la Guía de referencia de las herramientas y los SDK de AWS.

Carga de credenciales para una función Lambda de Node.js

Al crear una función de Lambda, tiene que crear un rol de IAM especial que tenga permiso para ejecutar la función. Este rol se denomina el rol de ejecución. Cuando configura una función de Lambda, tiene que especificar el rol de IAM que ha creado como el rol de ejecución correspondiente.

El rol de ejecución proporciona a la función de Lambda las credenciales que necesita para ejecutarse y para invocar otros servicios web. En consecuencia, no es necesario proporcionar credenciales al código Node.js que escribe dentro de una función de Lambda.

Para obtener más información sobre cómo crear un rol de ejecución de Lambda, consulte [Administración de permisos mediante un rol de IAM \(rol de ejecución\)](#) en la Guía para desarrolladores de AWS Lambda.

Carga de credenciales en Node.js desde el archivo de credenciales compartidas

Puede conservar los datos de sus credenciales de AWS en un archivo compartido que utilizan los SDK y la interfaz de línea de comandos. Cuando se carga el SDK para JavaScript, busca automáticamente el archivo de credenciales compartidas, que se llama “credentials”. La ubicación del archivo de credenciales compartido depende del sistema operativo:

- El archivo de credenciales compartidas en Linux, Unix y macOS: `~/.aws/credentials`.
- El archivo de credenciales compartidas en Windows: `C:\Users\USER_NAME\.aws\credentials`.

Si aún no tiene un archivo de credenciales compartidas, consulte [Autenticación de SDK con AWS](#). Después de seguir estas instrucciones, debería ver un texto similar al siguiente en el archivo de credenciales, donde `<YOUR_ACCESS_KEY_ID>` es el ID de clave de acceso y `<YOUR_SECRET_ACCESS_KEY>` es la clave de acceso secreta:

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

Para ver un ejemplo de uso de este archivo, consulte [Introducción a Node.js](#).

El encabezado de sección `[default]` especifica un perfil predeterminado y valores asociados para las credenciales. Puede crear perfiles adicionales en el mismo archivo de configuración compartido,

cada uno con su propia información de credenciales. En el ejemplo siguiente se muestra un archivo de configuración con el perfil predeterminado y dos perfiles adicionales:

```
[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>
```

De forma predeterminada, el SDK comprueba la variable de entorno `AWS_PROFILE` para determinar qué perfil se va a utilizar. Si la variable `AWS_PROFILE` no se establece en su entorno, el SDK utiliza las credenciales del perfil `[default]`. Para utilizar uno de los perfiles alternativos, configure o cambie el valor de la variable de entorno `AWS_PROFILE`. Por ejemplo, dado el archivo de configuración que se muestra más arriba, para utilizar las credenciales de la cuenta de trabajo, establezca la variable de entorno `AWS_PROFILE` en `work-account` (según corresponda para su sistema operativo).

Note

Cuando configure variables de entorno, asegúrese de tomar las medidas adecuadas después (según las necesidades de su sistema operativo) para que las variables estén disponibles en el entorno de shell o de comandos.

Después de configurar la variable de entorno (si es necesario), puede ejecutar un archivo de JavaScript que utiliza el SDK, como por ejemplo, un archivo llamado `script.js`.

```
$ node script.js
```

También puede seleccionar de forma explícita el perfil que el SDK utiliza, ya sea estableciendo `process.env.AWS_PROFILE` antes de cargar el SDK o seleccionando el proveedor de credenciales tal y como se muestra en el ejemplo siguiente:

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

Carga de credenciales en Node.js desde variables de entorno

El SDK detecta automáticamente las credenciales de AWS establecidas como variables en el entorno y las utiliza para sus solicitudes, lo que elimina la necesidad de administrar credenciales en su aplicación. Las variables de entorno que establece para suministrar las credenciales son:

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

Para obtener más información sobre cómo establecer variables de entorno, consulte [Compatibilidad con variables de entorno](#) en la Guía de referencia de las herramientas y los SDK de AWS.

Carga de credenciales en Node.js desde un archivo JSON

Puede cargar la configuración y las credenciales desde un documento JSON en el disco mediante `AWS.config.loadFromPath`. La ruta se especifica en relación con el directorio de trabajo actual de su proceso. Por ejemplo, para cargar credenciales desde un archivo `config.json` con el siguiente contenido:

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

A continuación, utilice el siguiente código:

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

Note

Cargar datos de configuración desde un documento JSON restablece todos los datos de configuración ya existentes. Añada datos de configuración adicionales después de usar esta técnica. La carga de credenciales desde un documento JSON no se admite en scripts de navegador.

Loading Credentials en Node.js mediante un Proceso de credenciales configurado

Puede obtener credenciales utilizando un método que no esté integrado en el SDK. Para ello, especifique un proceso de credenciales en el archivo de configuración de AWS compartido o en el archivo de credenciales compartido. Si la variable de entorno `AWS_SDK_LOAD_CONFIG` se establece en cualquier valor, el SDK preferirá el proceso especificado en el archivo de configuración al proceso especificado en el archivo de credenciales (si existe).

Para obtener más información sobre cómo especificar un proceso de credenciales en el archivo de configuración de AWS compartido o en el archivo de credenciales compartido, consulte la Referencia de comandos de AWS CLI, específicamente la información sobre la [obtención de credenciales de procesos externos](#).

Para obtener información sobre el uso de la variable de entorno `AWS_SDK_LOAD_CONFIG`, consulte [the section called “Uso de un archivo de configuración compartido”](#) en este documento.

Configuración de credenciales en un navegador web

Hay varias formas de proporcionar credenciales al SDK desde scripts de navegador. Algunas de ellas son más seguras y otras ofrecen más comodidad mientras se desarrolla un script. A continuación se muestran las formas de suministro de credenciales según su orden de recomendación:

1. Uso de Amazon Cognito Identity para autenticar a los usuarios y proporcionar credenciales
2. Uso de identidades federadas web
3. codificadas de forma rígida en el script

Warning

No se recomienda codificar de forma rígida las credenciales de AWS en sus scripts. Si las codifica de forma rígida, corre el riesgo de que su ID de clave de acceso y la clave de acceso secreta queden expuestos.

Temas

- [Uso de Amazon Cognito Identity para autenticar a los usuarios](#)
- [Uso de identidades federadas web para autenticar usuarios](#)

- [Ejemplos de identidades federadas web](#)

Uso de Amazon Cognito Identity para autenticar a los usuarios

La forma recomendada de obtener credenciales de AWS para los scripts del navegador es utilizar el objeto de credenciales de Amazon Cognito Identity, `AWS.CognitoIdentityCredentials`. Amazon Cognito permite la autenticación de los usuarios a través de proveedores de identidad de terceros.

Para usar Amazon Cognito Identity, primero debe crear un grupo de identidades en la consola Amazon Cognito. Un grupo de identidades representa el grupo de identidades que su aplicación proporciona a los usuarios. Las identidades que se dan a los usuarios identifican de forma inequívoca cada cuenta de usuario. Las identidades de Amazon Cognito no son credenciales. Se intercambian por credenciales utilizando la compatibilidad con las identidades web federadas en AWS Security Token Service (AWS STS).

Amazon Cognito es útil para administrar la abstracción de identidades a través de varios proveedores de identidades con el objeto `AWS.CognitoIdentityCredentials`. La identidad que se carga se intercambia por credenciales en AWS STS.

Configuración del objeto de credenciales de Amazon Cognito Identity

Si todavía no ha creado un grupo de identidades, cree uno para usarlo con los scripts de navegador en la [consola de Amazon Cognito](#) antes de configurar `AWS.CognitoIdentityCredentials`. Cree y asocie roles de IAM autenticados y sin autenticar para su grupo de identidades.

La identidad de los usuarios sin autenticar no se verifica, lo que hace que este rol sea adecuado para los usuarios invitados de la aplicación o para cuando no importa si se verifica la identidad de los usuarios. Los usuarios autenticados inician sesión en la aplicación a través de un proveedor de identidades externo que verifica sus identidades. Asegúrese de asignar los permisos de los recursos de forma adecuada, para no conceder acceso a ellos a los usuarios no autenticados.

Después de configurar un grupo de identidades con proveedores de identidades asociados, puede utilizar `AWS.CognitoIdentityCredentials` para autenticar a los usuarios. Para configurar las credenciales de la aplicación para utilizar `AWS.CognitoIdentityCredentials`, establezca la propiedad `credentials` de `AWS.Config` o una configuración específica para cada servicio. El siguiente ejemplo utiliza `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'
}
});
```

La propiedad opcional `Logins` es un mapeo entre los nombres de los proveedores de identidad y los tokens de identidad de los proveedores. La forma de obtener el token del proveedor de identidad depende del proveedor que se utilice. Por ejemplo, si Facebook es uno de sus proveedores de identidad, puede utilizar la función `FB.login` del [SDK de Facebook](#) para obtener un token de proveedor de identidad:

```
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.');
```

```
  }
});
```

Cambio de usuarios sin autenticar a usuarios autenticados

Amazon Cognito es compatible con los usuarios autenticados y no autenticados. Los usuarios sin autenticar reciben acceso a sus recursos incluso si no han iniciado sesión con alguno de sus proveedores de identidades. Este grado de acceso es útil para mostrar contenido a usuarios antes de que inicien sesión. Cada usuario sin autenticar tiene una identidad única en Amazon Cognito, aunque no haya iniciado sesión ni se haya autenticado individualmente.

Usuario sin autenticar inicialmente

Los usuarios suelen comenzar con el rol sin autenticar, para el que se establece la propiedad de credenciales de su objeto de configuración sin una propiedad `Logins`. En este caso, la configuración predeterminada podría tener el siguiente aspecto:

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

Cambie a usuario autenticado

Cuando un usuario sin autenticar inicia sesión en un proveedor de identidades y tiene un token, puede cambiar el usuario de no estar autenticado a estar autenticado llamando a una función personalizada que actualiza el objeto de credenciales y añade el 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;
}
```

También puede crear un objeto `CognitoIdentityCredentials`. Si lo hace, debe restablecer las propiedades de las credenciales de los objetos de servicio existentes que ha creado. Los objetos de servicio leen en la configuración global solo al realizarse la inicialización del objeto.

Para obtener más información acerca del objeto `CognitoIdentityCredentials`, consulte [AWS.CognitoIdentityCredentials](#) en la referencia de la API de AWS SDK para JavaScript.

Uso de identidades federadas web para autenticar usuarios

Puede configurar directamente distintos proveedores de identidades para obtener acceso a recursos de AWS a través de la federación de identidades web. AWS actualmente admite la autenticación de usuarios mediante las identidades web federadas a través de varios proveedores de identidades:

- [Login with Amazon](#)

- [Inicio de sesión con Facebook](#)
- [Inicio de sesión con Google](#)

Primero debe registrar la aplicación con los proveedores compatibles con su aplicación. Luego cree un rol de IAM y configure permisos para él. El rol de IAM que cree se utilizará para conceder los permisos que haya configurado para él a través del proveedor de identidades respectivo. Por ejemplo, puede configurar un rol que permita a los usuarios que han iniciado sesión a través de Facebook tener acceso de lectura a un bucket de Amazon S3 específico que usted controla.

Una vez que tenga un rol de IAM con privilegios configurados y una aplicación registrada en los proveedores de identidades de su elección, puede configurar el SDK para obtener credenciales para el rol de IAM usando código auxiliar, tal y como se indica a continuación:

```
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
});
```

El valor del parámetro `ProviderId` depende del proveedor de identidades especificado. El valor del parámetro `WebIdentityToken` es el token de acceso recuperado desde un inicio de sesión correcto con el proveedor de identidades. Para obtener más información sobre cómo configurar y recuperar tokens de acceso para cada proveedor de identidades, consulte la documentación del proveedor de identidades.

Paso 1: Registro en proveedores de identidades

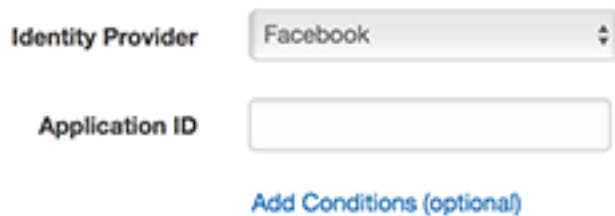
Para empezar, registre una aplicación en los proveedores de identidades que decida admitir. Se le pedirá que proporcione información que identifique su aplicación y posiblemente su autor. De este modo, se garantiza que los proveedores de identidades sepan quién está recibiendo su información de usuario. En todos los casos, el proveedor de identidades emitirá un ID de aplicación que se utilizará para configurar los roles de usuarios.

Paso 2: Creación de un rol de IAM para un proveedor de identidades

Después de obtener el ID de aplicación de un proveedor de identidades, vaya a la consola de IAM en <https://console.aws.amazon.com/iam/> para crear un rol de IAM nuevo.

Para crear un rol de IAM para proveedor de identidades

1. Vaya a la sección Roles de la consola y, a continuación, seleccione Create New Role (Crear nuevo rol).
2. Escriba un nombre para el nuevo rol que le ayudará a realizar un seguimiento de su uso, como por ejemplo **facebookIdentity** y luego elija Next Step (Paso siguiente).
3. En Select Role Type (Seleccionar tipo de rol), elija Role for Identity Provider Access (Rol para acceso del proveedor de identidades).
4. Para Grant access to web identity providers (Conceder acceso a los proveedores de identidades web), elija Select (Seleccionar).
5. En la lista Proveedor de identidades, elija el proveedor de identidades que desea usar para este rol de IAM.



The image shows a portion of the AWS IAM console interface. It features two main input fields: 'Identity Provider' and 'Application ID'. The 'Identity Provider' field is a dropdown menu with 'Facebook' selected. Below it is the 'Application ID' field, which is currently empty. A blue link labeled 'Add Conditions (optional)' is visible below the 'Application ID' field.

6. Escriba el ID de la aplicación que ha proporcionado el proveedor de identidades en Application ID (ID de la aplicación) y, a continuación, seleccione Next Step (Paso siguiente).
7. Configure permisos para los recursos que desea mostrar y permita el acceso a operaciones específicas en recursos específicos. Para obtener más información sobre los permisos de IAM, consulte [Descripción general de los permisos de AWS IAM](#) en la Guía del usuario de IAM. Revise y, si es necesario, personalice la relación de confianza del rol y, a continuación, seleccione Next Step (Paso siguiente).
8. Asocie las políticas adicionales que necesita y, a continuación, seleccione Next Step (Paso siguiente). Para obtener más información acerca de las políticas de IAM, consulte [Descripción general de las políticas de IAM](#) en la Guía del usuario de IAM.
9. Revise el nuevo rol y, a continuación, elija Create role (Crear rol).

Puede añadir más restricciones al rol como, por ejemplo, limitarlo a ID de usuario específicos. Si el rol concede permisos de escritura sobre sus recursos, asegúrese de que restringe el rol correctamente a los usuarios que tengan los privilegios adecuados; de lo contrario, cualquier usuario con una identidad de Amazon, Facebook o Google podrá modificar recursos de su aplicación.

Para obtener más información acerca de cómo usar la federación de identidades web en IAM, consulte [Acerca de identidades web federadas](#) en la Guía del usuario de IAM.

Paso 3: Obtención de un token de acceso después de iniciar sesión

Configure la acción de inicio de sesión para su aplicación mediante el SDK del proveedor de identidades. Puede descargar e instalar un SDK de JavaScript del proveedor de identidades que permita el inicio de sesión de usuario mediante OAuth u OpenID. Para obtener información acerca de cómo descargar y configurar el código del SDK en su aplicación, consulte la documentación del SDK para su proveedor de identidades:

- [Login with Amazon](#)
- [Inicio de sesión con Facebook](#)
- [Inicio de sesión con Google](#)

Paso 4: Obtención de credenciales temporales

Una vez que haya configurado su aplicación, los roles y los permisos a nivel de recursos, añada el código a la aplicación para obtener credenciales temporales. Estas credenciales se proporcionan a través de AWS Security Token Service con la identidad web federada. Los usuarios inician sesión en el proveedor de identidades, el cual devuelve un token de acceso. Configure el objeto `AWS.WebIdentityCredentials` con el ARN para el rol de IAM que ha creado para este proveedor de identidades:

```
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
});
```

Los objetos de servicio que se creen posteriormente tendrán las credenciales apropiadas. Los objetos creados antes de configurar la propiedad `AWS.config.credentials` no tienen las credenciales actuales.

También puede crear `AWS.WebIdentityCredentials` antes de recuperar el token de acceso. Esto le permite crear objetos de servicio que dependen de las credenciales antes de cargar el token de acceso. Para ello, cree el objeto de credenciales sin el parámetro `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;
```

A continuación, establezca `WebIdentityToken` en la devolución de llamada del SDK del proveedor de identidades que contiene el token de acceso:

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

Ejemplos de identidades federadas web

A continuación se ofrecen algunos ejemplos de uso de identidades federadas web para obtener credenciales en JavaScript de navegador. Estos ejemplos debe ejecutarse desde un esquema de host `http://` o `https://` para garantizar que el proveedor de identidades pueda realizar redirecciones a la aplicación.

Ejemplo de inicio de Login with Amazon

En el código siguiente se muestra cómo utilizar Login with Amazon como proveedor de identidades.

```
<a href="#" id="login">
  
</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>
```

Ejemplo de inicio de sesión con Facebook

En el código siguiente se muestra cómo utilizar el inicio de sesión con Facebook como proveedor de identidades:

```

<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>
```

Ejemplo de inicio de sesión con Google+

En el código siguiente se muestra cómo utilizar el inicio de sesión con Google+ como proveedor de identidades. El token de acceso que se utiliza para la identidad web federada de Google se almacena en `response.id_token` en lugar de almacenarse en `access_token`, como es el caso de otros proveedores de identidades.

```
<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>
```

Bloqueo de versiones de la API

AWS Los servicios de tienen números de versión de API para realizar un seguimiento de la compatibilidad de API. Las versiones de API en los servicios de AWS se identifican mediante una cadena de fecha con formato YYYY-mm-dd. Por ejemplo, la versión actual de la API de Amazon S3 es 2006-03-01.

Le recomendamos bloquear la versión de la API de un servicio si se basa en él en el código de producción. Así puede aislar sus aplicaciones de los cambios en los servicios derivados de las actualizaciones del SDK. Si no especifica una versión de la API al crear objetos de servicio, el SDK utiliza la última versión de la API de forma predeterminada. Esto podría hacer que su aplicación hiciera referencia a una API actualizada con cambios que afectan negativamente a su aplicación.

Para bloquear la versión de la API que utiliza para un servicio, transfiera el parámetro `apiVersion` cuando cree el objeto de servicio. En el ejemplo siguiente, un objeto de servicio `AWS.DynamoDB` que se acaba de crear está bloqueado en la versión de API `2011-12-05`:

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

Puede configurar globalmente un conjunto de versiones de la API de servicio mediante el parámetro `apiVersions` en `AWS.Config`. Por ejemplo, para establecer versiones específicas de las API de `DynamoDB` y `Amazon EC2`, junto con la API de `Amazon Redshift` actual, establezca `apiVersions` tal y como se indica a continuación:

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

Obtener versiones de la API

Para obtener la versión de la API de un servicio, consulte la sección Bloqueo de la versión de API en la página de referencia del servicio; por ejemplo, <https://docs.aws.amazon.com/AWSJavaScriptSDK/latest/AWS/S3.html> en el caso de `Amazon S3`.

Consideraciones de Node.js

Aunque el código `Node.js` es `JavaScript`, el uso de `AWS SDK for JavaScript` en `Node.js` puede ser diferente del uso del SDK en scripts de navegador. Algunos métodos de API funcionan en `Node.js` pero no en scripts de navegador y viceversa. Y el uso correcto de algunas API depende de su conocimiento de patrones de codificación de `Node.js` habituales como la importación y el uso de otros módulos de `Node.js` como el módulo `File System (fs)`.

Uso de módulos `Node.js` integrados

`Node.js` ofrece un conjunto de módulos integrados que puede utilizar sin necesidad de instalarlos. Para utilizar estos módulos, cree un objeto con el método `require` para especificar el nombre del módulo. Por ejemplo, para incluir el módulo `HTTP` integrado, utilice el código siguiente.

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

Invoque métodos del módulo como si fueran métodos de dicho objeto. Por ejemplo, a continuación le mostramos código que lee un archivo 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 obtener una lista completa de todos los módulos integrados que Node.js proporciona, consulte la [documentación de Node.js v6.11.1](#) sobre el sitio web de Node.js.

Uso de paquetes NPM

Además de los módulos integrados, también puede incluir e incorporar código de terceros desde npm, el administrador de paquetes de Node.js. Se trata de un repositorio de paquetes Node.js de código abierto y una interfaz de línea de comandos para instalar dichos paquetes. Para obtener más información acerca de npm y una lista de los paquetes disponibles actualmente, consulte <https://www.npmjs.com>. También puede obtener más información acerca de paquetes de Node.js adicionales que puede utilizar [aquí en GitHub](#).

AWS SDK para JavaScript es un ejemplo de paquete npm que puede usar con browserify. Para obtener más información, consulte [Compilación del SDK como dependencia con Browserify](#). Otro ejemplo webpack. Para obtener más información, consulte [Agrupación de aplicaciones con Webpack](#).

Temas

- [Configuración de maxSockets en Node.js](#)
- [Reutilización de conexiones con Keep-Alive en Node.js](#)
- [Configuración de proxies para Node.js](#)
- [Registro de paquetes de certificados en Node.js](#)

Configuración de maxSockets en Node.js

En Node.js, puede definir el número máximo de conexiones por origen. Si `maxSockets` está establecido, el cliente HTTP de bajo nivel pone en cola las solicitudes y las asigna a conectores a medida que estos están disponibles.

Esto le permite configurar un límite máximo del número de solicitudes simultáneas a un determinado origen a la vez. Si disminuye este valor, podrá reducir el número de errores de tiempo de espera o de limitación controlada. Sin embargo, también puede aumentar el uso de la memoria, ya que las solicitudes se ponen en cola hasta que un conector esté disponible.

En el siguiente ejemplo se muestra cómo establecer `maxSockets` para todos los objetos de servicio que cree. En este ejemplo se permiten hasta 25 conexiones simultáneas para cada punto de enlace de servicio.

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

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

Lo mismo se puede hacer por servicio.

```
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
  }
});
```

Cuando se utiliza el valor predeterminado de `https`, el SDK toma el valor `maxSockets` del `globalAgent`. Si el valor `maxSockets` no está definido o es `Infinity`, el SDK supondrá que el valor de `maxSockets` es 50.

Para obtener más información sobre cómo configurar `maxSockets` en Node.js, consulte la [documentación online de Node.js](#).

Reutilización de conexiones con Keep-Alive en Node.js

De forma predeterminada, el agente HTTP/HTTPS predeterminado de Node.js crea una nueva conexión TCP para cada nueva solicitud. Para evitar el costo que supone establecer una nueva conexión, puede reutilizar una conexión existente.

En el caso de las operaciones de corta duración, como las consultas de DynamoDB, la sobrecarga en latencia de la configuración de una conexión TCP puede ser mayor que la propia operación. Además, dado que el [cifrado en reposo](#) de DynamoDB está integrado con [AWS KMS](#), pueden producirse latencias desde la base de datos al tener que restablecer nuevas entradas de caché de AWS KMS para cada operación.

La forma más fácil de configurar SDK para JavaScript para que reutilice las conexiones TCP es establecer la variable de entorno `AWS_NODEJS_CONNECTION_REUSE_ENABLED` en 1. Esta característica se añadió en la versión [2.463.0](#).

También puede definir la propiedad `keepAlive` de un agente HTTP o HTTPS establecido en `true`, tal y como se muestra en el siguiente ejemplo.

```
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
  }
});
```

En el siguiente ejemplo, se muestra cómo se establece `keepAlive` únicamente para un cliente de 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
  }
});
```

Si `keepAlive` está habilitado, también puede establecer el retraso inicial de los paquetes TCP Keep-Alive con `keepAliveMsecs`, que de forma predeterminada es 1000ms. Consulte la [documentación de Node.js](#) para obtener más detalles.

Configuración de proxies para Node.js

Si no puede conectarse directamente a internet, el SDK para JavaScript admite el uso de proxies HTTP o HTTPS a través de un agente HTTP de terceros, como un [agente proxy](#). Para instalar el agente proxy, escriba lo siguiente en la línea de comandos.

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

Si decide utilizar otro proxy, primero siga las instrucciones de instalación y configuración de dicho proxy. Para utilizar este u otro proxy de terceros en su aplicación, debe especificar la propiedad `httpOptions` de `AWS.Config` para especificar el proxy que elija. En este ejemplo se muestra `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 obtener más información acerca de otras bibliotecas de proxy, consulte [npm, el administrador de paquetes de Node.js](#).

Registro de paquetes de certificados en Node.js

Los almacenes de confianza de Node.js incluyen los certificados necesarios para tener acceso a los servicios de AWS. En algunos casos, puede ser preferible incluir únicamente un conjunto específico de certificados.

En este ejemplo, se usa un certificado específico en el disco para crear un `https.Agent` que rechace las conexiones a menos que se proporcione el certificado designado. Luego, el `https.Agent` recién creado se utiliza para actualizar la configuración del 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
    })
  }
});
```

Consideraciones sobre los scripts de navegador

En los siguientes temas se describen consideraciones especiales para utilizar AWS SDK for JavaScript en scripts de navegador.

Temas

- [Creación del SDK para navegadores](#)
- [Cross-Origin Resource Sharing \(CORS, Uso compartido de recursos entre orígenes\)](#)

Creación del SDK para navegadores

El SDK para JavaScript se suministra como archivo JavaScript con soporte incluido para un conjunto predeterminado de servicios. Este archivo suele cargarse en scripts de navegador mediante una etiqueta `<script>` que hace referencia al paquete de SDK alojado. Sin embargo, es posible que necesite soporte para servicios que no sean el conjunto predeterminado o bien que necesite personalizar el SDK.

Si trabaja con el SDK fuera de un entorno que impone CORS en el navegador y si desea obtener acceso a todos los servicios que proporciona el SDK para JavaScript, puede crear una copia personalizada del SDK localmente clonando el repositorio y ejecutando las mismas herramientas de compilación que crean la versión alojada predeterminada del SDK. En las siguientes secciones se describen los pasos necesarios para compilar el SDK con servicios adicionales y versiones de API.

Temas

- [Uso del SDK Builder para compilar el SDK para JavaScript](#)
- [Uso de la CLI para compilar el SDK para JavaScript](#)
- [Creación de servicios y versiones de API concretos](#)
- [Compilación del SDK como dependencia con Browserify](#)

Uso del SDK Builder para compilar el SDK para JavaScript

La forma más sencilla de crear su propia compilación de AWS SDK para JavaScript consiste en utilizar la aplicación web del compilador del SDK en <https://sdk.amazonaws.com/builder/js>. Utilice el compilador del SDK para especificar servicios y las versiones de API que se deben incluir en la compilación.

Elija Select all services (Seleccionar todos los servicios) o elija Select default services (Seleccionar servicios predeterminados) como punto de partida desde el que puede añadir o eliminar servicios. Elija Development (Desarrollo) para facilitar la lectura del código o elija Minified (Minimizado) para crear una compilación minimizada para la implementación. Después de elegir los servicios y las versiones que desea incluir, elija Build (Compilar) para compilar y descargar el SDK personalizado.

Uso de la CLI para compilar el SDK para JavaScript

Para compilar el SDK para JavaScript utilizando la AWS CLI, primero tiene que clonar el repositorio de Git que contiene el código fuente del SDK. Debe tener Git y Node.js instalados en su equipo.

En primer lugar, clone el repositorio desde GitHub y cambie el directorio al directorio:

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

Después de clonar el repositorio, descargue los módulos de dependencia para el SDK y la herramienta de compilación:

```
npm install
```

Ahora puede compilar una versión empaquetada del SDK.

Compilación desde la línea de comandos

La herramienta del compilador se encuentra en `dist-tools/browser-builder.js`. Ejecute este script escribiendo:

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

Este comando compila el archivo `aws-sdk.js`. Este archivo está sin comprimir. De forma predeterminada, este paquete contiene solo el conjunto de servicios estándar.

Minimización de la salida de la compilación

Para reducir la cantidad de datos en la red, los archivos de JavaScript se pueden comprimir a través de un proceso denominado minimización. La minimización elimina comentarios, espacios innecesarios y otros caracteres que facilitan la lectura a las personas pero que no afectan a la ejecución del código. La herramienta del compilador puede producir salidas minimizadas o sin comprimir. Para minimizar la salida de la compilación, establezca la variable de entorno `MINIFY`:

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

Creación de servicios y versiones de API concretos

Puede seleccionar qué servicios desea compilar en el SDK. Para seleccionar servicios, especifique los nombres de los servicio, delimitados por comas, como si fueran parámetros. Por ejemplo, para compilar únicamente Amazon S3 y Amazon EC2, use el comando siguiente:

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

También puede seleccionar versiones de API específicas de los servicios compilados añadiendo el nombre de la versión después del nombre del servicio. Por ejemplo, para compilar las dos versiones de la API de Amazon DynamoDB, ejecute el comando siguiente:

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

Los identificadores de servicio y las versiones de API están disponibles en los archivos de configuración específicos de servicios en <https://github.com/aws/aws-sdk-js/tree/master/apis>.

Compilación de todos los servicios

Puede compilar todos los servicios y las versiones de la API incluyendo el parámetro `all`:

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

Compilación de servicios concretos

Para personalizar el conjunto de servicios seleccionados incluidos en la compilación, transfiera la variable de entorno `AWS_SERVICES` al comando Browserify que contiene la lista de servicios que quiere. El ejemplo siguiente compila los servicios Amazon EC2, Amazon S3 y DynamoDB.

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

Compilación del SDK como dependencia con Browserify

Node.js tiene un mecanismo basado en módulos para incluir código y funcionalidades de otros desarrolladores. JavaScript en navegadores web no admite de forma nativa este método modular. Sin embargo, con una herramienta llamada Browserify, puede utilizar el método de módulos Node.js y utilizar módulos escritos para Node.js en el navegador. Browserify compila las dependencias del módulo para un script de navegador en un único archivo JavaScript autónomo que puede utilizar en el navegador.

Puede compilar el SDK como una dependencia de biblioteca para cualquier script de navegador usando Browserify. Por ejemplo, el siguiente código Node.js requiere el SDK:

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

Este código de ejemplo se puede compilar en una versión compatible con navegador mediante Browserify:

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

La aplicación, incluidas sus dependencias del SDK, pasa entonces a estar disponible en el navegador a través de `browser-app.js`.

Para obtener más información acerca de Browserify, consulte el [sitio web de Browserify](#).

Cross-Origin Resource Sharing (CORS, Uso compartido de recursos entre orígenes)

El uso compartido de recursos entre orígenes o CORS es una característica de seguridad de los navegadores web modernos. Habilita a los navegadores web para que puedan negociar qué dominios pueden realizar solicitudes de sitios web o servicios externos. CORS es un factor importante que hay que tener en cuenta cuando se desarrollan aplicaciones de navegador con AWS SDK for JavaScript, ya que la mayoría de las solicitudes a recursos se envían a un dominio externo como, por ejemplo, el punto de conexión de un servicio web. Si su entorno de JavaScript aplica seguridad CORS, tiene que configurar CORS con el servicio.

CORS determina si se permitirá el uso compartido de recursos en una solicitud entre orígenes basándose en:

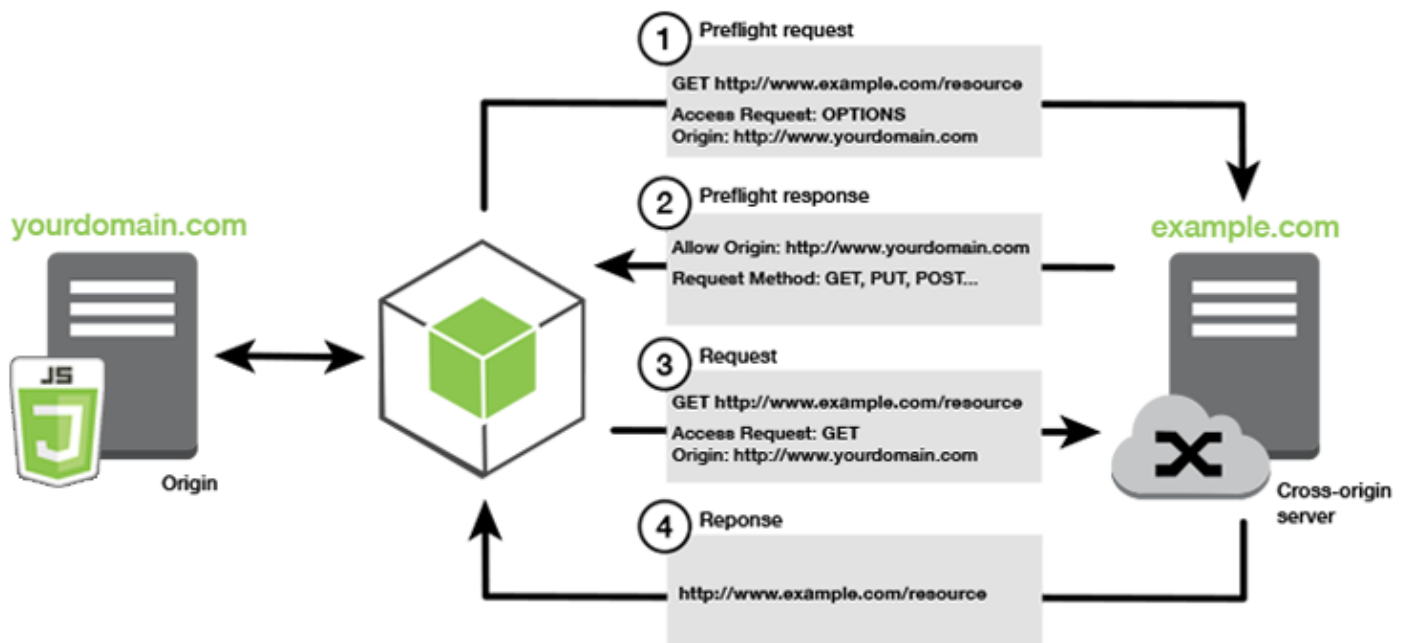
- El dominio específico que efectúa la solicitud.
- El tipo de solicitud HTTP que se realiza (GET, PUT, POST, DELETE, etc.).

Funcionamiento de CORS

En el caso más sencillo, el navegador script realiza una solicitud GET para obtener un recurso de un servidor que se encuentra en otro dominio. En función de cómo sea la configuración de CORS de dicho servidor, si la solicitud proviene de un dominio con permiso para enviar solicitudes GET, el servidor de orígenes cruzados responderá devolviendo el recurso que se ha solicitado.

Si el dominio de solicitud o el tipo de solicitud HTTP no está autorizado, se denegará la solicitud. Sin embargo, con CORS es posible realizar una solicitud preliminar antes de enviarla realmente. En dicho caso, se realiza una solicitud preliminar en la que se envía la operación de solicitud de acceso

OPTIONS. Si la configuración CORS del servidor de origen otorga acceso al dominio que realiza la solicitud, el servidor enviará una respuesta preliminar que contenga una lista de todos los tipos de solicitud HTTP que el dominio que realiza la solicitud puede hacer al recurso solicitado.



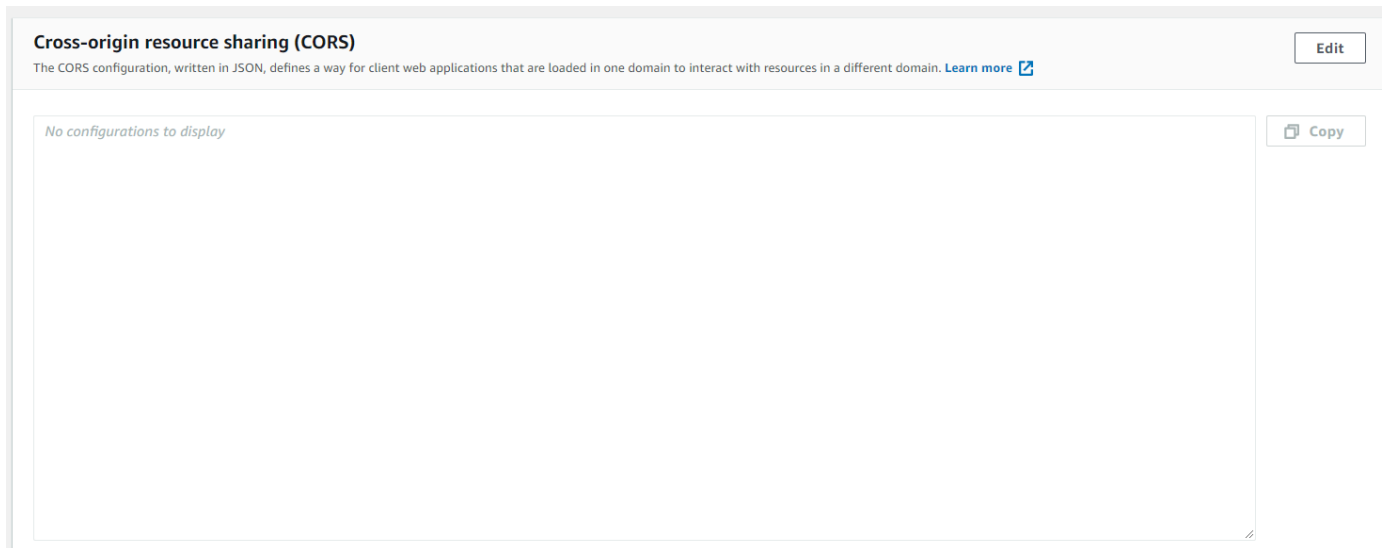
Configuración necesaria para CORS

Es preciso configurar CORS en los buckets de Amazon S3 para poder realizar operaciones en ellos. Puede que en algunos entornos JavaScript no se aplique CORS y, por consiguiente, no sea necesario configurarlo. Por ejemplo, si aloja su aplicación de un bucket de Amazon S3 y obtiene acceso a recursos de `*.s3.amazonaws.com` o algún otro punto de conexión específico, sus solicitudes no tendrán acceso a un dominio externo. Por lo tanto, esta configuración no necesita CORS. En este caso, se seguirá utilizando CORS para servicios que no sean Amazon S3.

Configuración de CORS para un bucket de Amazon S3

Puede configurar un bucket de Amazon S3 para utilizar CORS en la consola de Amazon S3.

1. En la consola de Amazon S3, elija el bucket que quiere editar.
2. Elija la pestaña Permisos y desplácese hasta el panel Compartir recursos entre orígenes (CORS).



3. Seleccione Editar y escriba su configuración de CORS en el Editor de configuraciones de CORS. A continuación, elija Guardar.

Una configuración de CORS es un archivo XML que contiene una serie de reglas dentro de una `<CORSRule>`. Una configuración puede tener hasta 100 reglas. Una regla se define con una de las siguientes etiquetas:

- `<AllowedOrigin>`, que especifica orígenes de dominios a los que permite realizar solicitudes de dominio cruzado.
- `<AllowedMethod>`, que especifica un tipo de solicitud que permite (GET, PUT, POST, DELETE, HEAD) en solicitudes de dominio cruzado.
- `<AllowedHeader>`, que especifica los encabezados que están permitidos en una solicitud preliminar.

Para ver las configuraciones de ejemplo, consulte [¿Cómo configuro el CORS en mi bucket?](#) en la Guía del usuario de Amazon Simple Storage Service.

Ejemplo de configuración de CORS

La siguiente muestra de configuración de CORS permite a un usuario ver, añadir, eliminar o actualizar objetos dentro de un bucket desde el dominio `example.org`, aunque se recomienda que defina el ámbito de `<AllowedOrigin>` al dominio de su sitio web. Puede especificar "*" para permitir cualquier origen.

⚠ Important

En la nueva consola S3, la configuración de CORS debe 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"
    ]
  }
]
```

```
}  
]
```

Esta configuración no autoriza al usuario a realizar acciones en el bucket. Habilita al modelo de seguridad del navegador para que permita una solicitud a Amazon S3. Los permisos tienen que configurarse a través de permisos de bucket o permisos de rol de IAM .

Puede utilizar `ExposeHeader` para permitir que el SDK lea los encabezados de respuestas que devuelve Amazon S3. Por ejemplo, si desea leer el encabezado `ETag` de una operación `PUT` o una carga multiparte, debe incluir la etiqueta `ExposeHeader` en su configuración, tal y como se muestra en el ejemplo anterior. El SDK solo puede obtener acceso a los encabezados que se exponen a través de la configuración de CORS. Si establece metadatos en el objeto, los valores se devuelven como encabezados con el prefijo `x-amz-meta-`, como `x-amz-meta-my-custom-header`, y también deben exponerse de la misma manera.

Agrupación de aplicaciones con Webpack

El uso de módulos de código por parte de aplicaciones web en scripts de navegador o Node.js crea dependencias. Estos módulos de código pueden tener dependencias propias, lo que se traduce en una colección de módulos interconectados que su aplicación necesita para funcionar. Para administrar dependencias, puede utilizar un instalador de módulos como Webpack.

El instalador de módulos Webpack analiza el código de la aplicación para buscar instrucciones `import` o `require` y crear agrupaciones que contengan todos los recursos que la aplicación necesita, para que los recursos se puedan servir fácilmente a través de la página web. El SDK para JavaScript se puede incluir en webpack como una de las dependencias que se deben incluir en el paquete de salida.

Para obtener más información acerca de Webpack, consulte el [instalador de módulos Webpack](#) en GitHub.

Instalación de Webpack

Para instalar el instalador de módulos Webpack, en primer lugar debe tener npm, el administrador de paquetes de Node.js, instalado. Escriba el comando siguiente para instalar la interfaz de línea de comandos (CLI) de Webpack y el módulo JavaScript.

```
npm install webpack
```

Puede que también tenga que instalar un complemento de Webpack que le permita cargar archivos JSON. Escriba el comando siguiente para instalar el complemento del cargador de JSON.

```
npm install json-loader
```

Configuración de Webpack

De forma predeterminada, Webpack busca un archivo de JavaScript denominado `webpack.config.js` en el directorio raíz del proyecto. Este archivo especifica sus opciones de configuración. A continuación se muestra un ejemplo de un archivo de configuración `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']
      }
    ]
  }
}
```

En este ejemplo, `browser.js` se especifica como punto de entrada. El punto de entrada es el archivo que Webpack utiliza para comenzar a buscar módulos importados. El nombre de archivo de la salida se especifica como `bundle.js`. Este archivo de salida contendrá todo el JavaScript que la aplicación necesita ejecutar. Si el código especificado en el punto de entrada importa o solicita otros módulos como, por ejemplo, el SDK para JavaScript, dicho código se empaqueta sin necesidad de especificarlo en la configuración.

La configuración que hay en el complemento `json-loader` que se ha instalado anteriormente especifica a Webpack cómo importar archivos JSON. De forma predeterminada, Webpack solo es compatible con JavaScript, pero utiliza cargadores para añadir soporte para importar otros tipos de archivos. Dado que el SDK para JavaScript usa archivos JSON intensivamente, webpack da un error al generar el paquete si `json-loader` no está incluido.

Ejecución de Webpack

Para desarrollar una aplicación para utilizar Webpack, añada lo siguiente al objeto `scripts` de su archivo `package.json`.

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

A continuación se muestra `package.json` de ejemplo que demuestra cómo añadir 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 compilar su aplicación, escriba el comando siguiente.

```
npm run build
```

Luego, el instalador del módulo Webpack genera el archivo JavaScript que ha especificado en el directorio raíz del proyecto.

Uso del grupo de Webpack

Para utilizar el grupo en un script de navegador, puede incorporar el paquete con una etiqueta `<script>` tal y como se muestra en el siguiente ejemplo.

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

Importación de servicios individuales

Uno de los beneficios de Webpack es que analiza las dependencias de su código y agrupa únicamente el código que su aplicación necesita. Si está utilizando el SDK para JavaScript, agrupando solo las partes del SDK que la aplicación usa realmente, puede reducir considerablemente el tamaño de la salida de webpack.

Observe el siguiente ejemplo del código que se utiliza para crear un objeto de servicio de 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'});
```

La función `require()` especifica todo el SDK. Un paquete webpack que se haya generado con este código incluirá el SDK completo, pero este no es necesario cuando solo se utiliza la clase de cliente de Amazon S3. El tamaño del paquete será considerablemente más pequeño si solo se incluye la parte del SDK que necesita para el servicio de Amazon S3. Incluso establecer la configuración no requiere el SDK completo, ya que puede establecer los datos de la configuración en el objeto de servicio de Amazon S3.

A continuación, se muestra cómo se ve este mismo código cuando se incluye solo la parte de Amazon S3 del 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}
});
```

Agrupación para Node.js

Puede utilizar Webpack para generar paquetes que se ejecutan en Node.js especificándolo como destino en la configuración.

```
target: "node"
```

Esto resulta útil al ejecutar una aplicación de Node.js en un entorno donde el espacio en el disco es limitado. A continuación se muestra un ejemplo de configuración de `webpack.config.js` con Node.js especificado como destino de salida.

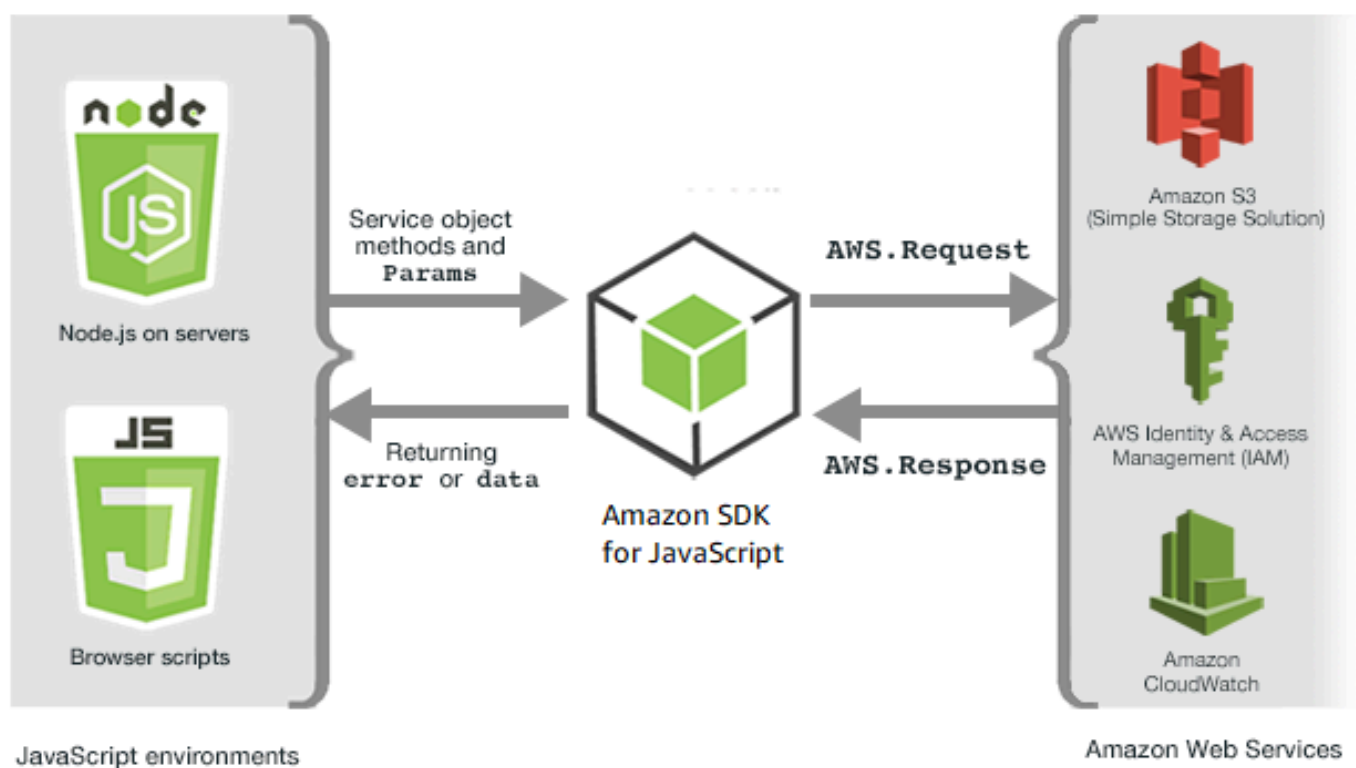
```
// Import path for resolving file paths
var path = require('path');
module.exports = {
  // Specify the entry point for our app
  entry: [
```

```
    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']
      }
    ]
  }
}
```

Uso de los servicios en el SDK para JavaScript

El AWS SDK para JavaScript proporciona acceso a los servicios que admite a través de una colección de clases del cliente. A partir de estas clases del cliente, se crean objetos de interfaz de servicios, normalmente denominados objetos de servicio. Cada servicio de AWS compatible tiene una o varias clases del cliente que ofrecen API de bajo nivel para utilizar características de servicio y recursos. Por ejemplo, las API de Amazon DynamoDB están disponibles a través de la clase `AWS.DynamoDB`.

Los servicios expuestos a través del SDK para JavaScript siguen el patrón de solicitud-respuesta para intercambiar mensajes con las aplicaciones que llaman. En este patrón, el código que invoca un servicio envía una solicitud HTTP/HTTPS a un punto de conexión para el servicio. La solicitud contiene parámetros necesarios para invocar correctamente la característica específica a la que se llama. El servicio que se invoca genera una respuesta que se devuelve al solicitante. La respuesta contiene datos si la operación se ha realizado correctamente o bien contiene información de error si dicha operación ha generado errores.



La invocación de un servicio de AWS incluye el ciclo de vida completo de solicitud y respuesta de una operación en un objeto de servicio, incluidos todos los reintentos. El objeto `AWS.Request` encapsula una solicitud en el SDK. La respuesta se encapsula en el SDK mediante el objeto

AWS .Response, que se proporciona al solicitante a través de una de varias técnicas como, por ejemplo, una función de devolución de llamada o una promesa de JavaScript.

Temas

- [Creación y llamada a objetos de servicio](#)
- [Registro de llamadas del AWS SDK para JavaScript](#)
- [Llamadas asíncronas a servicios](#)
- [Uso del objeto de respuesta](#)
- [Uso de JSON](#)
- [Estrategia de reintentos en la versión 2 de AWS SDK para JavaScript](#)

Creación y llamada a objetos de servicio

La API de JavaScript admite la mayoría de los servicios de AWS disponibles. Cada clase de servicio de la API de JavaScript proporciona acceso a todas las llamadas a la API de su servicio. Para obtener más información acerca de clases de servicios, las operaciones y los parámetros en la API de JavaScript, consulte la [referencia de la API](#).

Cuando utiliza el SDK en Node.js, añada el paquete del SDK a su aplicación mediante `require`, que ofrece soporte para todos los servicios actuales.

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

Cuando utiliza el SDK con JavaScript de navegador, carga el paquete del SDK en los scripts de navegador mediante el paquete de SDK alojado en AWS. Para cargar el paquete del SDK, añada el siguiente elemento `<script>`.

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

Para encontrar el `SDK_VERSION_NUMBER` actual, consulte la referencia de API para el SDK para JavaScript en la [Guía de referencia de la API de AWS SDK para JavaScript](#).

El paquete de SDK alojado predeterminado admite un subconjunto de los servicios de AWS disponibles. Para obtener una lista de los servicios predeterminados en el paquete del SDK hospedado para el navegador, consulte los [servicios compatibles](#) en la referencia de la API. Puede utilizar el SDK con otros servicios si la comprobación de seguridad CORS está deshabilitada. En

este caso, puede crear una versión personalizada del SDK para incluir los servicios adicionales que necesita. Para obtener más información acerca de la creación de una versión personalizada del SDK, consulte [Creación del SDK para navegadores](#).

Necesidad de uso de servicios individuales

Si necesita usar el SDK para JavaScript tal y como se muestra anteriormente, tiene que incluir todo el SDK en el código. O bien, puede elegir exigir solo los servicios individuales que su código utiliza. Considere el siguiente código que se utiliza para crear un objeto de servicio de 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'});
```

En el ejemplo anterior, la función `require` especifica todo el SDK. La cantidad de código que debe transportarse a través de la red, así como la sobrecarga de memoria del código serían considerablemente inferiores si solo se incluyese una parte del SDK que necesita para el servicio de Amazon S3. Para solicitar un servicio individual, llame a la función `require` tal y como se muestra, incluido el constructor de servicio todo en minúsculas.

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

A continuación se muestra cómo se ve el código para crear el objeto de servicio de Amazon S3 anterior cuando incluye solo la parte de Amazon S3 del 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}
});
```

Puede seguir obteniendo acceso al espacio de nombres de AWS global sin tener todos los servicios asociados a él.

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

Se trata de una técnica útil cuando se aplica la misma configuración en varios servicios individuales para, por ejemplo, proporcionar las mismas credenciales a todos los servicios. La necesidad de servicios individuales debería reducir el tiempo de carga y el consumo de memoria en Node.js. Cuando se realiza con una herramienta de agrupación como Browserify o Webpack, el uso de servicios individuales hace que el SDK se reduzca a una fracción del tamaño completo. Esto es útil en los entornos con memoria o espacio de disco restringidos, como un dispositivo IoT o una función de Lambda.

Creación de objetos de servicio

Para obtener acceso a características de servicio a través de la API de JavaScript, primero debe crear un objeto de servicio a través del cual obtener acceso a un conjunto de características que proporciona el cliente subyacente. Por lo general cada servicio proporciona una clase de cliente; sin embargo, algunos servicios dividen el acceso a sus características entre varias clases de cliente.

Para utilizar una característica, debe crear una instancia de la clase que proporciona acceso a dicha característica. En el siguiente ejemplo se muestra cómo crear un objeto de servicio para DynamoDB a partir de la clase de cliente `AWS.DynamoDB`.

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

De forma predeterminada un objeto de servicio se configura con la configuración global que también se utiliza para configurar el SDK. Sin embargo, puede configurar un objeto de servicio con datos de configuración de tiempo de ejecución específicos de dicho objeto de servicio. Los datos de configuración específicos del servicio se aplican después de aplicar los valores de la configuración global.

En el ejemplo siguiente, se crea un objeto de servicio de Amazon EC2 con la configuración para una región específica, pero aparte de esto se utiliza la configuración global.

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

Además de admitir la configuración específica del servicio aplicada a un objeto de servicio individual, también puede aplicar la configuración específica del servicio a todos los objetos de servicio recién

creados de una determinada clase. Por ejemplo, para configurar todos los objetos de servicio creados a partir de la clase de Amazon EC2 para utilizar la región Oeste de EE. UU. (Oregón) (us-west-2), añade lo siguiente al objeto de configuración global `AWS.config`.

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

Bloqueo de la versión de API de un objeto de servicio

Puede bloquear un objeto de servicio en una versión de API específica de un servicio especificando la opción `apiVersion` al crear el objeto. En el siguiente ejemplo, se crea un objeto de servicio de DynamoDB que está bloqueado para una versión de API específica.

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

Para obtener más información acerca del bloqueo de la versión de la API de un objeto de servicio, consulte [Bloqueo de versiones de la API](#).

Especificación de parámetros de objetos de servicio

Al llamar a un método de un objeto de servicio, transfiera los parámetros en JSON según los requiera la API. Por ejemplo, en Amazon S3, para obtener un objeto para un bucket y una clave especificados, transfiera los siguientes parámetros al método `getObject`. Para obtener más información acerca de cómo transferir parámetros JSON, consulte [Uso de JSON](#).

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

Para obtener más información acerca de los parámetros de Amazon S3, consulte [Class: AWS.S3](#) en la referencia de la API.

Además, puede vincular valores a parámetros individuales al crear un objeto de servicio mediante el parámetro `params`. El valor del parámetro `params` de objetos de servicio es un mapa que especifica uno o varios valores de parámetros definidos por el objeto de servicio. En el siguiente ejemplo se muestra el parámetro `Bucket` de un objeto de servicio de Amazon S3 vinculado a un bucket llamado `amzn-s3-demo-bucket`.

```
var s3bucket = new AWS.S3({params: {Bucket: 'amzn-s3-demo-bucket'}, apiVersion: '2006-03-01'});
```

Al vincular el objeto de servicio a un bucket, el objeto de servicio `s3bucket` trata el valor del parámetro `amzn-s3-demo-bucket` como un valor predeterminado que ya no necesita especificarse para operaciones posteriores. Todos los valores de parámetros vinculados se omiten cuando se utiliza el objeto para operaciones donde el valor del parámetro no se puede aplicar. Puede anular este parámetro vinculado cuando hace llamadas en el objeto de servicio, especificando un nuevo valor.

```
var s3bucket = new AWS.S3({ params: {Bucket: 'amzn-s3-demo-bucket'}, apiVersion:
  '2006-03-01' });
s3bucket.getObject({Key: 'keyName'});
// ...
s3bucket.getObject({Bucket: 'amzn-s3-demo-bucket3', Key: 'keyOtherName'});
```

Los detalles sobre los parámetros disponibles para cada método se encuentran en la referencia de la API.

Registro de llamadas del AWS SDK para JavaScript

El AWS SDK para JavaScript dispone de un registrador integrado, por lo que puede registrar las llamadas a la API que realiza con el SDK para JavaScript.

Para activar el registrador e imprimir las entradas de registro en la consola, añada la siguiente instrucción al código.

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

Este es un ejemplo del resultado del registro.

```
[AWS s3 200 0.185s 0 retries] createMultipartUpload({ Bucket: 'amzn-s3-demo-logging-
bucket', Key: 'issues_1704' })
```

Uso de registradores de terceros

También puede utilizar un registrador de terceros, siempre y cuando tenga operaciones `log()` o `write()` para escribir en un servidor o archivo de registro. Debe instalar y configurar su registrador personalizado siguiendo las instrucciones indicadas para poder utilizarlo con el SDK para JavaScript.

Logplease es uno de estos registradores que se pueden utilizar en scripts de navegador o en Node.js. En Node.js, puede configurar logplease para escribir entradas de registro en un archivo de registro. También puede usarlo con webpack.

Al usar un registrador de terceros, establezca todas las opciones antes de asignar el registrador a `AWS.Config.logger`. Por ejemplo, el código siguiente especifica un archivo de registro externo y establece el nivel de registro de 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 obtener más información acerca de logplease, consulte la sección del [registrador de JavaScript sencillo logplease](#) en GitHub.

Llamadas asíncronas a servicios

Todas las solicitudes que se realizan a través del SDK son asíncronas. Es importante recordarlo cuando se escriben scripts de navegador. La ejecución de JavaScript en un navegador web normalmente tiene un solo subproceso de ejecución. Después de realizar una llamada asíncrona a un servicio de AWS, el script de navegador continúa ejecutándose y, en el proceso, puede intentar ejecutar código que dependa de ese resultado asíncrono antes de volver.

La realización de llamadas asíncronas a un servicio de AWS incluye la administración de dichas llamadas de modo que su código no intente utilizar datos antes de que estos estén disponibles. En los temas de esta sección se explica la necesidad de administrar llamadas asíncronas y técnicas diferentes de detalles que puede utilizar para administrarlas.

Temas

- [Administración de llamadas asíncronas](#)

- [Uso de una función de devolución de llamada anónima](#)
- [Uso de un objeto de un agente de escucha de eventos de objetos de solicitud](#)
- [Uso de async/await](#)
- [Uso de promesas de JavaScript](#)

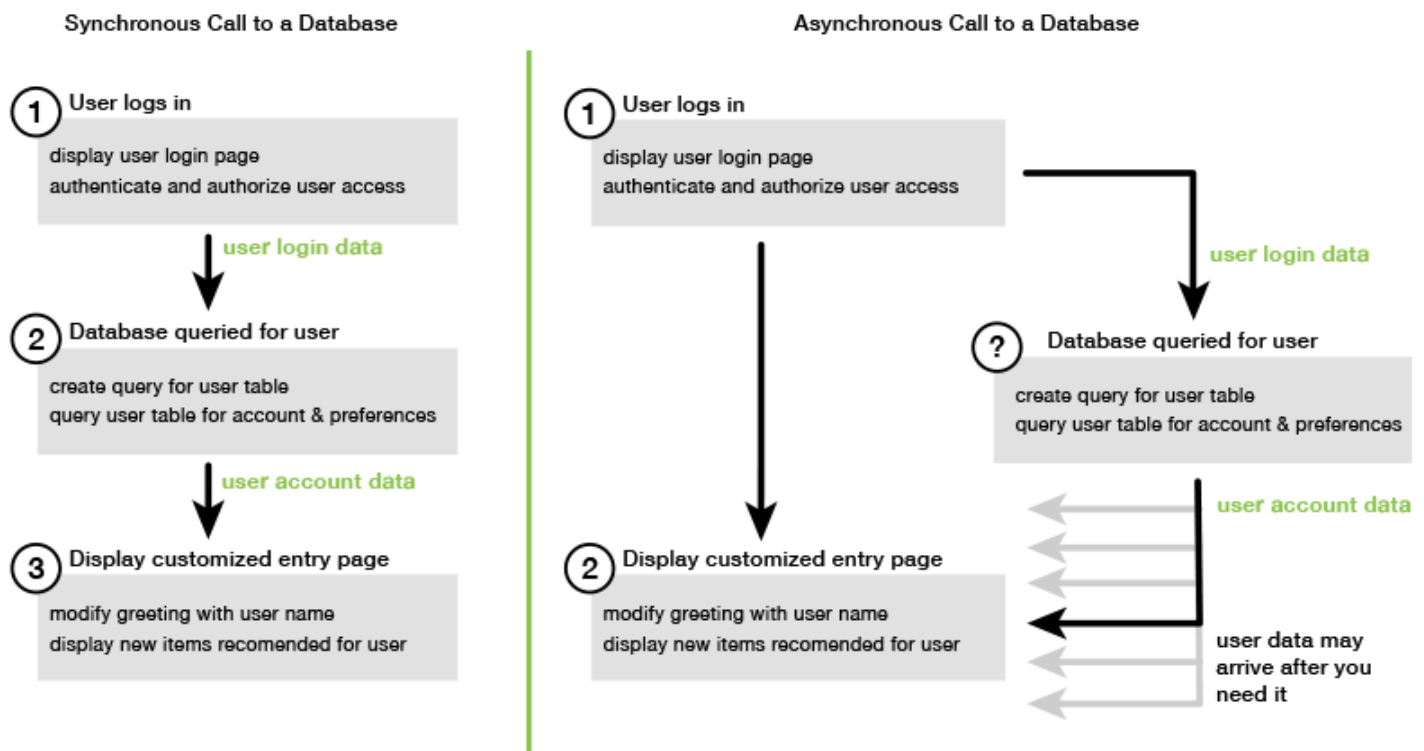
Administración de llamadas asíncronas

Por ejemplo, la página de inicio de un sitio web de e-commerce permite iniciar sesión a los clientes que regresan. Parte del beneficio para los clientes que inician sesión es que, después de iniciar sesión, el sitio se personaliza a sí mismo para adaptarse a sus preferencias concretas. Para que esto suceda:

1. El cliente tiene que iniciar sesión y validarse con sus credenciales de inicio de sesión.
2. Las preferencias del cliente se solicitan a partir de una base de datos del cliente.
3. La base de datos proporciona las preferencias del cliente que se utilizan para personalizar el sitio antes de que se cargue la página.

Si dichas tareas se ejecutan de forma síncrona, cada una tiene que finalizar antes de que la siguiente pueda comenzar. La página web no puede acabar de cargarse hasta que no lleguen las preferencias del cliente desde la base de datos. Sin embargo, después de que la consulta de la base de datos se envíe al servidor, la recepción de los datos del cliente se puede retrasar o incluso generar errores debido a atascos en la red, un tráfico de base de datos excepcionalmente alto o una conexión de dispositivo móvil de mala calidad.

Para evitar que el sitio web se congele en estas condiciones, llame a la base de datos de forma asíncrona. Después de que se ejecute la llamada a la base de datos y se envíe su solicitud asíncrona, el código sigue ejecutándose según lo previsto. Si no administra correctamente la respuesta de una llamada asíncrona, el código puede intentar utilizar información que espera recibir de la base de datos cuando dichos datos todavía no están disponibles.



Uso de una función de devolución de llamada anónima

Cada método de objeto de servicio que crea un objeto `AWS.Request` puede aceptar una función de devolución de llamada anónima como último parámetro. La firma de dicha función de devolución de llamada es:

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

Esta función de devolución de llamada se ejecuta cuando se devuelve una respuesta correcta o datos de error. Si la llamada al método se realiza correctamente, el contenido de la respuesta está disponible en la función de devolución de llamada en el parámetro `data`. Si la llamada no se realiza correctamente, se proporcionan los detalles sobre el error en el parámetro `error`.

Normalmente el código contenido en la función de devolución de llamada realiza una prueba para detectar errores. Si el resultado de la prueba devuelve errores, los procesará. Si no se devuelve ningún error, el código recuperará los datos de la respuesta en el parámetro `data`. La forma básica de la función de devolución de llamada es similar a la de este ejemplo.

```
function(error, data) {
```

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

En el ejemplo anterior, los detalles del error o de los datos devueltos se registran en la consola. A continuación se muestra un ejemplo que muestra una función de devolución de llamada transferida como parte de una llamada a un método en un objeto de servicio.

```
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
    }
});
```

Acceso a los objetos de solicitud y respuesta

Dentro de la función de devolución de llamada, la palabra clave de JavaScript `this` se refiere al objeto `AWS.Response` subyacente para la mayoría de los servicios. En el ejemplo siguiente, la propiedad `httpResponse` de un objeto `AWS.Response` se utiliza dentro de una función de devolución de llamada para registrar los datos de respuesta sin procesar y los encabezados para ayudar con la depuración.

```
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
    }
});
```

Además, debido a que el objeto `AWS.Response` tiene una propiedad `Request` que contiene la `AWS.Request` que se ha enviado a través de la llamada al método original, también puede obtener acceso a los detalles de la solicitud que se realizó.

Uso de un objeto de un agente de escucha de eventos de objetos de solicitud

Si no crea ni transfiere una función de devolución de llamada anónima como parámetro cuando llama a un método de objeto de servicio, la llamada al método genera un objeto `AWS.Request` que debe enviarse manualmente usando su método `send`.

Para procesar la respuesta, debe crear un agente de escucha de eventos para que el objeto `AWS.Request` registre una función de devolución de llamada para la llamada al método. En el siguiente ejemplo se muestra cómo crear el objeto `AWS.Request` para llamar a un método de objeto de servicio y al agente de escucha de eventos para que la devolución sea correcta.

```
// 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();
```

Una vez que se ha llamado al método `send` en el objeto `AWS.Request`, el controlador de eventos se ejecuta cuando el objeto de servicio recibe un objeto `AWS.Response`.

Para obtener más información acerca del objeto `AWS.Request`, consulte [Class: AWS.Request](#) en la referencia de la API. Para obtener más información acerca del objeto `AWS.Response`, consulte [Uso del objeto de respuesta](#) o [Class: AWS.Response](#) en la referencia de la API.

Encadenar varias devoluciones de llamadas

Puede registrar varias devoluciones de llamada en cualquier objeto de solicitud. Se pueden registrar varias devoluciones de llamada para diferentes eventos o para el mismo evento. También puede encadenar devoluciones de llamadas, tal y como se muestra en el siguiente ejemplo.

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

Eventos de finalización de objetos de solicitud

El objeto `AWS.Request` plantea estos eventos de finalización en función de la respuesta de cada método de operación de servicio:

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

Puede registrar una función de devolución de llamada en respuesta a cualquiera de estos eventos. Para obtener una lista completa de todos los eventos de objetos de solicitud, consulte [Class: AWS.Request](#) en la referencia de la API.

El evento `success`

El evento `success` se plantea cuando se recibe una respuesta de éxito desde el objeto de servicio. Aquí se muestra cómo registrar una función de devolución de llamada para este evento.

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

La respuesta proporciona una propiedad `data` que contiene los datos de respuesta serializada del servicio. Por ejemplo, la siguiente llamada al método `listBuckets` del objeto de servicio de Amazon S3

```
s3.listBuckets.on('success', function(response) {
```

```
console.log(response.data);
}).send();
```

devuelve la respuesta y luego imprime el siguiente contenido de la propiedad `data` a la consola.

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

El evento `error`

El evento `error` se plantea cuando se recibe una respuesta de error desde el objeto de servicio. Aquí se muestra cómo registrar una función de devolución de llamada para este evento.

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

Cuando el evento `error` se produce, el valor de la propiedad `data` de la respuesta es `null` y la propiedad `error` contiene los datos del error. El objeto `error` asociado se transfiere como primer parámetro a la función de devolución de llamada registrada. Por ejemplo, el código siguiente:

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

devuelve el error y luego imprime los datos de error siguientes a la consola.

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

El evento `complete`

El evento `complete` se produce cuando una llamada al objeto de servicio ha terminado, independientemente de si la llamada se traduce en éxito o en error. Aquí se muestra cómo registrar una función de devolución de llamada para este evento.

```
request.on('complete', function(response) {
```

```
// event handler code
});
```

Utilice la devolución de llamada del evento `complete` para gestionar cualquier limpieza de solicitud que debe ejecutarse independientemente de si el resultado ha sido un éxito o un error. Si utiliza los datos de respuesta dentro de una devolución de llamada para el evento `complete`, en primer lugar, compruebe las propiedades `response.data` o `response.error` antes de intentar obtener acceso a una de ellas, tal y como se muestra en el siguiente ejemplo.

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

Eventos de HTTP de objetos de solicitud

El objeto `AWS.Request` plantea estos eventos HTTP en función de la respuesta de cada método de operación de servicio:

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

Puede registrar una función de devolución de llamada en respuesta a cualquiera de estos eventos. Para obtener una lista completa de todos los eventos de objetos de solicitud, consulte [Class: AWS.Request](#) en la referencia de la API.

El evento `httpHeaders`

El evento `httpHeaders` se produce cuando el servidor remoto envía los encabezados. Aquí se muestra cómo registrar una función de devolución de llamada para este evento.

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

El parámetro `statusCode` para la función de devolución de llamada es el código de estado HTTP. El parámetro `headers` contiene los encabezados de la respuesta.

El evento `httpData`

El `httpData` evento se plantea para transmitir en streaming los paquetes de datos de respuesta desde el servicio. Aquí se muestra cómo registrar una función de devolución de llamada para este evento.

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

Este evento se suele utilizar para recibir respuestas de gran tamaño en fragmentos cuando cargar toda la respuesta en la memoria no es práctico. Este evento tiene un parámetro `chunk` adicional que contiene una parte de los datos reales del servidor.

Si registra una devolución de llamada para el evento `httpData`, la propiedad `data` de la respuesta contiene toda la salida serializada de la solicitud. Debe eliminar el agente de escucha `httpData` predeterminado si no tiene la cantidad adicional de memoria y análisis adicional para los controladores integrados.

Los eventos `httpDownloadProgress` y `httpUploadProgress`

El evento `httpUploadProgress` se produce cuando la solicitud HTTP ha cargado más datos. Igualmente, el evento `httpDownloadProgress` se produce cuando la solicitud HTTP ha descargado más datos. Aquí se muestra cómo registrar una función de devolución de llamada para estos eventos.

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

El parámetro `progress` para la función de devolución de llamada contiene un objeto con los bytes cargados y totales de la solicitud.

El evento `httpError`

El evento `httpError` se produce cuando la solicitud HTTP genera un error. Aquí se muestra cómo registrar una función de devolución de llamada para este evento.

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

El parámetro `error` para la función de devolución de llamada contiene el error que se generó.

El evento `httpDone`

El evento `httpDone` se produce cuando el servidor termina el envío de datos. Aquí se muestra cómo registrar una función de devolución de llamada para este evento.

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

Uso de `async/await`

Puede utilizar el patrón `async/await` en sus llamadas al AWS SDK para JavaScript. La mayoría de las funciones que aceptan una devolución de llamada no devuelven una promesa. Como solo se utilizan funciones `await` que devuelven una promesa, para utilizar el patrón `async/await` hay que encadenar el método `.promise()` hasta el final de la llamada y eliminar la devolución de llamada.

En el siguiente ejemplo, se usa `async/await` para enumerar todas las tablas de Amazon DynamoDB en `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

No todos los navegadores admiten `async/await`. Consulte las [funciones asíncronas](#) para obtener una lista de navegadores compatibles con `async/await`.

Uso de promesas de JavaScript

El método `AWS.Request.promise` proporciona una forma de llamar a una operación de servicio y administrar el flujo asíncrono en lugar de utilizar devoluciones de llamada. En Node.js y en los scripts de navegador, se devuelve un objeto `AWS.Request` cuando se llama a una operación de servicio sin una función de devolución de llamada. Puede llamar al método `send` de la solicitud para realizar la llamada de servicio.

Sin embargo, `AWS.Request.promise` comienza inmediatamente la llamada de servicio y devuelve una promesa que se cumple con la propiedad `data` de la respuesta o se rechaza con la propiedad `error` de la respuesta.

```
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 */
  }
);
```

En el siguiente ejemplo se devuelve una promesa que se cumple con un objeto `data` o se rechaza con un objeto `error`. Con el uso de las promesas, una única devolución de llamada no es responsable de la detección de errores. En su lugar, se llama a la devolución de llamada correcta en función del éxito o el error de una solicitud.

```
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);
});
```

Coordinación de varias promesas

En algunas situaciones, el código debe realizar varias llamadas asíncronas que requieren acción solo cuando todas han tenido una devolución correcta. Si administra estas llamadas a métodos asíncronos individuales con promesas, puede crear una promesa adicional que utilice el método `all`. Este método cumple esta promesa paraguas en el momento en que transfiere la matriz de promesas al método siempre y cuando dicha matriz de promesas se cumpla. Se transfiere a la función de devolución de llamada una matriz de los valores de las promesas que se transfieren al método `all`.

En el siguiente ejemplo, una función de Lambda tiene que realizar tres llamadas asíncronas a Amazon DynamoDB, pero solo puede completarse después de que las promesas para cada llamada se cumplan.

```
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);
});
```

Compatibilidad del navegador y Node.js con las promesas

La compatibilidad con promesas de JavaScript (ECMAScript 2015) depende del motor y la versión de JavaScript en los que se ejecuta el código. Para ayudar a determinar la compatibilidad con las promesas de JavaScript en cada entorno donde el código necesita ejecutarse, consulte la [tabla de compatibilidad ECMAScript](#) en GitHub.

Uso de otras implementaciones de promesas

Además de la implementación de promesas nativa de ECMAScript 2015, también puede utilizar bibliotecas de promesas de terceros, como:

- [bluebird](#)
- [RSVP](#)
- [Q](#)

Estas bibliotecas de promesas opcionales pueden ser útiles si necesita que el código se ejecute en entornos que no admitan la implementación de promesas nativas en ECMAScript 5 y en ECMAScript de 2015.

Para utilizar una biblioteca de promesas de terceros, establezca una dependencia de promesas en el SDK llamando al método `setPromisesDependency` del objeto de configuración global. En los scripts de navegador, asegúrese de cargar la biblioteca de promesas de terceros antes de cargar el SDK. En el siguiente ejemplo, el SDK está configurado para utilizar la implementación en la biblioteca de promesas `bluebird`.

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

Para volver a utilizar la implementación de promesas nativa del motor de JavaScript, vuelva a llamar a `setPromisesDependency` y transfiera `null` en vez de un nombre de biblioteca.

Uso del objeto de respuesta

Después de llamar a un método objeto de servicio, este devuelve un objeto `AWS.Response` transfiriéndolo a su función de devolución de llamada. Puede obtener acceso al contenido de la respuesta a través de las propiedades del objeto `AWS.Response`. Puede usar dos propiedades del objeto `AWS.Response` para obtener acceso al contenido de la respuesta:

- `dataPropiedad`
- `errorPropiedad`

Cuando se utiliza el mecanismo de devolución de llamada estándar, estos dos propiedades se proporcionan como parámetros en la función de devolución de llamada anónima tal y como se muestra en el siguiente ejemplo.

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

Acceso a los datos devueltos en el objeto de respuesta

La propiedad `data` del objeto `AWS.Response` contiene los datos serializados devueltos por la solicitud de servicio. Cuando la solicitud se realiza correctamente, la propiedad `data` incluye un objeto que contiene un mapa a los datos devueltos. La propiedad `data` puede ser un valor nulo si se produce un error.

A continuación se muestra un ejemplo llamada al método `getItem` de una tabla de DynamoDB para recuperar el nombre de un archivo de imagen y utilizarlo como parte de un juego.

```
// 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();
```

En este ejemplo, la tabla de DynamoDB es una búsqueda de imágenes que muestra los resultados de una máquina tragaperras según lo especificado por los parámetros en `slotParams`.

Tras una llamada del método `getItem` realizada correctamente, la propiedad `data` del objeto `AWS.Response` contiene un objeto `Item` devuelto por DynamoDB. El acceso a los datos devueltos se efectúa de acuerdo con el parámetro `ProjectionExpression` de la solicitud, que en este caso significa el miembro `imageFile` del objeto `Item`. Debido a que el miembro `imageFile` contiene un valor de cadena, el usuario obtiene acceso al nombre de archivo de la misma imagen a través del valor del elemento secundario `S` de `imageFile`.

Paginación por los datos devueltos

En ocasiones el contenido de la propiedad `data` que una solicitud de servicio devuelve abarca varias páginas. Puede obtener acceso a la siguiente página de datos llamando al método `response.nextPage`. Este método envía una nueva solicitud. La respuesta de la solicitud se puede capturar con una devolución de llamada o con agentes de escucha de éxito y error.

Puede comprobar si los datos devueltos por una solicitud de servicio tienen páginas de datos adicionales llamando al método `response.hasNextPage`. Este método devuelve un valor booleano para indicar si la llamada a `response.nextPage` devuelve datos adicionales.

```
s3.listObjects({Bucket: 'bucket'}).on('success', function handlePage(response) {
  // do something with response.data
  if (response.hasNextPage()) {
    response.nextPage().on('success', handlePage).send();
  }
}).send();
```

Acceso a información de error desde un objeto de respuesta

La propiedad `error` del objeto `AWS.Response` contiene los datos de error disponibles en caso de que se produzca un error de servicio o un error de transferencia. El error devuelto adopta el siguiente formato.

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

En caso de que se produzca un error, el valor de la propiedad `data` es `null`. Si gestiona eventos que pueden estar en un estado de error, compruebe siempre si la propiedad `error` se estableció antes de intentar obtener acceso al valor de la propiedad `data`.

Acceso al objeto de solicitud de origen

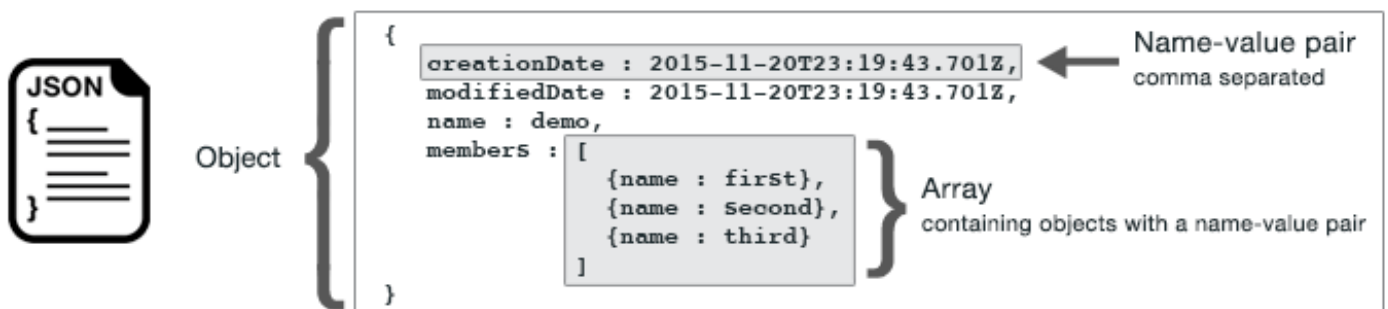
La propiedad `request` proporciona acceso al objeto `AWS.Request` de origen. Puede ser útil para hacer referencia al objeto `AWS.Request` original para obtener acceso a los parámetros originales que ha enviado. En el ejemplo siguiente, la propiedad `request` se utiliza para obtener acceso al parámetro `Key` de la solicitud de servicio original.

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

Uso de JSON

JSON es un formato para el intercambio de datos que pueden leer tanto humanos como las máquinas. Aunque el nombre JSON es el acrónimo de JavaScript Object Notation (notación de objetos JavaScript), el formato de JSON es independiente de cualquier lenguaje de programación.

El SDK para JavaScript utiliza JSON para enviar datos a objetos de servicio cuando realiza solicitudes y recibe datos de objetos de servicio como JSON. Para obtener más información sobre JSON, consulte json.org.



JSON representa los datos de dos formas:

- Un objeto, que es una colección sin ordenar de pares de nombre-valor. Un objeto se define entre las llaves izquierda (`{`) y derecha (`}`). Cada par de nombre-valor comienza por el nombre, seguido de dos puntos, seguido del valor. Los pares de nombre-valor están separados por comas.

- Una matriz, que es una colección ordenada de valores. Una matriz se define entre los corchetes izquierdo ([) y derecho (]). Los elementos de la matriz están separados por comas.

A continuación, se muestra un ejemplo de un objeto JSON que contiene una matriz de objetos en la que los objetos representan las naipes de un juego de cartas. Cada carta está definida con dos pares de nombre-valor, uno que especifica un valor único para identificar la carta y otra que especifica una dirección URL que apunta a la imagen de la carta correspondiente.

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

Parámetros de JSON como objeto de servicio

A continuación, se muestra un ejemplo de JSON sencillo que se utiliza para definir los parámetros de una llamada a un objeto de servicio de Lambda.

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

El objeto `pullParams` se define mediante tres pares de nombre-valor, separados por comas e incluidos entre llaves (izquierda y derecha). Cuando se proporcionan parámetros a una llamada de método de objeto de servicio, los nombres se determinan mediante nombres de parámetros para el método de objeto de servicio al que tiene previsto llamar. Al invocar una función de Lambda, `FunctionName`, `InvocationType` y `LogType` son los parámetros que se utilizan para llamar al método `invoke` de un objeto de servicio de Lambda.

Cuando transfiera parámetros a una llamada de método de objeto de servicio, proporcione el objeto JSON a la llamada al método, tal y como se muestra en el siguiente ejemplo de invocación a una función de 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);
  }
});
```

Devolución de datos como JSON

JSON ofrece una forma estándar de transferir datos entre partes de una aplicación que necesitan enviar varios valores al mismo tiempo. Los métodos de clases de cliente en la API suelen devolver JSON en el parámetro `data` que se transfiere a sus funciones de devolución de llamada. Por ejemplo, a continuación se muestra una llamada al método `getBucketCors` de la clase de cliente de 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));
  }
});
```

El valor de `data` es un objeto JSON; en este ejemplo, JSON que describe la configuración CORS actual para un bucket de Amazon S3 especificado.

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

```
        "MaxAgeSeconds": 3000
    }
  ]
}
```

Estrategia de reintentos en la versión 2 de AWS SDK para JavaScript

Numerosos componentes de una red, como los servidores DNS, los conmutadores o los balanceadores de carga, entre otros, pueden generar errores en cualquier punto de la vida de una solicitud determinada. La técnica habitual para abordar estas respuestas de error en un entorno de red consiste en implementar los reintentos en la aplicación cliente. Esta técnica aumenta la fiabilidad de la aplicación y reduce los costos operativos para el desarrollador. AWS SDK implementan una lógica de reintentos automatizados para las solicitudes de AWS.

Comportamiento de reintentos basado en el retroceso exponencial

La versión 2 de AWS SDK para JavaScript implementa la lógica de reintentos mediante el [retroceso exponencial con fluctuación completa](#) para obtener un mejor control del flujo. El retardo exponencial se basa en la idea de utilizar tiempos de espera progresivamente más largos entre reintentos para las respuestas a errores consecutivos. La fluctuación (retardo aleatorio) se utiliza para evitar colisiones sucesivas.

Prueba del retardo de reintentos en la versión 2

Para probar el retardo de reintentos en la versión 2, se ha actualizado el código en [node_modules/aws-sdk/lib/event_listeners.js](#) en `console.log` al valor presente en la variable `delay` de la siguiente manera:

```
// delay < 0 is a signal from customBackoff to skip retries
if (willRetry && delay >= 0) {
  resp.error = null;
  console.log('retry delay: ' + delay);
  setTimeout(done, delay);
} else {
  done();
}
```

Retardos de reintentos con la configuración predeterminada

Puede probar el retardo de cualquier operación en los clientes del SDK de AWS. Llamamos a la operación `listTables` en un cliente de DynamoDB con el siguiente código:

```
import AWS from "aws-sdk";

const region = "us-east-1";
const client = new AWS.DynamoDB({ region });
await client.listTables({}).promise();
```

Para probar los reintentos, simulamos `NetworkingError` mediante la desconexión de Internet del dispositivo que ejecuta el código de prueba. También puede configurar el proxy para que devuelva un error personalizado.

Al ejecutar el código, puede ver que el retardo de reintentos mediante el retroceso exponencial con fluctuación es como sigue:

```
retry delay: 7.39361151766359
retry delay: 9.0672860785882
retry delay: 134.89340825668168
retry delay: 398.53559817403965
retry delay: 523.8076165896343
retry delay: 1323.8789643058465
```

Como el reintento utiliza la fluctuación, obtendrá valores diferentes en la ejecución del código de ejemplo.

Retardos de reintentos con base personalizada

La versión 2 de AWS SDK para JavaScript permite pasar un número base personalizado de milisegundos para utilizarlo en el retroceso exponencial para los reintentos de la operación. Está predeterminado a 100 ms para todos los servicios excepto DynamoDB, donde está predeterminado a 50 ms.

Probamos los reintentos con una base personalizada de 1000 ms de la siguiente manera:

```
...
const client = new AWS.DynamoDB({ region, retryDelayOptions: { base: 1000 } });
...
```

Simulamos `NetworkingError` mediante la desconexión de Internet del dispositivo que ejecuta el código de prueba. Puede ver que los valores para el retardo de reintentos son más altos en comparación con la ejecución anterior, en la que el valor predeterminado era de 50 o 100 ms.

```
retry delay: 356.2841549924913
retry delay: 1183.5216495444615
retry delay: 2266.997988094194
retry delay: 1244.6948354966453
retry delay: 4200.323030066383
```

Como el reintento utiliza la fluctuación, obtendrá valores diferentes en la ejecución del código de ejemplo.

Retardos de reintentos con algoritmo de retroceso personalizado

La versión 2 de AWS SDK para JavaScript también permite pasar una función de retroceso personalizada que acepta un recuento de reintentos y un error y devuelve la cantidad de tiempo que se retrasará en milisegundos. Si el resultado es un valor negativo distinto de cero, no se realizarán más intentos de reintentos.

Probamos la función de retroceso personalizada que utiliza el retroceso lineal con un valor base de 200 ms de la siguiente manera:

```
...
const client = new AWS.DynamoDB({
  region,
  retryDelayOptions: { customBackoff: (count, error) => (count + 1) * 200 },
});
...
```

Simulamos `NetworkingError` mediante la desconexión de Internet del dispositivo que ejecuta el código de prueba. Puede ver que los valores para el retardo de reintentos son múltiplos de 200.

```
retry delay: 200
retry delay: 400
retry delay: 600
retry delay: 800
retry delay: 1000
```

Ejemplos de código SDK para JavaScript

En el tema de esta sección hay varios ejemplos de cómo usar AWS SDK for JavaScript con las API de diversos servicios para realizar tareas habituales.

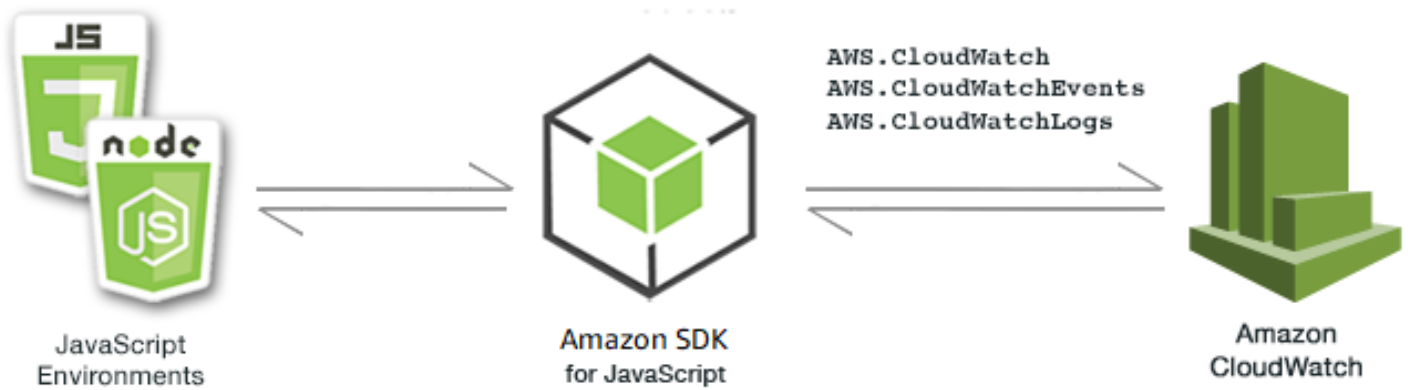
Busque el código fuente de estos ejemplos y otros en la documentación de AWS [repositorio de ejemplos de código en GitHub](#). Para proponer un nuevo ejemplo de código para que el equipo de documentación de AWS considere la posibilidad de crearlo, cree una nueva solicitud. El equipo está buscando crear ejemplos de código que abarquen situaciones y casos de uso más amplios, en comparación con fragmentos de código sencillos que tratan solo llamadas a la API individuales. Para obtener instrucciones, consulte la sección Authoring code en las [pautas de contribución](#).

Temas

- [Ejemplos de Amazon CloudWatch](#)
- [Ejemplos de Amazon DynamoDB](#)
- [Ejemplos de Amazon EC2](#)
- [AWS Elemental MediaConvertEjemplos de](#)
- [Ejemplos de AWS IAM](#)
- [Ejemplo de Amazon Kinesis](#)
- [Ejemplos de Amazon S3](#)
- [Ejemplos de Amazon Simple Email Service](#)
- [Ejemplos de Amazon Simple Notification Service](#)
- [Ejemplos de Amazon SQS](#)

Ejemplos de Amazon CloudWatch

Amazon CloudWatch (CloudWatch) es un servicio web que supervisa sus recursos de Amazon Web Services y las aplicaciones que ejecuta en tiempo real en AWS. Puede utilizar CloudWatch para recopilar y hacer un seguimiento de métricas, que son las variables que puede medir en los recursos y aplicaciones. Las alarmas de CloudWatch envían notificaciones o realizan cambios automáticamente en los recursos que está supervisando basándose en las reglas que defina.



La API de JavaScript para CloudWatch se expone a través de las clases de cliente `AWS.CloudWatch`, `AWS.CloudWatchEvents` y `AWS.CloudWatchLogs`. Para obtener más información acerca de cómo usar las clases de cliente de CloudWatch, consulte [Class: AWS.CloudWatch](#), [Class: AWS.CloudWatchEvents](#) y [Class: AWS.CloudWatchLogs](#) en la referencia de la API.

Temas

- [Creación de alarmas de Amazon CloudWatch](#)
- [Uso de acciones de alarma en Amazon CloudWatch](#)
- [Obtención de métricas de Amazon CloudWatch](#)
- [Envío de eventos a Amazon CloudWatch Events](#)
- [Uso de filtros de suscripción en Registros de Amazon CloudWatch](#)

Creación de alarmas de Amazon CloudWatch



Este ejemplo de código de Node.js muestra:

- Cómo recuperar información básica acerca de sus alarmas de CloudWatch.
- Cómo crear y eliminar una alarma de CloudWatch.

El escenario

Una alarma vigila una única métrica durante el período especificado y realiza una o varias acciones en función del valor de la métrica relativo a un determinado umbral durante una serie de períodos de tiempo.

En este ejemplo, se usan una serie de módulos de Node.js para crear alarmas en CloudWatch. Los módulos de Node.js usan el SDK de JavaScript para crear alarmas mediante los métodos de clase de cliente `AWS.CloudWatch` siguientes:

- [describeAlarms](#)
- [putMetricAlarm](#)
- [deleteAlarms](#)

Para obtener más información sobre la configuración de las alarmas, consulte [Crear alarmas de Amazon CloudWatch](#) en la Guía del usuario de Amazon CloudWatch.

Tareas previas necesarias

Para configurar y ejecutar este ejemplo, primero debe completar estas tareas:

- Instale Node.js. Para obtener más información acerca de la instalación de Node.js consulte el [sitio web de Node.js](#).
- Cree un archivo de configuraciones compartidas con sus credenciales de usuario. Para obtener más información acerca de cómo crear un archivo de credenciales compartidas, consulte [Carga de credenciales en Node.js desde el archivo de credenciales compartidas](#).

Descripción de alarmas

Cree un módulo de Node.js con el nombre de archivo `cw_describealarms.js`. Asegúrese de configurar el SDK tal y como se ha indicado anteriormente. Para obtener acceso a CloudWatch, cree un objeto de servicio de `AWS.CloudWatch`. Cree un objeto JSON para contener los parámetros para recuperar descripciones de alarmas y limitar el número de alarmas a aquellas que tienen el estado `INSUFFICIENT_DATA`. Luego llame al método `describeAlarms` del objeto de servicio `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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos.

```
node cw_describealarms.js
```

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Creación de una alarma para una métrica de CloudWatch

Cree un módulo de Node.js con el nombre de archivo `cw_putmetricalarm.js`. Asegúrese de configurar el SDK tal y como se ha indicado anteriormente. Para obtener acceso a CloudWatch, cree un objeto de servicio de `AWS.CloudWatch`. Cree un objeto JSON para los parámetros necesarios para crear una alarma basada en una métrica; en este caso, el uso de la CPU en una instancia de Amazon EC2. Los parámetros restantes se establecen para que la alarma se desencadene cuando la métrica supere un umbral del 70 por ciento. Luego llame al método `describeAlarms` del objeto de servicio `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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos.

```
node cw_putmetricalarm.js
```

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Eliminación de una alarma

Cree un módulo de Node.js con el nombre de archivo `cw_deletealarms.js`. Asegúrese de configurar el SDK tal y como se ha indicado anteriormente. Para obtener acceso a CloudWatch, cree un objeto de servicio de `AWS.CloudWatch`. Cree un objeto JSON para que contenga los nombres de las alarmas que desea eliminar. Luego llame al método `deleteAlarms` del objeto de servicio `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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos.

```
node cw_deletealarms.js
```

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Uso de acciones de alarma en Amazon CloudWatch



Este ejemplo de código de Node.js muestra:

- Cómo cambiar el estado de sus instancias de Amazon EC2 automáticamente en función de una alarma de CloudWatch.

El escenario

Mediante las acciones de alarma, puede crear alarmas que paran, terminan, reinician o recuperan automáticamente sus instancias de Amazon EC2. Puede utilizar las acciones parar o terminar

cuando ya no necesita que se ejecute una instancia. Puede usar las acciones reiniciar y recuperar para reiniciar automáticamente esas instancias.

En este ejemplo se usan una serie de módulos de Node.js para definir una acción de alarma en CloudWatch que desencadene el reinicio de una instancia de Amazon EC2. Los módulos de Node.js usan el SDK para JavaScript para gestionar instancias de Amazon EC2 mediante los métodos de clase de cliente CloudWatch siguientes:

- [enableAlarmActions](#)
- [disableAlarmActions](#)

Para obtener más información acerca de las acciones de alarma de CloudWatch, consulte [Crear alarmas para parar, terminar, reiniciar o recuperar una instancia](#) en la Guía del usuario de Amazon CloudWatch.

Tareas previas necesarias

Para configurar y ejecutar este ejemplo, primero debe completar estas tareas:

- Instale Node.js. Para obtener más información acerca de la instalación de Node.js consulte el [sitio web de Node.js](#).
- Cree un archivo de configuraciones compartidas con sus credenciales de usuario. Para obtener más información acerca de cómo crear un archivo de credenciales compartidas, consulte [Carga de credenciales en Node.js desde el archivo de credenciales compartidas](#).
- Cree un rol de IAM cuya política conceda permiso para describir, reiniciar, detener o terminar una instancia de Amazon EC2. Para obtener más información sobre cómo crear un rol de IAM, consulte [Creación de un rol para delegar permisos a un servicio de AWS](#) en la Guía del usuario de IAM.

Utilice la siguiente política de funciones al crear la función de IAM.

JSON

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

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

Configure el SDK para JavaScript creando un objeto de configuración global y luego configurando la región para su código. En este ejemplo, la región está establecida en `us-west-2`.

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

Creación y habilitación de acciones en una alarma

Cree un módulo de Node.js con el nombre de archivo `enablealarmactions.js`. Asegúrese de configurar el SDK tal y como se ha indicado anteriormente. Para obtener acceso a CloudWatch, cree un objeto de servicio de `AWS.CloudWatch`.

Cree un objeto JSON para contener los parámetros para crear una alarma y especifique `ActionsEnabled` como `true` y una matriz de ARN para las acciones que la alarma desencadenará. Llame al método `putMetricAlarm` del objeto de servicio `AWS.CloudWatch`, que crea la alarma si no existe o la actualiza si existe.

En la función de devolución de llamada para `putMetricAlarm`, al realizarse correctamente cree un objeto JSON que contenga el nombre de la alarma de CloudWatch. Llame al método `enableAlarmActions` para habilitar la acción de alarma.

```
// 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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos.

```
node cw_enablealarmactions.js
```

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Deshabilitar acciones en una alarma

Cree un módulo de Node.js con el nombre de archivo `cw_disablealarmactions.js`. Asegúrese de configurar el SDK tal y como se ha indicado anteriormente. Para obtener acceso a CloudWatch, cree un objeto de servicio de `AWS.CloudWatch`. Cree un objeto JSON que contenga el nombre de la alarma de CloudWatch. Llame al método `disableAlarmActions` para deshabilitar las acciones para esta alarma.

```
// 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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos.

```
node cw_disablealarmactions.js
```

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Obtención de métricas de Amazon CloudWatch



Este ejemplo de código de Node.js muestra:

- Cómo recuperar una lista de métricas de CloudWatch publicadas.
- Cómo publicar puntos de datos en métricas de CloudWatch.

El escenario

Las métricas son los datos sobre el desempeño de los sistemas. Puede habilitar la supervisión detallada de algunos recursos, como las instancias de Amazon EC2, o de sus propias métricas de aplicación.

En este ejemplo se van a utilizar una serie de módulos de Node.js para obtener métricas de CloudWatch y para enviar eventos a Amazon CloudWatch Events. Los módulos de Node.js usan el SDK para JavaScript para obtener métricas de CloudWatch mediante los métodos de la clase de cliente CloudWatch siguiente:

- [listMetrics](#)
- [putMetricData](#)

Para obtener más información sobre las métricas de CloudWatch, consulte [Uso de métricas de Amazon CloudWatch](#) en la Guía del usuario de Amazon CloudWatch.

Tareas previas necesarias

Para configurar y ejecutar este ejemplo, primero debe completar estas tareas:

- Instale Node.js. Para obtener más información acerca de la instalación de Node.js consulte el [sitio web de Node.js](#).
- Cree un archivo de configuraciones compartidas con sus credenciales de usuario. Para obtener más información acerca de cómo crear un archivo de credenciales compartidas, consulte [Carga de credenciales en Node.js desde el archivo de credenciales compartidas](#).

Mostrar métricas

Cree un módulo de Node.js con el nombre de archivo `cw_listmetrics.js`. Asegúrese de configurar el SDK tal y como se ha indicado anteriormente. Para obtener acceso a CloudWatch, cree un objeto de servicio de `AWS.CloudWatch`. Cree un objeto JSON que contenga los parámetros necesarios para generar métricas en el espacio de nombres `AWS/Logs`. Llame al método `listMetrics` para obtener una lista de la 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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos.

```
node cw_listmetrics.js
```

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Envío de métricas personalizadas

Cree un módulo de Node.js con el nombre de archivo `cw_putmetricdata.js`. Asegúrese de configurar el SDK tal y como se ha indicado anteriormente. Para obtener acceso a CloudWatch, cree un objeto de servicio de `AWS.CloudWatch`. Cree un objeto JSON que contenga los parámetros necesarios para enviar un punto de datos para la métrica personalizada `PAGES_VISITED`. Llame al 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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos.

```
node cw_putmetricdata.js
```

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Envío de eventos a Amazon CloudWatch Events



Este ejemplo de código de Node.js muestra:

- Cómo crear y actualizar una regla utilizada para desencadenar un evento.
- Cómo definir uno o varios destinos para responder a un evento.
- Cómo enviar eventos que se corresponden con los destinos para la gestión.

El escenario

Amazon CloudWatch Events proporciona un flujo de eventos de sistema casi en tiempo real que describen cambios en los recursos de Amazon Web Services a cualquiera de los distintos destinos. Mediante reglas sencillas, puede asignar los eventos y dirigirlos a uno o más flujos o funciones de destino.

En este ejemplo se van a utilizar una serie de módulos de Node.js para enviar eventos a CloudWatch Events. Los módulos de Node.js usan el SDK para JavaScript para gestionar instancias mediante los métodos de clase de cliente `CloudWatchEvents` siguientes:

- [putRule](#)
- [putTargets](#)
- [putEvents](#)

Para obtener más información acerca del uso de Eventos de CloudWatch, consulte [Agregar eventos con PutEvents](#) la Guía del usuario de Eventos de Amazon CloudWatch.

Tareas previas necesarias

Para configurar y ejecutar este ejemplo, primero debe completar estas tareas:

- Instale Node.js. Para obtener más información acerca de la instalación de Node.js consulte el [sitio web de Node.js](#).
- Cree un archivo de configuraciones compartidas con sus credenciales de usuario. Para obtener más información acerca de cómo crear un archivo de credenciales compartidas, consulte [Carga de credenciales en Node.js desde el archivo de credenciales compartidas](#).
- Cree una función de Lambda usando el esquema hello-world, que actuará como destino de los eventos. Para obtener información sobre cómo hacerlo, consulte [Paso 1: Crear una función de AWS Lambda](#) en la Guía del usuario de Amazon CloudWatch Events.
- Cree un rol de IAM cuya política conceda permiso a CloudWatch Events y que incluya `events.amazonaws.com` como entidad de confianza. Para obtener más información sobre cómo crear un rol de IAM, consulte [Creación de un rol para delegar permisos a un servicio de AWS](#) en la Guía del usuario de IAM.

Utilice la siguiente política de funciones al crear la función de IAM.

JSON

```
{
  "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"
    }
  ]
}
```

Utilice la siguiente relación de confianza al crear la función de IAM.

JSON

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

Creación de una regla programada

Cree un módulo de Node.js con el nombre de archivo `cwe_putrule.js`. Asegúrese de configurar el SDK tal y como se ha indicado anteriormente. Para obtener acceso a CloudWatch Events, cree un objeto de servicio de `AWS.CloudWatchEvents`. Cree un objeto JSON que contenga los parámetros necesarios para especificar la nueva regla programada, que incluye lo siguiente:

- Un nombre para la regla
- El ARN del rol de IAM que ha creado anteriormente
- Una expresión para programar la activación de la regla cada cinco minutos

Llame al método `putRule` para crear la regla. La devolución de llamada devuelve al ARN o la regla nueva o actualizada.

```
// 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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos.

```
node cwe_putrule.js
```

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Agregar un destino de la función de AWS Lambda

Cree un módulo de Node.js con el nombre de archivo `cwe_puttargets.js`. Asegúrese de configurar el SDK tal y como se ha indicado anteriormente. Para obtener acceso a CloudWatch Events, cree un objeto de servicio de `AWS.CloudWatchEvents`. Cree un objeto JSON que contenga los parámetros necesarios para especificar la regla a la que desea asociar el destino, incluido el ARN de la función de Lambda que ha creado. Llame al método `putTargets` del objeto de servicio de `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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos.

```
node cwe_puttargets.js
```

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Envío de eventos

Cree un módulo de Node.js con el nombre de archivo `cwe_putevents.js`. Asegúrese de configurar el SDK tal y como se ha indicado anteriormente. Para obtener acceso a CloudWatch Events, cree un objeto de servicio de `AWS.CloudWatchEvents`. Cree un objeto JSON que contenga los parámetros necesarios para enviar eventos. Para cada evento, incluya el origen del evento, los ARN de cualquier recurso afectado por el evento y detalles del evento. Llame al método `putEvents` del objeto de servicio de `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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos.

```
node cwe_putevents.js
```

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Uso de filtros de suscripción en Registros de Amazon CloudWatch



Este ejemplo de código de Node.js muestra:

- Cómo crear y eliminar filtros para eventos de registro en Registros de CloudWatch.

El escenario

Las suscripciones proporcionan acceso a la fuente en tiempo real de eventos de registro de Registros de CloudWatch y envían esa fuente a otros servicios como, por ejemplo, un flujo de Amazon Kinesis o AWS Lambda, para el procesamiento, el análisis o la carga personalizados en otros sistemas. Un filtro de suscripción define el patrón que se utilizará para filtrar los eventos de registro que se envían a su recurso de AWS.

En este ejemplo se van a utilizar una serie de módulos de Node.js para enumerar, crear y administrar un filtro de suscripción en Registros de CloudWatch. El destino de los eventos de registro es una

función de Lambda. Los módulos de Node.js usan el SDK para JavaScript para administrar filtros de suscripción mediante los métodos de clase de cliente de CloudWatchLogs siguientes:

- [putSubscriptionFilters](#)
- [describeSubscriptionFilters](#)
- [deleteSubscriptionFilter](#)

Para obtener más información, consulte [Procesamiento en tiempo real de datos de registro con suscripciones](#) en la Guía del usuario de Registros de Amazon CloudWatch.

Tareas previas necesarias

Para configurar y ejecutar este ejemplo, primero debe completar estas tareas:

- Instale Node.js. Para obtener más información acerca de la instalación de Node.js consulte el [sitio web de Node.js](#).
- Cree un archivo de configuraciones compartidas con sus credenciales de usuario. Para obtener más información acerca de cómo crear un archivo de credenciales compartidas, consulte [Carga de credenciales en Node.js desde el archivo de credenciales compartidas](#).
- Cree una función de Lambda como destino para los eventos de registro. Tendrá que utilizar el ARN de esta función. Para obtener más información acerca de cómo configurar una función de Lambda, consulte [Filtros de suscripción con AWS Lambda](#) en la Guía del usuario de Registros de Amazon CloudWatch.
- Cree un rol de IAM cuya política conceda permiso para invocar la función de Lambda que ha creado y conceda acceso completo a Registros de CloudWatch o aplique la siguiente política a la función de ejecución que cree para la función de Lambda. Para obtener más información sobre cómo crear un rol de IAM, consulte [Creación de un rol para delegar permisos a un servicio de AWS](#) en la Guía del usuario de IAM.

Utilice la siguiente política de funciones al crear la función de IAM.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
```

```

    {
      "Effect": "Allow",
      "Action": [
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:PutLogEvents"
      ],
      "Resource": "arn:aws:logs:*:*:*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "lambda:InvokeFunction"
      ],
      "Resource": [
        "*"
      ]
    }
  ]
}

```

Descripción de los filtros de suscripción existentes

Cree un módulo de Node.js con el nombre de archivo `cwl_describesubscriptionfilters.js`. Asegúrese de configurar el SDK tal y como se ha indicado anteriormente. Para obtener acceso a Registros de CloudWatch, cree un objeto de servicio de `AWS.CloudWatchLogs`. Cree un objeto JSON que contenga los parámetros necesarios para describir sus filtros existentes, como el nombre del grupo de registros y el número máximo de filtros que desea describir. Llame al 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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos.

```
node cw1_describesubscriptionfilters.js
```

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Creación de un filtro de suscripción

Cree un módulo de Node.js con el nombre de archivo `cw1_putsubscriptionfilter.js`. Asegúrese de configurar el SDK tal y como se ha indicado anteriormente. Para obtener acceso a Registros de CloudWatch, cree un objeto de servicio de `AWS.CloudWatchLogs`. Cree un objeto JSON que contenga los parámetros necesarios para crear un filtro, como el ARN de la función de Lambda de destino, el nombre del filtro, el patrón de cadena para el filtrado y el nombre del grupo de registros. Llame al 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 cw1 = new AWS.CloudWatchLogs({ apiVersion: "2014-03-28" });

var params = {
  destinationArn: "LAMBDA_FUNCTION_ARN",
  filterName: "FILTER_NAME",
  filterPattern: "ERROR",
  logGroupName: "LOG_GROUP",
};

cw1.putSubscriptionFilter(params, function (err, data) {
```

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

Para ejecutar el ejemplo, escriba lo siguiente en la línea de comandos.

```
node cwl_putsubscriptionfilter.js
```

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Eliminación de un filtro de suscripción

Cree un módulo de Node.js con el nombre de archivo `cwl_deletesubscriptionfilters.js`. Asegúrese de configurar el SDK tal y como se ha indicado anteriormente. Para obtener acceso a Registros de CloudWatch, cree un objeto de servicio de `AWS.CloudWatchLogs`. Cree un objeto JSON que contenga los parámetros necesarios para eliminar un filtro, como los nombres del filtro y el grupo de registros. Llame al 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);
  }
});
```

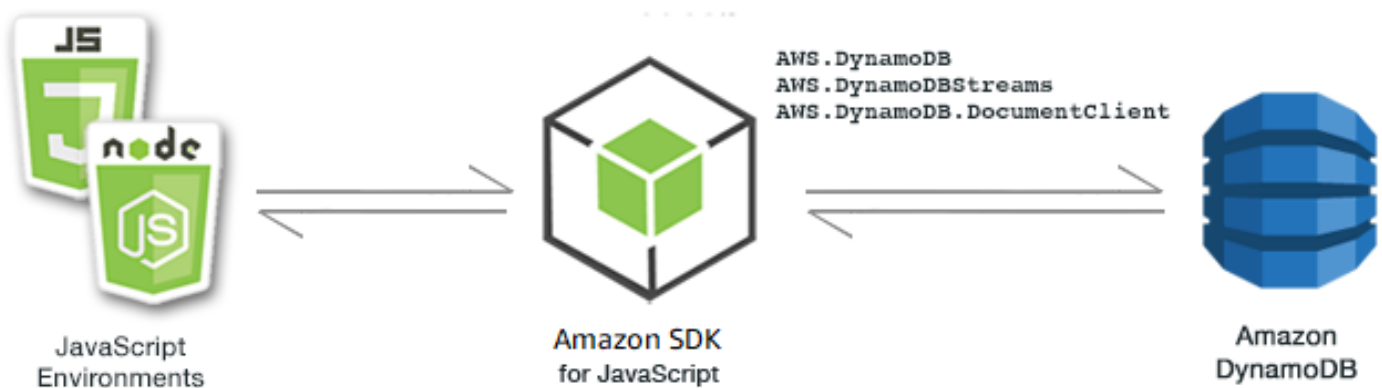
Para ejecutar el ejemplo, escriba lo siguiente en la línea de comandos.

```
node cwl_deletesubscriptionfilter.js
```

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Ejemplos de Amazon DynamoDB

Amazon DynamoDB es una base de datos de la nube NoSQL completamente administrada compatible con modelos de almacenamiento de documentos y clave-valor. Puede crear tablas sin esquema para los datos sin necesidad de aprovisionar ni de mantener servidores de bases de datos dedicados.



La API de JavaScript para DynamoDB se expone a través de las clases de cliente `AWS.DynamoDB`, `AWS.DynamoDBStreams` y `AWS.DynamoDB.DocumentClient`. Para obtener más información acerca de cómo usar la clase de cliente de DynamoDB, consulte [Class: AWS.DynamoDB](#), [Class: AWS.DynamoDBStreams](#) y [Class: AWS.DynamoDB.DocumentClient](#) en la referencia de la API.

Temas

- [Creación y uso de tablas en DynamoDB](#)
- [Lectura y escritura de un único elemento en DynamoDB](#)
- [Lectura y escritura de elementos en lotes en DynamoDB](#)
- [Consulta y examen de una tabla de DynamoDB](#)
- [Uso del cliente de documentos de DynamoDB](#)

Creación y uso de tablas en DynamoDB



Este ejemplo de código de Node.js muestra:

- Cómo crear y administrar tablas utilizadas para almacenar y recuperar datos desde DynamoDB.

El escenario

Al igual que otros sistemas de base de datos, DynamoDB almacena datos en tablas. Una tabla de DynamoDB es una colección de datos que se organizan en elementos similares a las filas. Para almacenar u obtener acceso a datos de DynamoDB, debe crear y trabajar con tablas.

En este ejemplo, va a utilizar una serie de módulos Node.js para realizar operaciones básicas con una tabla de DynamoDB. El código utiliza el SDK de JavaScript para crear y trabajar con tablas mediante los métodos siguientes de la clase de cliente de AWS .DynamoDB:

- [createTable](#)
- [listTables](#)
- [describeTable](#)
- [deleteTable](#)

Tareas previas necesarias

Para configurar y ejecutar este ejemplo, primero debe completar las tareas siguientes:

- Instale Node.js. Para obtener más información, consulte el sitio web de [Node.js](#).
- Cree un archivo de configuraciones compartidas con sus credenciales de usuario. Para obtener más información acerca de cómo crear un archivo de credenciales compartidas, consulte [Carga de credenciales en Node.js desde el archivo de credenciales compartidas](#).

Creación de una tabla

Cree un módulo de Node.js con el nombre de archivo `ddb_createtable.js`. Asegúrese de configurar el SDK tal y como se ha indicado anteriormente. Para acceder a DynamoDB, cree un objeto de servicio de `AWS.DynamoDB`. Cree un objeto JSON que contenga los parámetros necesarios para crear una tabla, que en este ejemplo incluye el nombre y el tipo de datos para cada atributo, el esquema de claves, el nombre de la tabla y las unidades de rendimiento que deben provisionarse. Llame al método `createTable` del objeto de servicio de 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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos.

```
node ddb_createtable.js
```

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Creación de una lista de tablas

Cree un módulo de Node.js con el nombre de archivo `ddb_listtables.js`. Asegúrese de configurar el SDK tal y como se ha indicado anteriormente. Para acceder a DynamoDB, cree un objeto de servicio de `AWS.DynamoDB`. Cree un objeto JSON que contenga los parámetros necesarios para obtener una lista de las tablas, que en este ejemplo limita el número de tablas enumeradas a 10. Llame al método `listTables` del objeto de servicio de 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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos.

```
node ddb_listtables.js
```

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Descripción de una tabla

Cree un módulo de Node.js con el nombre de archivo `ddb_describetable.js`. Asegúrese de configurar el SDK tal y como se ha indicado anteriormente. Para acceder a DynamoDB, cree un objeto de servicio de `AWS.DynamoDB`. Cree un objeto JSON que contenga los parámetros necesarios para describir una tabla, que en este ejemplo incluye el nombre de la tabla que se proporciona como parámetro de la línea de comandos. Llame al método `describeTable` del objeto de servicio de 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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos.

```
node ddb_describetable.js TABLE_NAME
```

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Eliminación de una tabla

Cree un módulo de Node.js con el nombre de archivo `ddb_deletetable.js`. Asegúrese de configurar el SDK tal y como se ha indicado anteriormente. Para acceder a DynamoDB, cree un objeto de servicio de `AWS.DynamoDB`. Cree un objeto JSON que contenga los parámetros necesarios para eliminar una tabla. En este ejemplo, el nombre de la tabla se proporciona como parámetro de la línea de comandos. Llame al método `deleteTable` del objeto de servicio de 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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos.

```
node ddb_deletetable.js TABLE_NAME
```

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Lectura y escritura de un único elemento en DynamoDB



Este ejemplo de código de Node.js muestra:

- Cómo añadir un elemento a una tabla de DynamoDB.
- Cómo recuperar un elemento de una tabla de DynamoDB.
- Cómo eliminar un elemento de una tabla de DynamoDB

El escenario

En este ejemplo, va a utilizar una serie de módulos de Node.js para leer y escribir un elemento en una tabla de DynamoDB mediante los métodos siguientes de la clase de cliente de AWS .DynamoDB:

- [putItem](#)
- [getItem](#)
- [deleteItem](#)

Tareas previas necesarias

Para configurar y ejecutar este ejemplo, primero debe completar las tareas siguientes:

- Instale Node.js. Para obtener más información, consulte el sitio web de [Node.js](#).
- Cree un archivo de configuraciones compartidas con sus credenciales de usuario. Para obtener más información acerca de cómo crear un archivo de credenciales compartidas, consulte [Carga de credenciales en Node.js desde el archivo de credenciales compartidas](#).
- Cree una tabla de DynamoDB a cuyos elementos pueda obtener acceso. Para obtener más información acerca de cómo crear una tabla de DynamoDB, consulte [Creación y uso de tablas en DynamoDB](#).

Escritura de un elemento

Cree un módulo de Node.js con el nombre de archivo `ddb_putitem.js`. Asegúrese de configurar el SDK tal y como se ha indicado anteriormente. Para acceder a DynamoDB, cree un objeto de servicio de `AWS.DynamoDB`. Cree un objeto JSON que contenga los parámetros necesarios para añadir un elemento, que en este ejemplo incluye el nombre de la tabla y un mapa que define los atributos que deben establecerse y los valores para cada atributo. Llame al método `putItem` del objeto de servicio de 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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos.

```
node ddb_putitem.js
```

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Obtención de un elemento

Cree un módulo de Node.js con el nombre de archivo `ddb_getitem.js`. Asegúrese de configurar el SDK tal y como se ha indicado anteriormente. Para acceder a DynamoDB, cree un objeto de servicio de `AWS.DynamoDB`. Para identificar el elemento que se desea obtener, debe proporcionar el valor de la clave principal de ese elemento en la tabla. De forma predeterminada, el método `getItem` devuelve todos los valores de atributos definidos para el elemento. Para obtener únicamente un subconjunto de todos los posibles valores de atributo, especifique una expresión de proyección.

Cree un objeto JSON que contenga los parámetros necesarios para obtener un elemento, que en este ejemplo incluye el nombre de la tabla, el nombre y el valor de la clave para el elemento que recibe y una expresión de proyección que identifica el atributo del elemento que desea recuperar. Llame al método `getItem` del objeto de servicio de 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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos.

```
node ddb_getitem.js
```

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Eliminación de un elemento

Cree un módulo de Node.js con el nombre de archivo `ddb_deleteitem.js`. Asegúrese de configurar el SDK tal y como se ha indicado anteriormente. Para acceder a DynamoDB, cree un objeto de servicio de `AWS.DynamoDB`. Cree un objeto JSON que contenga los parámetros necesarios para eliminar un elemento, que, en este ejemplo, contiene el nombre de la tabla y tanto el nombre como el valor de la clave del elemento que va a eliminar. Llame al método `deleteItem` del objeto de servicio de 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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos.

```
node ddb_deleteitem.js
```

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Lectura y escritura de elementos en lotes en DynamoDB



Este ejemplo de código de Node.js muestra:

- Cómo leer y escribir lotes de elementos en una tabla de DynamoDB.

El escenario

En este ejemplo, va a utilizar una serie de módulos de Node.js para poner un lote de elementos de una tabla de DynamoDB, así como para leer un lote de elementos. El código utiliza el SDK para JavaScript para realizar operaciones de lectura y escritura por lotes mediante los métodos siguientes de clase de cliente de DynamoDB:

- [batchGetItem](#)
- [batchWriteItem](#)

Tareas previas necesarias

Para configurar y ejecutar este ejemplo, primero debe completar las tareas siguientes:

- Instale Node.js. Para obtener más información, consulte el sitio web de [Node.js](#).
- Cree un archivo de configuraciones compartidas con sus credenciales de usuario. Para obtener más información acerca de cómo crear un archivo de credenciales compartidas, consulte [Carga de credenciales en Node.js desde el archivo de credenciales compartidas](#).
- Cree una tabla de DynamoDB a cuyos elementos pueda obtener acceso. Para obtener más información acerca de cómo crear una tabla de DynamoDB, consulte [Creación y uso de tablas en DynamoDB](#).

Lectura de elementos en lotes

Cree un módulo de Node.js con el nombre de archivo `ddb_batchgetitem.js`. Asegúrese de configurar el SDK tal y como se ha indicado anteriormente. Para acceder a DynamoDB, cree un objeto de servicio de `AWS.DynamoDB`. Cree un objeto JSON que contenga los parámetros necesarios para obtener un lote de elementos, que en este ejemplo incluye el nombre de una o varias tablas donde se leerá, los valores de claves para leer en cada tabla y la expresión de proyección que especifica los atributos que se devolverán. Llame al método `batchGetItem` del objeto de servicio de 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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos.

```
node ddb_batchgetitem.js
```

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Escritura de elementos en lotes

Cree un módulo de Node.js con el nombre de archivo `ddb_batchwriteitem.js`. Asegúrese de configurar el SDK tal y como se ha indicado anteriormente. Para acceder a DynamoDB, cree un objeto de servicio de `AWS.DynamoDB`. Cree un objeto JSON que contenga los parámetros necesarios para obtener un lote de elementos, que en este ejemplo incluye la tabla en la que desea escribir elementos, las claves que desea escribir para cada elemento y los atributos junto con sus valores. Llame al método `batchWriteItem` del objeto de servicio de 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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos.

```
node ddb_batchwriteitem.js
```

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Consulta y examen de una tabla de DynamoDB



Este ejemplo de código de Node.js muestra:

- Cómo consultar y examinar una tabla de DynamoDB para buscar elementos.

El escenario

Con las consultas se encuentran elementos de una tabla o un índice secundario usando únicamente valores de atributos de la clave principal. Debe proporcionar un nombre de clave de partición y un valor que deben buscarse. También puede proporcionar un nombre y un valor de clave de ordenación y utilizar un operador de comparación para ajustar los resultados de búsqueda. El examen busca elementos comprobando cada uno de ellos en la tabla especificada.

En este ejemplo, va a utilizar una serie de módulos Node.js para identificar uno o varios elementos que desea recuperar de una tabla de DynamoDB. El código utiliza el SDK para JavaScript para consultar y analizar tablas mediante los métodos siguientes de la clase de cliente de DynamoDB:

- [consulta](#)
- [scan](#)

Tareas previas necesarias

Para configurar y ejecutar este ejemplo, primero debe completar las tareas siguientes:

- Instale Node.js. Para obtener más información, consulte el sitio web de [Node.js](#).
- Cree un archivo de configuraciones compartidas con sus credenciales de usuario. Para obtener más información acerca de cómo crear un archivo de credenciales compartidas, consulte [Carga de credenciales en Node.js desde el archivo de credenciales compartidas](#).
- Cree una tabla de DynamoDB a cuyos elementos pueda obtener acceso. Para obtener más información acerca de cómo crear una tabla de DynamoDB, consulte [Creación y uso de tablas en DynamoDB](#).

Consulta de una tabla

En este ejemplo, se consulta una tabla que contiene información de episodios sobre una serie de vídeos y se devuelven los títulos y subtítulos de episodios de la segunda temporada posteriores al episodio 9 que contienen una frase especificada en su subtítulo.

Cree un módulo de Node.js con el nombre de archivo `ddb_query.js`. Asegúrese de configurar el SDK tal y como se ha indicado anteriormente. Para acceder a DynamoDB, cree un objeto de servicio de `AWS.DynamoDB`. Cree un objeto JSON que contenga los parámetros necesarios para consultar la tabla, que en este ejemplo incluye el nombre de la tabla, los `ExpressionAttributeValues` necesarios para la consulta, una `KeyConditionExpression` que utilice dichos valores para definir qué elementos devuelve la consulta, así como los nombres de los valores de atributos que deben devolverse para cada elemento. Llame al método `query` del objeto de servicio de 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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos.

```
node ddb_query.js
```

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Examen de una tabla

Cree un módulo de Node.js con el nombre de archivo `ddb_scan.js`. Asegúrese de configurar el SDK tal y como se ha indicado anteriormente. Para acceder a DynamoDB, cree un objeto de servicio de `AWS.DynamoDB`. Cree un objeto JSON que contenga los parámetros necesarios para examinar la tabla en búsqueda de elementos, que en este ejemplo incluyen el nombre de la tabla, la lista de los valores de atributos que deben devolverse por cada elemento coincidente y una expresión para filtrar el conjunto de resultados para encontrar elementos que contengan una frase especificada. Llame al método `scan` del objeto de servicio de 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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos.

```
node ddb_scan.js
```

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Uso del cliente de documentos de DynamoDB



Este ejemplo de código de Node.js muestra:

- Cómo obtener acceso a una tabla de DynamoDB mediante el cliente de documentos.

El escenario

El cliente de documentos de DynamoDB simplifica el trabajo con los elementos, ya que abstrae la noción de valores de atributos. Esta abstracción comenta los tipos de JavaScript nativos suministrados como parámetros de entrada, así como convierte datos de respuesta comentados en tipos de JavaScript nativos.

Para obtener más información sobre la clase del cliente de documentos de DynamoDB, consulte [AWS.DynamoDB.DocumentClient](#) en la referencia de la API. Para obtener más información sobre la programación con Amazon DynamoDB, consulte [Programación con DynamoDB](#) en la Guía para desarrolladores de Amazon DynamoDB.

En este ejemplo, va a utilizar una serie de módulos de Node.js para realizar operaciones básicas en una tabla de DynamoDB mediante el cliente de documentos. El código utiliza el SDK para JavaScript para consultar y analizar tablas mediante los métodos siguientes de la clase de cliente de documentos de DynamoDB:

- [get](#)
- [put](#)
- [update](#)
- [consulta](#)
- [delete](#)

Tareas previas necesarias

Para configurar y ejecutar este ejemplo, primero debe completar las tareas siguientes:

- Instale Node.js. Para obtener más información, consulte el sitio web de [Node.js](#).
- Cree un archivo de configuraciones compartidas con sus credenciales de usuario. Para obtener más información acerca de cómo crear un archivo de credenciales compartidas, consulte [Carga de credenciales en Node.js desde el archivo de credenciales compartidas](#).
- Cree una tabla de DynamoDB a cuyos elementos pueda obtener acceso. Para obtener más información acerca de cómo crear una tabla de DynamoDB utilizando el SDK para JavaScript, consulte [Creación y uso de tablas en DynamoDB](#). También puede utilizar la [consola de DynamoDB](#) para crear una tabla.

Obtención de un elemento de una tabla

Cree un módulo de Node.js con el nombre de archivo `ddbdoc_get.js`. Asegúrese de configurar el SDK tal y como se ha indicado anteriormente. Para acceder a DynamoDB, cree un objeto de `AWS.DynamoDB.DocumentClient`. Cree un objeto JSON que contenga los parámetros necesarios para obtener un elemento de la tabla, que en este ejemplo incluye el nombre de la tabla, el nombre de la clave hash en dicha tabla y el valor de la clave hash para el elemento que desea obtener. Llame al método `get` del cliente de documentos de 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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos.

```
node ddbdoc_get.js
```

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Colocación de un elemento en una tabla

Cree un módulo de Node.js con el nombre de archivo `ddbdoc_put.js`. Asegúrese de configurar el SDK tal y como se ha indicado anteriormente. Para acceder a DynamoDB, cree un objeto de `AWS.DynamoDB.DocumentClient`. Cree un objeto JSON que contenga los parámetros necesarios para escribir un elemento en la tabla, que en este ejemplo incluye el nombre de la tabla y una descripción del elemento para añadir o actualizar que incluye el clave hash y el valor, así como los nombres y los valores de atributos que se van a establecer en el elemento. Llame al método `put` del cliente de documentos de 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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos.

```
node ddbdoc_put.js
```

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Actualización de un elemento en una tabla

Cree un módulo de Node.js con el nombre de archivo `ddbdoc_update.js`. Asegúrese de configurar el SDK tal y como se ha indicado anteriormente. Para acceder a DynamoDB, cree un objeto de `AWS.DynamoDB.DocumentClient`. Cree un objeto JSON que contenga los parámetros necesarios para escribir un elemento en la tabla, que en este ejemplo incluye el nombre de la tabla, la clave del elemento que se va a actualizar, un conjunto de `UpdateExpressions` que definen los atributos del elemento que se va a actualizar con tokens a los que asigna valores en los parámetros `ExpressionAttributeValue`. Llame al método `update` del cliente de documentos de 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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos.

```
node ddbdoc_update.js
```

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Consulta de una tabla

En este ejemplo, se consulta una tabla que contiene información de episodios sobre una serie de vídeos y se devuelven los títulos y subtítulos de episodios de la segunda temporada posteriores al episodio 9 que contienen una frase especificada en su subtítulo.

Cree un módulo de Node.js con el nombre de archivo `ddbdoc_query.js`. Asegúrese de configurar el SDK tal y como se ha indicado anteriormente. Para acceder a DynamoDB, cree un objeto de `AWS.DynamoDB.DocumentClient`. Cree un objeto JSON que contenga los parámetros necesarios para consultar la tabla, que en este ejemplo incluye el nombre de la tabla, los `ExpressionAttributeValues` necesarios para la consulta y una `KeyConditionExpression` que utilice dichos valores para definir qué elementos devuelve la consulta. Llame al método `query` del cliente de documentos de 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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos.

```
node ddbdoc_query.js
```

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Eliminación de un elemento de una tabla

Cree un módulo de Node.js con el nombre de archivo `ddbdoc_delete.js`. Asegúrese de configurar el SDK tal y como se ha indicado anteriormente. Para acceder a DynamoDB, cree un objeto de `AWS.DynamoDB.DocumentClient`. Cree un objeto JSON que contenga los parámetros necesarios para eliminar un elemento de la tabla, que en este ejemplo incluye el nombre de la tabla, así como el nombre y el valor de la clave hash del elemento que desea eliminar. Llame al método `delete` del cliente de documentos de 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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos.

```
node ddbdoc_delete.js
```

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Ejemplos de Amazon EC2

Amazon Elastic Compute Cloud (Amazon EC2) es un servicio web que proporciona alojamiento de servidores virtuales en la nube. Se ha diseñado para facilitar a los desarrolladores la informática en la nube en la Web, mediante una capacidad de cómputo variable.



La API de JavaScript para Amazon EC2 se expone a través de la clase de cliente `AWS . EC2`. Para obtener más información acerca de cómo usar la clase de cliente de Amazon EC2, consulte [Class: AWS . EC2](#) en la referencia de la API.

Temas

- [Creación de una instancia de Amazon EC2](#)
- [Administración de instancias de Amazon EC2](#)
- [Uso de pares de claves de Amazon EC2](#)

- [Uso de las regiones y las zonas de disponibilidad de Amazon EC2](#)
- [Trabajar con grupos de seguridad en Amazon EC2](#)
- [Uso de direcciones IP elásticas en Amazon EC2](#)

Creación de una instancia de Amazon EC2



Este ejemplo de código de Node.js muestra:

- Cómo crear una instancia de Amazon EC2 a partir de una imagen de máquina de Amazon (AMI) pública.
- Cómo crear y asignar etiquetas a la nueva instancia de Amazon EC2.

Acerca del ejemplo

En este ejemplo va a utilizar un módulo de Node.js para crear una instancia de Amazon EC2 y asignarle un par de claves y etiquetas. El código utiliza el SDK para JavaScript para crear y etiquetar una instancia mediante estos métodos de la clase de cliente de Amazon EC2:

- [runInstances](#)
- [createTags](#)

Tareas previas necesarias

Para configurar y ejecutar este ejemplo, primero debe completar las tareas siguientes.

- Instale Node.js. Para obtener más información, consulte el sitio web de [Node.js](#).
- Cree un archivo de configuraciones compartidas con sus credenciales de usuario. Para obtener más información acerca de cómo crear un archivo de credenciales compartidas, consulte [Carga de credenciales en Node.js desde el archivo de credenciales compartidas](#).
- Cree un par de claves. Para obtener más información, consulte [Uso de pares de claves de Amazon EC2](#). En este ejemplo utiliza el nombre del par de claves.

Creación y etiquetado de una instancia

Cree un módulo de Node.js con el nombre de archivo `ec2_createinstances.js`. Asegúrese de configurar el SDK tal y como se ha indicado anteriormente.

Cree un objeto para transferir los parámetros para el método `runInstances` de la clase de cliente `AWS.EC2`, como el nombre del par de claves que se va a asignar y el ID de la AMI que se va a ejecutar. Para llamar al método `runInstances`, cree una promesa para invocar un objeto de servicio de Amazon EC2 transfiriendo los parámetros. Luego gestione la respuesta en la devolución de llamada de la promesa.

A continuación, el código añade una etiqueta `Name` a una nueva instancia, que la consola de Amazon EC2 reconoce y muestra en el campo Nombre de la lista de instancias. Puede añadir hasta 50 etiquetas a una instancia, que se pueden añadir todas en una única llamada al 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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos.

```
node ec2_createinstances.js
```

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Administración de instancias de Amazon EC2



Este ejemplo de código de Node.js muestra:

- Cómo recuperar información básica acerca de sus instancias de Amazon EC2.
- Cómo iniciar y detener la supervisión detallada de una instancia de Amazon EC2.
- Cómo iniciar y detener una instancia de Amazon EC2.
- Cómo reiniciar una instancia de Amazon EC2.

El escenario

En este ejemplo, va a utilizar una serie de módulos de Node.js para realizar varias operaciones de administración de instancias básicas. Los módulos de Node.js usan el SDK para JavaScript para gestionar instancias usando los métodos de clase de cliente de Amazon EC2 siguientes:

- [describeInstances](#)
- [monitorInstances](#)
- [unmonitorInstances](#)
- [startInstances](#)
- [stopInstances](#)
- [rebootInstances](#)

Para obtener más información acerca del ciclo de vida de las instancias de Amazon EC2, consulte [Ciclo de vida de las instancias](#) en la Guía del usuario de Amazon EC2.

Tareas previas necesarias

Para configurar y ejecutar este ejemplo, primero debe completar las tareas siguientes:

- Instale Node.js. Para obtener más información acerca de la instalación de Node.js consulte el [sitio web de Node.js](#).
- Cree un archivo de configuraciones compartidas con sus credenciales de usuario. Para obtener más información acerca de cómo crear un archivo de credenciales compartidas, consulte [Carga de credenciales en Node.js desde el archivo de credenciales compartidas](#).
- Crear una instancia de Amazon EC2. Para obtener más información sobre la creación de instancias de Amazon EC2, consulte [Instancias de Amazon EC2](#) en la Guía del usuario de Amazon EC2 o [Instancias de Amazon EC2](#) en la Guía del usuario de Amazon EC2.

Descripción de sus instancias

Cree un módulo de Node.js con el nombre de archivo `ec2_describeinstances.js`. Asegúrese de configurar el SDK tal y como se ha indicado anteriormente. Para acceder a Amazon EC2, cree un objeto de servicio de `AWS.EC2`. Llame al método `describeInstances` del objeto de servicio de Amazon EC2 para recuperar una descripción detallada de sus instancias.

```
// 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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos.

```
node ec2_describeinstances.js
```

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Administración de la monitorización de instancias

Cree un módulo de Node.js con el nombre de archivo `ec2_monitorinstances.js`. Asegúrese de configurar el SDK tal y como se ha indicado anteriormente. Para acceder a Amazon EC2, cree un objeto de servicio de `AWS.EC2`. Añada el ID de las instancias cuya monitorización desea controlar.

Según el valor de un argumento de línea de comando (ON u OFF), llame al método `monitorInstances` del objeto de servicio de Amazon EC2 para empezar la supervisión detallada

de las instancias especificadas o llame al método `unmonitorInstances`. Utilice el parámetro `DryRun` para probar si tiene permiso para cambiar la monitorización de instancias antes de intentar realmente cambiar la monitorización de estas instancias.

```
// 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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos y especifique ON para comenzar la monitorización detallada u OFF para interrumpir la monitorización.

```
node ec2_monitorinstances.js ON
```

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Inicio y detención de instancias

Cree un módulo de Node.js con el nombre de archivo `ec2_startstopinstances.js`. Asegúrese de configurar el SDK tal y como se ha indicado anteriormente. Para acceder a Amazon EC2, cree un objeto de servicio de AWS . EC2. Añada los ID de las instancias que desee iniciar o detener.

Según el valor de un argumento de línea de comando (START o STOP), llame al método `startInstances` del objeto de servicio de Amazon EC2 para iniciar las instancias especificadas o al método `stopInstances` para detenerlas. Utilice el parámetro `DryRun` para probar si tiene permiso antes de intentar realmente iniciar o detener las instancias seleccionadas.

```
// 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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos y especifique START para iniciar las instancias o STOP para detenerlas.

```
node ec2_startstopinstances.js START INSTANCE_ID
```

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Reinicio de instancias

Cree un módulo de Node.js con el nombre de archivo `ec2_rebootinstances.js`. Asegúrese de configurar el SDK tal y como se ha indicado anteriormente. Para acceder a Amazon EC2,

Cree un objeto de servicio de Amazon EC2. Añada los ID de las instancias que desee reiniciar. Llame al método `rebootInstances` del objeto de servicio `AWS . EC2` para reiniciar las instancias especificadas. Utilice el parámetro `DryRun` para probar si tiene permiso para reiniciar estas instancias antes de intentar realmente reiniciarlas.

```
// 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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos.

```
node ec2_rebootinstances.js
```

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Uso de pares de claves de Amazon EC2



Este ejemplo de código de Node.js muestra:

- Cómo recuperar información acerca de sus pares de clave.
- Cómo crear un par de claves para obtener acceso a una instancia de Amazon EC2.
- Cómo eliminar un par de claves ya existente.

El escenario

Amazon EC2 utiliza la criptografía de clave pública para cifrar y descifrar la información de inicio de sesión. En la criptografía de clave pública se utiliza una clave pública para cifrar datos y, a continuación, el destinatario utiliza la clave privada para descifrar los datos. El conjunto de clave pública y clave privada se denomina par de claves.

En este ejemplo, va a utilizar una serie de módulos de Node.js para realizar varias operaciones de administración de pares de claves de Amazon EC2. Los módulos de Node.js usan el SDK para JavaScript para gestionar instancias usando estos métodos de clase de cliente de Amazon EC2:

- [createKeyPair](#)
- [deleteKeyPair](#)
- [describeKeyPairs](#)

Para obtener más información acerca de los pares de claves de Amazon EC2, consulte [Pares de claves de Amazon EC2](#) en la Guía del usuario de Amazon EC2 o [Pares de claves de Amazon EC2 e instancias de Windows](#) en la Guía del usuario de Amazon EC2.

Tareas previas necesarias

Para configurar y ejecutar este ejemplo, primero debe completar las tareas siguientes:

- Instale Node.js. Para obtener más información acerca de la instalación de Node.js consulte el [sitio web de Node.js](#).

- Cree un archivo de configuraciones compartidas con sus credenciales de usuario. Para obtener más información acerca de cómo crear un archivo de credenciales compartidas, consulte [Carga de credenciales en Node.js desde el archivo de credenciales compartidas](#).

Descripción de sus pares de claves

Cree un módulo de Node.js con el nombre de archivo `ec2_describekeypairs.js`. Asegúrese de configurar el SDK tal y como se ha indicado anteriormente. Para acceder a Amazon EC2, cree un objeto de servicio de `AWS.EC2`. Cree un objeto JSON vacío para almacenar los parámetros que el método `describeKeyPairs` necesita para devolver las descripciones de todos sus pares de claves. También puede proporcionar una matriz de nombres de pares de claves en la parte `KeyName` de los parámetros en el archivo JSON al 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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos.

```
node ec2_describekeypairs.js
```

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Creación de un par de claves

Cada par de claves requiere un nombre. Amazon EC2 asocia la clave pública al nombre que especifique como nombre de la clave. Cree un módulo de Node.js con el nombre de

archivo `ec2_createkeypair.js`. Asegúrese de configurar el SDK tal y como se ha indicado anteriormente. Para acceder a Amazon EC2, cree un objeto de servicio de `AWS.EC2`. Cree los parámetros de JSON para especificar el nombre del par de claves y luego transfírelos para llamar al 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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos.

```
node ec2_createkeypair.js
```

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Eliminación de un par de claves

Cree un módulo de Node.js con el nombre de archivo `ec2_deletekeypair.js`. Asegúrese de configurar el SDK tal y como se ha indicado anteriormente. Para acceder a Amazon EC2, cree un objeto de servicio de `AWS.EC2`. Cree los parámetros JSON para especificar el nombre del par de claves que desea eliminar. A continuación, llame al 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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos.

```
node ec2_deletekeypair.js
```

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Uso de las regiones y las zonas de disponibilidad de Amazon EC2



Este ejemplo de código de Node.js muestra:

- Cómo recuperar descripciones para regiones y zonas de disponibilidad.

El escenario

Amazon EC2 está alojado en varias ubicaciones de todo el mundo. Dichas ubicaciones se componen de regiones y zonas de disponibilidad. Cada región es un área geográfica independiente. Cada

región tiene varias ubicaciones aisladas conocidas como zonas de disponibilidad. Amazon EC2 ofrece la posibilidad de colocar instancias y datos en varias ubicaciones.

En este ejemplo, va a utilizar una serie de módulos de Node.js para recuperar detalles sobre regiones y zonas de disponibilidad. Los módulos de Node.js usan el SDK para JavaScript para gestionar instancias usando los métodos siguientes de clase de cliente de Amazon EC2:

- [describeAvailabilityZones](#)
- [describeRegions](#)

Para obtener más información sobre las regiones y las zonas de disponibilidad, consulte [Regiones y zonas de disponibilidad](#) en la Guía del usuario de Amazon EC2 o [Regiones y zonas de disponibilidad](#) en la Guía del usuario de Amazon EC2.

Tareas previas necesarias

Para configurar y ejecutar este ejemplo, primero debe completar estas tareas:

- Instale Node.js. Para obtener más información acerca de la instalación de Node.js consulte el [sitio web de Node.js](#).
- Cree un archivo de configuraciones compartidas con sus credenciales de usuario. Para obtener más información acerca de cómo crear un archivo de credenciales compartidas, consulte [Carga de credenciales en Node.js desde el archivo de credenciales compartidas](#).

Descripción de regiones y zonas de disponibilidad

Cree un módulo de Node.js con el nombre de archivo `ec2_describeregionsandzones.js`. Asegúrese de configurar el SDK tal y como se ha indicado anteriormente. Para acceder a Amazon EC2, cree un objeto de servicio de AWS . EC2. Cree un objeto JSON vacío para transferirlo como parámetros que devuelvan todas las descripciones disponibles. A continuación, llame a los métodos `describeRegions` y `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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos.

```
node ec2_describeregionsandzones.js
```

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Trabajar con grupos de seguridad en Amazon EC2



Este ejemplo de código de Node.js muestra:

- Cómo recuperar información acerca de sus grupos de seguridad.
- Cómo crear un grupo de seguridad para obtener acceso a una instancia de Amazon EC2.
- Cómo eliminar un grupo de seguridad ya existente.

El escenario

Un grupo de seguridad de Amazon EC2 funciona como un firewall virtual que controla el tráfico de una o varias instancias. Se añaden reglas a cada grupo de seguridad para permitir el tráfico con sus instancias asociadas. Puede modificar las reglas de un grupo de seguridad en cualquier momento: las nuevas reglas se aplican automáticamente a todas las instancias que están asociadas al grupo de seguridad.

En este ejemplo, va a utilizar una serie de módulos de Node.js para realizar varias operaciones de Amazon EC2 en las que haya grupos de seguridad. Los módulos de Node.js usan el SDK para JavaScript para gestionar instancias usando los métodos siguientes de clase de cliente de Amazon EC2:

- [describeSecurityGroups](#)
- [authorizeSecurityGroupIngress](#)
- [createSecurityGroup](#)
- [describeVpcs](#)
- [deleteSecurityGroup](#)

Para obtener más información sobre los grupos de seguridad de Amazon EC2, consulte [Grupos de seguridad de Amazon EC2 para instancias de Linux](#) en la Guía del usuario de Amazon EC2 o [Grupos de seguridad de Amazon EC2 para instancias de Windows](#) en la Guía del usuario de Amazon EC2.

Tareas previas necesarias

Para configurar y ejecutar este ejemplo, primero debe completar las tareas siguientes:

- Instale Node.js. Para obtener más información acerca de la instalación de Node.js consulte el [sitio web de Node.js](#).
- Cree un archivo de configuraciones compartidas con sus credenciales de usuario. Para obtener más información acerca de cómo crear un archivo de credenciales compartidas, consulte [Carga de credenciales en Node.js desde el archivo de credenciales compartidas](#).

Descripción de los grupos de seguridad

Cree un módulo de Node.js con el nombre de archivo `ec2_describesecuritygroups.js`. Asegúrese de configurar el SDK tal y como se ha indicado anteriormente. Para acceder a Amazon

EC2, cree un objeto de servicio de AWS . EC2. Cree un objeto JSON para transferirlo como parámetros como, por ejemplo, los ID de los grupos de seguridad que desea describir. Luego llame al método `describeSecurityGroups` del objeto de servicio de 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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos.

```
node ec2_describesecuritygroups.js
```

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Creación de un grupo de seguridad y reglas

Cree un módulo de Node.js con el nombre de archivo `ec2_createsecuritygroup.js`. Asegúrese de configurar el SDK tal y como se ha indicado anteriormente. Para acceder a Amazon EC2, cree un objeto de servicio de AWS . EC2. Cree un objeto JSON para los parámetros que especifiquen el nombre del grupo de seguridad, una descripción y el ID de la VPC. Transfiera los parámetros al método `createSecurityGroup`.

Después de crear correctamente el grupo de seguridad, puede definir sus reglas para permitir el tráfico de entrada. Cree un objeto JSON para parámetros que especifiquen el protocolo IP y

los puertos de entrada donde la instancia de Amazon EC2 va a recibir el tráfico. Transfiera los parámetros al 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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos.

```
node ec2_createsecuritygroup.js
```

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Eliminación de un grupo de seguridad

Cree un módulo de Node.js con el nombre de archivo `ec2_deletesecuritygroup.js`. Asegúrese de configurar el SDK tal y como se ha indicado anteriormente. Para acceder a Amazon EC2, cree un objeto de servicio de AWS . EC2. Cree los parámetros JSON para especificar el nombre del grupo de seguridad que desea eliminar. A continuación, llame al 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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos.

```
node ec2_deletesecuritygroup.js
```

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Uso de direcciones IP elásticas en Amazon EC2



Este ejemplo de código de Node.js muestra:

- Cómo recuperar descripciones de sus direcciones IP elásticas.
- Cómo asignar y liberar una dirección IP elástica.
- Cómo asociar una dirección IP elástica a una instancia de Amazon EC2.

El escenario

Una dirección IP elástica es una dirección IP estática que se ha diseñado para la informática en la nube dinámica. Una dirección IP elástica se asocia a su cuenta de AWS. Se trata de una dirección IP pública a la que se puede obtener acceso desde Internet. Si la instancia no tiene una dirección IP pública, puede asociar una dirección IP elástica (EIP) a la instancia para habilitar la comunicación con Internet.

En este ejemplo, va a utilizar una serie de módulos de Node.js para realizar varias operaciones de Amazon EC2 en las que haya direcciones IP elásticas. Los módulos de Node.js usan el SDK para

JavaScript para gestionar direcciones IP elásticas usando estos métodos de clase de cliente de Amazon EC2:

- [describeAddresses](#)
- [allocateAddress](#)
- [associateAddress](#)
- [releaseAddress](#)

Para obtener más información acerca de las direcciones IP elásticas en Amazon EC2, consulte [Direcciones IP elásticas](#) en la Guía del usuario de Amazon EC2 o [Direcciones IP elásticas](#) en la Guía del usuario de Amazon EC2.

Tareas previas necesarias

Para configurar y ejecutar este ejemplo, primero debe completar las tareas siguientes:

- Instale Node.js. Para obtener más información acerca de la instalación de Node.js consulte el [sitio web de Node.js](#).
- Cree un archivo de configuraciones compartidas con sus credenciales de usuario. Para obtener más información acerca de cómo crear un archivo de credenciales compartidas, consulte [Carga de credenciales en Node.js desde el archivo de credenciales compartidas](#).
- Crear una instancia de Amazon EC2. Para obtener más información sobre la creación de instancias de Amazon EC2, consulte [Instancias de Amazon EC2](#) en la Guía del usuario de Amazon EC2 o [Instancias de Amazon EC2](#) en la Guía del usuario de Amazon EC2.

Descripción de direcciones IP elásticas

Cree un módulo de Node.js con el nombre de archivo `ec2_describeaddresses.js`. Asegúrese de configurar el SDK tal y como se ha indicado anteriormente. Para acceder a Amazon EC2, cree un objeto de servicio de `AWS.EC2`. Cree un objeto JSON para transferirlo como parámetros como el filtro de las direcciones que devuelven las direcciones en su VPC. Para recuperar las descripciones de todas las direcciones IP elásticas, omita un filtro del JSON de parámetros. Luego llame al método `describeAddresses` del objeto de servicio de 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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos.

```
node ec2_describeaddresses.js
```

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Asignación y asociación de una dirección IP elástica a una instancia de Amazon EC2

Cree un módulo de Node.js con el nombre de archivo `ec2_allocateaddress.js`. Asegúrese de configurar el SDK tal y como se ha indicado anteriormente. Para acceder a Amazon EC2, cree un objeto de servicio de `AWS.EC2`. Cree un objeto JSON para los parámetros que se utilizan para asignar una dirección IP elástica, que en este caso especifica que el `Domain` es una VPC. Llame al método `allocateAddress` del objeto de servicio de Amazon EC2.

Si la llamada se realiza correctamente, el parámetro `data` a la función de devolución de llamada tiene una propiedad `AllocationId` que identifica la dirección IP elástica asignada.

Cree un objeto JSON para los parámetros que se usan para asociar una dirección IP elástica a una instancia de Amazon EC2, incluido el `AllocationId` de la dirección recién asignada y el `InstanceId` de la instancia de Amazon EC2. Luego llame al método `associateAddresses` del objeto de servicio de 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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos.

```
node ec2_allocateaddress.js
```

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Liberación de una dirección IP elástica

Cree un módulo de Node.js con el nombre de archivo `ec2_releaseaddress.js`. Asegúrese de configurar el SDK tal y como se ha indicado anteriormente. Para acceder a Amazon EC2, cree un objeto de servicio de AWS . EC2. Cree un objeto JSON para los parámetros que se usan para liberar

una dirección IP elástica, que en este caso especifica el `AllocationId` de la dirección IP elástica. La liberación de una dirección IP elástica también la desvincula de cualquier instancia de Amazon EC2. Llame al método `releaseAddress` del objeto de servicio de 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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos.

```
node ec2_releaseaddress.js
```

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

AWS Elemental MediaConvertEjemplos de

AWS Elemental MediaConvert es un servicio de transcodificación de vídeo basado en archivos con características apropiadas para los medios de difusión. Puede utilizarlo para crear recursos para su difusión y para la entrega de vídeo bajo demanda (VOD) a través de Internet. Para obtener más información, consulte la [Guía del usuario de AWS Elemental MediaConvert](#).

La API de JavaScript para MediaConvert se expone a través de la clase de cliente de `AWS.MediaConvert`. Para obtener más información, consulte [Class: AWS.MediaConvert](#) en la referencia de la API.

Temas

- [Creación y administración de trabajos de transcodificación en MediaConvert](#)
- [Uso de plantillas de trabajo en MediaConvert](#)

Creación y administración de trabajos de transcodificación en MediaConvert



Este ejemplo de código de Node.js muestra:

- Cómo crear trabajos de transcodificación en MediaConvert.
- Cómo cancelar un trabajo de transcodificación.
- Cómo recuperar el JSON de un trabajo de transcodificación finalizado.
- Cómo recuperar una matriz JSON para un máximo de 20 de los últimos trabajos creados.

El escenario

En este ejemplo, se utiliza un módulo de Node.js para llamar a MediaConvert para crear y administrar trabajos de transcodificación. El código usa el SDK de JavaScript para realizar esta operación mediante los métodos de la clase de cliente de MediaConvert siguientes:

- [createJob](#)
- [cancelJob](#)
- [getJob](#)
- [listJobs](#)

Tareas previas necesarias

Para configurar y ejecutar este ejemplo, primero debe completar las tareas siguientes:

- Instale Node.js. Para obtener más información, consulte el sitio web de [Node.js](#).

- Cree un archivo de configuraciones compartidas con sus credenciales de usuario. Para obtener más información acerca de cómo crear un archivo de credenciales compartidas, consulte [Carga de credenciales en Node.js desde el archivo de credenciales compartidas](#).
- Crear y configurar buckets de Amazon S3 que proporcionan almacenamiento para archivos de entrada de trabajo y archivos de salida. Para obtener información detallada, consulte [Crear almacenamiento para archivos](#) en la Guía del usuario de AWS Elemental MediaConvert.
- Cargar el vídeo de entrada en el bucket de Amazon S3 que ha provisionado para el almacenamiento de entrada. Para obtener una lista de los códecs y contenedores compatibles con la entrada de vídeo, consulte [Códecs y contenedores de entrada compatibles](#) en la Guía del usuario de AWS Elemental MediaConvert.
- Cree un rol de IAM que conceda a MediaConvert acceso a los archivos de entrada y a los buckets de Amazon S3 donde se almacenan los archivos de salida. Para obtener más información, consulte [Configuración de los permisos de IAM](#) en la Guía del usuario de AWS Elemental MediaConvert.

Definición de un trabajo de transcodificación sencillo

Cree un módulo de Node.js con el nombre de archivo `emc_createjob.js`. Asegúrese de configurar el SDK tal y como se ha indicado anteriormente. Cree el JSON que define los parámetros del trabajo de transcodificación.

Estos parámetros son bastante detallados. Puede utilizar la [consola de AWS Elemental MediaConvert](#) para generar los parámetros JSON del trabajo eligiendo la configuración del trabajo en la consola y, a continuación, eligiendo Mostrar JSON del trabajo en la parte inferior de la sección Trabajo. En este ejemplo, se muestra el JSON de un trabajo sencillo.

```
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",  
},  
};
```

Creación de un trabajo de transcodificación

Después de crear el JSON de los parámetros del trabajo, llame al método `createJob` creando una promesa que invoque un objeto de servicio `AWS.MediaConvert` mediante la transferencia de los parámetros. Luego gestione la respuesta en la devolución de llamada de la promesa. El ID del trabajo creado se devuelve en la respuesta `data`.

```
// 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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos.

```
node emc_createjob.js
```

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Cancelación de un trabajo de transcodificación

Cree un módulo de Node.js con el nombre de archivo `emc_canceljob.js`. Asegúrese de configurar el SDK tal y como se ha indicado anteriormente. Cree el JSON que incluye el ID del trabajo que desea cancelar. Luego llame al método `cancelJob` creando una promesa para invocar un objeto de servicio `AWS.MediaConvert` mediante los parámetros. Gestione la respuesta en la devolución de llamada de la promesa.

```
// 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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos.

```
node ec2_canceljob.js
```

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Listado de los trabajos de transcodificación recientes

Cree un módulo de Node.js con el nombre de archivo `emc_listjobs.js`. Asegúrese de configurar el SDK tal y como se ha indicado anteriormente.

Cree el JSON de los parámetros, incluidos los valores que especificarán si se debe ordenar la lista en orden ASCENDING o DESCENDING, el ARN de la cola de trabajos que se va comprobar y el estado de los trabajos que se deben incluir. Luego llame al método `listJobs` creando una promesa para invocar un objeto de servicio `AWS.MediaConvert` mediante los parámetros. Gestione la respuesta en la devolución de llamada de la promesa.

```
// 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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos.

```
node emc_listjobs.js
```

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Uso de plantillas de trabajo en MediaConvert



Este ejemplo de código de Node.js muestra:

- Cómo crear una plantilla de trabajo personalizada de MediaConvert.
- Cómo utilizar una plantilla de trabajo para crear un trabajo de transcodificación.
- Cómo generar una lista de todas sus plantillas de trabajo.
- Cómo eliminar plantillas de trabajos.

El escenario

El JSON necesario para crear un trabajo de transcodificación en MediaConvert es detallado, ya que contiene un gran número de opciones de configuración. Puede simplificar en gran medida la creación del trabajo guardando la configuración de funcionalidad comprobada en una plantilla de trabajo que pueda utilizar para crear trabajos posteriores. En este ejemplo, se utiliza un módulo de Node.js para llamar a MediaConvert para crear, utilizar y administrar plantillas de trabajos. El código usa el SDK de JavaScript para realizar esta operación mediante los métodos de la clase de cliente de MediaConvert siguientes:

- [createJobTemplate](#)
- [createJob](#)
- [deleteJobTemplate](#)
- [listJobTemplates](#)

Tareas previas necesarias

Para configurar y ejecutar este ejemplo, primero debe completar las tareas siguientes:

- Instale Node.js. Para obtener más información, consulte el sitio web de [Node.js](#).
- Cree un archivo de configuraciones compartidas con sus credenciales de usuario. Para obtener más información acerca de cómo crear un archivo de credenciales compartidas, consulte [Carga de credenciales en Node.js desde el archivo de credenciales compartidas](#).
- Cree un rol de IAM que conceda a MediaConvert acceso a los archivos de entrada y a los buckets de Amazon S3 donde se almacenan los archivos de salida. Para obtener más información, consulte [Configuración de los permisos de IAM](#) en la Guía del usuario de AWS Elemental MediaConvert.

Creación de una plantilla de trabajo

Cree un módulo de Node.js con el nombre de archivo `emc_create_jobtemplate.js`. Asegúrese de configurar el SDK tal y como se ha indicado anteriormente.

Especifique los parámetros JSON para la creación de plantillas. Puede utilizar la mayoría de los parámetros JSON de un trabajo anterior realizado correctamente para especificar los valores de Settings en la plantilla. En este ejemplo se utiliza la configuración de trabajo de [Creación y administración de trabajos de transcodificación en MediaConvert](#).

Llame al método `createJobTemplate` creando una promesa para invocar un objeto de servicio AWS. MediaConvert mediante los parámetros. Luego gestione la respuesta en la devolución de llamada de la promesa.

```
// 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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos.

```
node emc_create_jobtemplate.js
```

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Creación de un trabajo de transcodificación a partir de una plantilla de trabajo

Cree un módulo de Node.js con el nombre de archivo `emc_template_createjob.js`. Asegúrese de configurar el SDK tal y como se ha indicado anteriormente.

Cree los parámetros JSON de creación del trabajo, como el nombre de la plantilla de trabajo que desea utilizar y el método `Settings` que se utilizará, que son específicos del trabajo que está creando. Luego llame al método `createJobs` creando una promesa para invocar un objeto de servicio AWS `MediaConvert` mediante los parámetros. Gestione la respuesta en la devolución de llamada de la promesa.

```
// 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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos.

```
node emc_template_createjob.js
```

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Generación de una lista de sus plantillas de trabajo

Cree un módulo de Node.js con el nombre de archivo `emc_listtemplates.js`. Asegúrese de configurar el SDK tal y como se ha indicado anteriormente.

Cree un objeto para pasar los parámetros de solicitud vacíos para el método `listTemplates` de la clase de cliente `AWS.MediaConvert`. Incluya valores para determinar qué plantillas incluirá en la lista (`NAME`, `CREATION_DATE`, `SYSTEM`), la cantidad que se incluirán en la lista y su orden de clasificación. Para llamar al método `listTemplates`, cree una promesa para invocar un objeto de servicio de `MediaConvert` mediante la transferencia de los parámetros. Luego gestione la respuesta en la devolución de llamada de la promesa.

```
// 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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos.

```
node emc_listtemplates.js
```

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Eliminación de una plantilla de trabajo

Cree un módulo de Node.js con el nombre de archivo `emc_deletetemplate.js`. Asegúrese de configurar el SDK tal y como se ha indicado anteriormente.

Cree un objeto para transferir el nombre de la plantilla de trabajo que desea eliminar como parámetros para el método `deleteJobTemplate` de la clase de cliente `AWS.MediaConvert`. Para llamar al método `deleteJobTemplate`, cree una promesa para invocar un objeto de servicio de `MediaConvert` mediante la transferencia de los parámetros. Gestione la respuesta en la devolución de llamada de la promesa.

```
// 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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos.

```
node emc_deletetemplate.js
```

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Ejemplos de AWS IAM

AWS Identity and Access Management (IAM) es un servicio web que permite a los clientes de Amazon Web Services administrar usuarios y permisos de usuario en AWS. El servicio está dirigido a las organizaciones con múltiples usuarios o sistemas en la nube que utilizan productos de AWS. Con IAM, puede administrar de forma centralizada los usuarios, las credenciales de seguridad, como las claves de acceso, y los permisos que controlan los recursos de AWS a los que pueden acceder los usuarios.



La API de JavaScript para IAM se expone a través de la clase de cliente `AWS.IAM`. Para obtener más información acerca de cómo usar la clase de cliente de IAM, consulte [Class: AWS.IAM](#) en la referencia de la API.

Temas

- [Administración de usuarios de IAM](#)
- [Uso de políticas de IAM](#)
- [Administración de las claves de acceso de IAM](#)
- [Uso de certificados de servidor de IAM](#)
- [Administración de alias de cuenta de IAM](#)

Administración de usuarios de IAM



Este ejemplo de código de Node.js muestra:

- Cómo recuperar una lista de usuarios de IAM.
- Cómo crear y eliminar usuarios.
- Cómo actualizar un nombre de usuario.

El escenario

En este ejemplo se van a utilizar una serie de módulos de Node.js para crear y administrar usuarios en IAM. Los módulos de Node.js utilizan el SDK para JavaScript para crear, eliminar y actualizar usuarios mediante los métodos siguientes de la clase de cliente AWS . IAM:

- [createUser](#)
- [listUsers](#)
- [updateUser](#)
- [getUser](#)
- [deleteUser](#)

Para obtener más información acerca de IAM, consulte [Usuarios de IAM](#) en la Guía del usuario de IAM.

Tareas previas necesarias

Para configurar y ejecutar este ejemplo, primero debe completar estas tareas:

- Instale Node.js. Para obtener más información acerca de la instalación de Node.js consulte el [sitio web de Node.js](#).
- Cree un archivo de configuraciones compartidas con sus credenciales de usuario. Para obtener más información acerca de cómo crear un archivo de credenciales compartidas, consulte [Carga de credenciales en Node.js desde el archivo de credenciales compartidas](#).

Crear un usuario

Cree un módulo de Node.js con el nombre de archivo `iam_createuser.js`. Asegúrese de configurar el SDK tal y como se ha indicado anteriormente. Para obtener acceso a IAM, cree un objeto de servicio `AWS.IAM`. Cree un objeto JSON que contenga los parámetros necesarios y que consista en el nombre de usuario que desea usar para el nuevo usuario como un parámetro de línea de comandos.

Llame al método `getUser` del objeto de servicio `AWS.IAM` para ver si el nombre de usuario ya existe. Si el nombre de usuario no existe, llame al método `createUser` para crearlo. Si el nombre ya existe, escriba un mensaje a tal efecto a la consola.

```
// 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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos.

```
node iam_createuser.js USER_NAME
```

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Generación de una lista de usuarios en su cuenta

Cree un módulo de Node.js con el nombre de archivo `iam_listusers.js`. Asegúrese de configurar el SDK tal y como se ha indicado anteriormente. Para obtener acceso a IAM, cree un objeto de servicio `AWS.IAM`. Cree un objeto JSON que contenga los parámetros necesarios para generar una lista de sus usuarios y limite el número de resultados devueltos estableciendo el parámetro `MaxItems` en 10. Llame al método `listUsers` del objeto de servicio de `AWS.IAM`. Escriba el primer nombre de usuario y la fecha de creación en la consola.

```
// 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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos.

```
node iam_listusers.js
```

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Actualización de un nombre de un usuario

Cree un módulo de Node.js con el nombre de archivo `iam_updateuser.js`. Asegúrese de configurar el SDK tal y como se ha indicado anteriormente. Para obtener acceso a IAM, cree un objeto de servicio `AWS.IAM`. Cree un objeto JSON que contenga los parámetros necesarios para generar una lista de sus usuarios y especifique los nombres de usuario actual y nuevo como parámetros de línea de comandos. Llame al método `updateUser` del objeto de servicio de `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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos; especifique el nombre de usuario actual seguido del nuevo nombre de usuario.

```
node iam_updateuser.js ORIGINAL_USERNAME NEW_USERNAME
```

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Eliminar un usuario

Cree un módulo de Node.js con el nombre de archivo `iam_deleteuser.js`. Asegúrese de configurar el SDK tal y como se ha indicado anteriormente. Para obtener acceso a IAM, cree un

objeto de servicio AWS . IAM. Cree un objeto JSON que contenga los parámetros necesarios y que consista en el nombre de usuario que desea eliminar como un parámetro de línea de comandos.

Llame al método `getUser` del objeto de servicio AWS . IAM para ver si el nombre de usuario ya existe. Si el nombre de usuario no existe, escriba un mensaje a tal efecto a la consola. Si el usuario existe, llame al método `deleteUser` para eliminarlo.

```
// 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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos.

```
node iam_deleteuser.js USER_NAME
```

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Uso de políticas de IAM



Este ejemplo de código de Node.js muestra:

- Cómo crear y eliminar políticas de IAM.
- Cómo asociar y desconectar políticas de IAM de los roles.

El escenario

Conceda permisos a un usuario mediante una política, que es un documento que elabora una lista de las acciones que puede realizar un usuario y los recursos que afectan a dichas acciones. De forma predeterminada, todas las acciones o recursos que no se permiten de forma explícita se deniegan. Puede crear políticas y asociarlas a usuarios, a grupos de usuarios, a roles asumidos por usuarios y a recursos.

En este ejemplo, se van a utilizar una serie de módulos de Node.js para administrar políticas en IAM. Los módulos de Node.js usan el SDK para JavaScript para crear y eliminar políticas, así como para asociar y desvincular políticas de roles utilizando estos métodos de la clase de cliente de AWS . IAM:

- [createPolicy](#)
- [getPolicy](#)
- [listAttachedRolePolicies](#)
- [attachRolePolicy](#)
- [detachRolePolicy](#)

Para obtener más información sobre las políticas de IAM, consulte [Información general sobre la administración del acceso: permisos y políticas](#) en la Guía del usuario de IAM.

Tareas previas necesarias

Para configurar y ejecutar este ejemplo, primero debe completar estas tareas:

- Instale Node.js. Para obtener más información acerca de la instalación de Node.js consulte el [sitio web de Node.js](#).
- Cree un archivo de configuraciones compartidas con sus credenciales de usuario. Para obtener más información acerca de cómo crear un archivo de credenciales compartidas, consulte [Carga de credenciales en Node.js desde el archivo de credenciales compartidas](#).
- Cree un rol de IAM al que pueda asociar políticas. Para obtener más información sobre la creación de roles, consulte [Creación de roles de IAM](#) en la Guía del usuario de IAM.

Crear una política de IAM

Cree un módulo de Node.js con el nombre de archivo `iam_createpolicy.js`. Asegúrese de configurar el SDK tal y como se ha indicado anteriormente. Para obtener acceso a IAM, cree un objeto de servicio `AWS.IAM`. Cree dos objetos JSON, uno de ellos que contenga el documento de política que desea crear y el otro que contenga los parámetros necesarios para crear la política, lo que incluye el JSON de política y el nombre que desea dar a la política. Asegúrese de convertir en representación textual al objeto JSON de política en los parámetros. Llame al método `createPolicy` del objeto de servicio de `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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos.

```
node iam_createpolicy.js
```

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Obtención de una política de IAM

Cree un módulo de Node.js con el nombre de archivo `iam_getpolicy.js`. Asegúrese de configurar el SDK tal y como se ha indicado anteriormente. Para obtener acceso a IAM, cree un objeto de servicio `AWS.IAM`. Cree un objeto JSON que contenga los parámetros necesarios para recuperar una política, que es el ARN de la política que desea obtener. Llame al método `getPolicy` del objeto de servicio de `AWS.IAM`. Escriba la descripción de la política en la consola.

```
// 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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos.

```
node iam_getpolicy.js
```

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Asociación de una política de roles administrada

Cree un módulo de Node.js con el nombre de archivo `iam_attachrolepolicy.js`. Asegúrese de configurar el SDK tal y como se ha indicado anteriormente. Para obtener acceso a IAM, cree un objeto de servicio `AWS.IAM`. Cree un objeto JSON que contenga los parámetros necesarios para obtener una lista de políticas de IAM administradas asociada a un rol, que se compone del nombre del rol. Proporcione el nombre del rol como un parámetro de línea de comandos. Llame al método `listAttachedRolePolicies` del objeto de servicio `AWS.IAM`, que devuelve una matriz de políticas administradas a la función de devolución de llamada.

Compruebe los miembros de la matriz para ver si la política que desea asociar al rol ya se ha asociado. Si la política no está asociada, llame al método `attachRolePolicy` para asociarla.

```
// 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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos.

```
node iam_attachrolepolicy.js IAM_ROLE_NAME
```

Desvinculación de una política de roles administrada

Cree un módulo de Node.js con el nombre de archivo `iam_detachrolepolicy.js`. Asegúrese de configurar el SDK tal y como se ha indicado anteriormente. Para obtener acceso a IAM, cree un objeto de servicio AWS . IAM. Cree un objeto JSON que contenga los parámetros necesarios para obtener una lista de políticas de IAM administradas asociada a un rol, que se compone del nombre del rol. Proporcione el nombre del rol como un parámetro de línea de comandos. Llame al método `listAttachedRolePolicies` del objeto de servicio AWS . IAM, que devuelve una matriz de políticas administradas en la función de devolución de llamada.

Compruebe los miembros de la matriz para ver si la política que desea desvincular del rol está asociada. Si la política está asociada, llame al método `detachRolePolicy` para desvincularla.

```
// 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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos.

```
node iam_detachrolepolicy.js IAM_ROLE_NAME
```

Administración de las claves de acceso de IAM



Este ejemplo de código de Node.js muestra:

- Cómo administrar las claves de acceso de sus usuarios.

El escenario

Los usuarios necesitan sus propias claves de acceso para hacer llamadas mediante programación a AWS desde el SDK para JavaScript. Para atender esta necesidad, puede crear, modificar, ver o rotar claves de acceso (ID de clave de acceso y claves de acceso secretas) de los usuarios de IAM. De forma predeterminada, cuando crea una clave de acceso, su estado es `Active`, lo que significa que el usuario puede utilizar la clave de acceso para las llamadas a la API.

En este ejemplo, se van a utilizar una serie de módulos de Node.js para administrar las claves de acceso en IAM. Los módulos de Node.js usan el SDK para JavaScript para gestionar claves de acceso de IAM mediante los métodos de clase de cliente AWS . IAM siguientes:

- [createAccessKey](#)
- [listAccessKeys](#)
- [getAccessKeyLastUsed](#)
- [updateAccessKey](#)
- [deleteAccessKey](#)

Para obtener más información acerca de las claves de acceso, consulte [Claves de acceso](#) en la Guía del usuario de IAM.

Tareas previas necesarias

Para configurar y ejecutar este ejemplo, primero debe completar estas tareas:

- Instale Node.js. Para obtener más información acerca de la instalación de Node.js consulte el [sitio web de Node.js](#).

- Cree un archivo de configuraciones compartidas con sus credenciales de usuario. Para obtener más información acerca de cómo crear un archivo de credenciales compartidas, consulte [Carga de credenciales en Node.js desde el archivo de credenciales compartidas](#).

Creación de claves de acceso para un usuario

Cree un módulo de Node.js con el nombre de archivo `iam_createaccesskeys.js`. Asegúrese de configurar el SDK tal y como se ha indicado anteriormente. Para obtener acceso a IAM, cree un objeto de servicio `AWS.IAM`. Cree un objeto JSON que contenga los parámetros necesarios para crear nuevas claves de acceso y que incluya el nombre del usuario de IAM. Llame al método `createAccessKey` del objeto de servicio de `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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos. Asegúrese de transferir los datos devueltos a un archivo de texto para no perder la clave secreta, que solo puede proporcionarse una vez.

```
node iam_createaccesskeys.js > newuserkeys.txt
```

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Generación de una lista de claves de acceso de un usuario

Cree un módulo de Node.js con el nombre de archivo `iam_listaccesskeys.js`. Asegúrese de configurar el SDK tal y como se ha indicado anteriormente. Para obtener acceso a IAM, cree un

objeto de servicio `AWS.IAM`. Cree un objeto JSON que contenga los parámetros necesarios para recuperar las claves de acceso del usuario y que incluya el nombre del usuario de IAM y, de forma opcional, el número máximo de pares de claves de acceso que desee que aparezcan en la lista. Llame al método `listAccessKeys` del objeto de servicio de `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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos.

```
node iam_listaccesskeys.js
```

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Obtención de información del último uso de las claves de acceso

Cree un módulo de Node.js con el nombre de archivo `iam_accesskeylastused.js`. Asegúrese de configurar el SDK tal y como se ha indicado anteriormente. Para obtener acceso a IAM, cree un objeto de servicio `AWS.IAM`. Cree un objeto JSON que contenga los parámetros necesarios para crear nuevas claves de acceso, que es el ID de clave de acceso cuya información sobre la última vez que se usó quiere obtener. Llame al método `getAccessKeyLastUsed` del objeto de servicio de `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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos.

```
node iam_accesskeylastused.js
```

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Actualización del estado de la clave de acceso

Cree un módulo de Node.js con el nombre de archivo `iam_updateaccesskey.js`. Asegúrese de configurar el SDK tal y como se ha indicado anteriormente. Para obtener acceso a IAM, cree un objeto de servicio `AWS.IAM`. Cree un objeto JSON que contenga los parámetros necesarios para actualizar el estado de una clave de acceso, que incluya el ID de clave de acceso y el estado actualizado. El estado puede ser `Active` o `Inactive`. Llame al método `updateAccessKey` del objeto de servicio de `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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos.

```
node iam_updateaccesskey.js
```

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Eliminación de claves de acceso

Cree un módulo de Node.js con el nombre de archivo `iam_deleteaccesskey.js`. Asegúrese de configurar el SDK tal y como se ha indicado anteriormente. Para obtener acceso a IAM, cree un objeto de servicio `AWS.IAM`. Cree un objeto JSON que contenga los parámetros necesarios para eliminar claves de acceso, que incluya el ID de clave de acceso y el nombre del usuario. Llame al método `deleteAccessKey` del objeto de servicio de `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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos.

```
node iam_deleteaccesskey.js
```

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Uso de certificados de servidor de IAM



Este ejemplo de código de Node.js muestra:

- Cómo realizar tareas básicas al administrar certificados de servidor para conexiones HTTPS.

El escenario

Para habilitar las conexiones HTTPS en su sitio web o aplicación en AWS, necesita un certificado de servidor SSL/TLS. Para utilizar un certificado obtenido de un proveedor externo en su sitio web o aplicación en AWS, debe cargar el certificado en IAM o importarlo en AWS Certificate Manager.

En este ejemplo, se van a utilizar una serie de módulos de Node.js para gestionar certificados de servidor en IAM. Los módulos de Node.js usan el SDK para JavaScript para administrar certificados de servidor mediante los métodos de clase de cliente AWS . IAM siguientes:

- [listServerCertificates](#)
- [getServerCertificate](#)
- [updateServerCertificate](#)

- [deleteServerCertificate](#)

Para obtener más información sobre los certificados de servidor en IAM, consulte [Uso de certificados de servidor](#) en la Guía del usuario de IAM.

Tareas previas necesarias

Para configurar y ejecutar este ejemplo, primero debe completar estas tareas:

- Instale Node.js. Para obtener más información acerca de la instalación de Node.js consulte el [sitio web de Node.js](#).
- Cree un archivo de configuraciones compartidas con sus credenciales de usuario. Para obtener más información acerca de cómo crear un archivo de credenciales compartidas, consulte [Carga de credenciales en Node.js desde el archivo de credenciales compartidas](#).

Generación de una lista de sus certificados de servidor

Cree un módulo de Node.js con el nombre de archivo `iam_listservercerts.js`. Asegúrese de configurar el SDK tal y como se ha indicado anteriormente. Para obtener acceso a IAM, cree un objeto de servicio `AWS.IAM`. Llame al método `listServerCertificates` del objeto de servicio de `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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos.

```
node iam_listservercerts.js
```

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Obtener un certificado de servidor

Cree un módulo de Node.js con el nombre de archivo `iam_getservercert.js`. Asegúrese de configurar el SDK tal y como se ha indicado anteriormente. Para obtener acceso a IAM, cree un objeto de servicio `AWS.IAM`. Cree un objeto JSON que contenga los parámetros necesarios para obtener un certificado, que consisten en el nombre del certificado de usuario que quiere. Llame al método `getServerCertificates` del objeto de servicio de `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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos.

```
node iam_getservercert.js
```

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Actualizar un certificado de servidor

Cree un módulo de Node.js con el nombre de archivo `iam_updateservercert.js`. Asegúrese de configurar el SDK tal y como se ha indicado anteriormente. Para obtener acceso a IAM, cree un

objeto de servicio `AWS.IAM`. Cree un objeto JSON que contenga los parámetros necesarios para actualizar un certificado, que consiste en el nombre del certificado de servidor ya existente, así como el nombre del nuevo certificado. Llame al método `updateServerCertificate` del objeto de servicio de `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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos.

```
node iam_updateservercert.js
```

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Eliminar un certificado de servidor

Cree un módulo de Node.js con el nombre de archivo `iam_deleteservercert.js`. Asegúrese de configurar el SDK tal y como se ha indicado anteriormente. Para obtener acceso a IAM, cree un objeto de servicio `AWS.IAM`. Cree un objeto JSON que contenga los parámetros necesarios para eliminar un certificado de servidor, que consisten en el nombre del certificado que quiere eliminar. Llame al método `deleteServerCertificates` del objeto de servicio de `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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos.

```
node iam_deleteservercert.js
```

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Administración de alias de cuenta de IAM



Este ejemplo de código de Node.js muestra:

- Cómo administrar alias para su ID de cuenta de AWS.

El escenario

Si quiere que la dirección URL de la página de inicio de sesión contenga el nombre de su empresa u otro identificador fácilmente reconocible en lugar de su ID de cuenta de AWS, puede crear un alias para el ID de cuenta de AWS. Si crea un alias de cuenta de AWS, la dirección URL de su página de inicio de sesión cambia para incorporar dicho alias.

En este ejemplo, se van a utilizar una serie de módulos de Node.js para crear y administrar alias de cuenta de IAM. Los módulos de Node.js usan el SDK para JavaScript para gestionar alias mediante los métodos de clase de cliente AWS . IAM siguientes:

- [createAccountAlias](#)
- [listAccountAliases](#)
- [deleteAccountAlias](#)

Para obtener más información acerca de los alias de cuenta de IAM, consulte [ID y alias de su cuenta de AWS](#) en la Guía del usuario de IAM.

Tareas previas necesarias

Para configurar y ejecutar este ejemplo, primero debe completar estas tareas:

- Instale Node.js. Para obtener más información acerca de la instalación de Node.js consulte el [sitio web de Node.js](#).
- Cree un archivo de configuraciones compartidas con sus credenciales de usuario. Para obtener más información acerca de cómo crear un archivo de credenciales compartidas, consulte [Carga de credenciales en Node.js desde el archivo de credenciales compartidas](#).

Creación de un alias de cuenta

Cree un módulo de Node.js con el nombre de archivo `iam_createaccountalias.js`. Asegúrese de configurar el SDK tal y como se ha indicado anteriormente. Para obtener acceso a IAM, cree un objeto de servicio AWS . IAM. Cree un objeto JSON que contenga los parámetros necesarios para crear un alias de cuenta y que incluya el alias que desea crear. Llame al método `createAccountAlias` del objeto de servicio de 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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos.

```
node iam_createaccountaliases.js ALIAS
```

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Mostrar alias de cuenta

Cree un módulo de Node.js con el nombre de archivo `iam_listaccountaliases.js`. Asegúrese de configurar el SDK tal y como se ha indicado anteriormente. Para obtener acceso a IAM, cree un objeto de servicio `AWS.IAM`. Cree un objeto JSON que contenga los parámetros necesarios para generar una lista de alias de la cuenta y que incluya el número máximo de elementos que se devolverán. Llame al método `listAccountAliases` del objeto de servicio de `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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos.

```
node iam_listaccountaliases.js
```

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Eliminación de un alias de la cuenta

Cree un módulo de Node.js con el nombre de archivo `iam_deleteaccountalias.js`. Asegúrese de configurar el SDK tal y como se ha indicado anteriormente. Para obtener acceso a IAM, cree un objeto de servicio `AWS.IAM`. Cree un objeto JSON que contenga los parámetros necesarios para eliminar un alias de la cuenta y que incluya el alias que desea eliminar. Llame al método `deleteAccountAlias` del objeto de servicio de `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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos.

```
node iam_deleteaccountalias.js ALIAS
```

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Ejemplo de Amazon Kinesis

Amazon Kinesis es una plataforma que sirve para transmitir en streaming datos en AWS. Ofrece servicios eficientes para la carga y el análisis de datos de streaming y proporciona la capacidad para crear aplicaciones de datos de streaming personalizadas para necesidades especiales.



La API de JavaScript para Kinesis se expone a través de la clase de cliente `AWS.Kinesis`. Para obtener más información acerca de cómo usar la clase de cliente de Kinesis, consulte [Class: AWS.Kinesis](#) en la referencia de la API.

Temas

- [Captura de la progresión del desplazamiento en una página web con Amazon Kinesis](#)

Captura de la progresión del desplazamiento en una página web con Amazon Kinesis



Este ejemplo de script de navegador muestra:

- Cómo capturar la progresión del desplazamiento en una página web con Amazon Kinesis como ejemplo de métricas de uso de la página de streaming para su análisis posterior.

El escenario

En este ejemplo, una sencilla página HTML simula el contenido de una página de blog. A medida que lector se desplaza por la entrada de blog simulada, el script de navegador usa el SDK para JavaScript para registrar la distancia de desplazamiento hacia abajo por la página y envía dichos datos a Kinesis mediante el método [putRecords](#) de la clase de cliente de Kinesis. A continuación, los datos de streaming que captura Amazon Kinesis Data Streams pueden procesarse mediante

instancias de Amazon EC2 y almacenarse en cualquiera de los diversos almacenes de datos existentes, incluidos Amazon DynamoDB y Amazon Redshift.

Tareas previas necesarias

Para configurar y ejecutar este ejemplo, primero debe completar estas tareas:

- Crear un flujo de Kinesis. Debe incluir el ARN de recurso de la transmisión en el script de navegador. Para obtener más información sobre la creación de Amazon Kinesis Data Streams, consulte la sección sobre [administración de secuencias de Kinesis](#) en la Guía para desarrolladores de Amazon Kinesis Data Streams.
- Crear un grupo de identidades de Amazon Cognito con el acceso habilitado para identidades sin autenticar. Debe incluir el ID del grupo de identidades en el código para obtener las credenciales para el script del navegador. Para obtener más información sobre los grupos de identidades de Amazon Cognito, consulte [Grupos de identidades](#) en la Guía para desarrolladores de Amazon Cognito.
- Crear un rol de IAM cuya política conceda permiso para enviar datos a una secuencia de Kinesis. Para obtener más información sobre cómo crear un rol de IAM, consulte [Creación de un rol para delegar permisos a un servicio de AWS](#) en la Guía del usuario de IAM.

Utilice la siguiente política de funciones al crear la función de IAM.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "mobileanalytics:PutEvents",
        "cognito-sync:*"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Effect": "Allow",
```

```
    "Action": [
      "kinesis:Put*"
    ],
    "Resource": [
      "arn:aws:kinesis:us-east-1:111122223333:stream/stream-name"
    ]
  }
]
}
```

La página de blog

El código HTML de la página del blog se compone principalmente de una serie de párrafos incluidos en un elemento `<div>`. La altura desplazable de este `<div>` se utiliza como ayuda para calcular hasta dónde se ha desplazado un lector por el contenido a medida que va leyendo. El HTML también contiene un par de elementos `<script>`. Uno de estos elementos añade el SDK para JavaScript a la página y el otro añade el script de navegador que captura la progresión del desplazamiento en la página y lo notifica a 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>
```

Configuración del SDK

Obtenga las credenciales necesarias para configurar el SDK llamando al método `CognitoIdentityCredentials`, indicando el ID de grupo de identidades de Amazon Cognito. Si todo funciona correctamente, cree el objeto de servicio de Kinesis en la función de devolución de llamada.

El siguiente fragmento de código muestra este paso. (Consulte [Captura del código de progresión del desplazamiento en la página web](#) para ver el ejemplo 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",
  });
```

Creación de registros de desplazamiento

La progresión del desplazamiento se calcula mediante las propiedades `scrollHeight` y `scrollTop` de `<div>` que tiene el contenido de la entrada de blog. Cada registro de desplazamiento se crea en una función de agente de escucha de eventos para el evento de `scroll` y luego se añade a una matriz de registros para su envío periódico a Kinesis.

El siguiente fragmento de código muestra este paso. (Consulte [Captura del código de progresión del desplazamiento en la página web](#) para ver el ejemplo 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);
});
```

Envío de registros a Kinesis

Una vez por segundo, si hay registros en la matriz, dichos registros pendientes se envían a Kinesis.

El siguiente fragmento de código muestra este paso. (Consulte [Captura del código de progresión del desplazamiento en la página web](#) para ver el ejemplo 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 del código de progresión del desplazamiento en la página web

A continuación, se muestra el código de script de navegador para el ejemplo de Kinesis capturando la progresión del desplazamiento por la página web.

```
// 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);
});
```

Ejemplos de Amazon S3

Amazon Simple Storage Service (Amazon S3) es un servicio web que proporciona almacenamiento en la nube con un alto grado de escalabilidad. Amazon S3 ofrece almacenamiento de objetos fácil de usar con una sencilla interfaz de servicios web que puede utilizarse para almacenar y recuperar la cantidad de datos que desee, cuando desee y desde cualquier lugar de la web.



La API de JavaScript para Amazon S3 se expone a través de la clase de cliente `AWS.S3`. Para obtener más información acerca de cómo usar la clase de cliente de Amazon S3, consulte [Class: AWS.S3](#) en la referencia de la API.

Temas

- [Ejemplos de Amazon S3 en un navegador](#)
- [Ejemplos de Node.js de Amazon S3](#)

Ejemplos de Amazon S3 en un navegador

Los siguientes temas muestran dos ejemplos de cómo se puede usar AWS SDK para JavaScript en el navegador para interactuar con buckets de Amazon S3.

- El primero muestra un escenario sencillo en el que cualquier usuario (sin autenticar) puede ver las fotos existentes en un bucket de Amazon S3.
- El segundo muestra un escenario más complejo en el que los usuarios pueden realizar operaciones en las fotos del bucket, como cargarlas, eliminarlas, etc.

Temas

- [Visualización de fotos en un bucket de Amazon S3 desde un navegador](#)
- [Carga de fotos en Amazon S3 desde un navegador](#)

Visualización de fotos en un bucket de Amazon S3 desde un navegador



Este ejemplo de código de script de navegador muestra:

- Cómo crear un álbum de fotos en un bucket de Amazon Simple Storage Service (Amazon S3) y permitir que los usuarios no autenticados vean las fotos.

El escenario

En este ejemplo, una página HTML sencilla proporciona una aplicación basada en navegador para ver las fotos de un álbum de fotos. El álbum de fotos se encuentra en un bucket de Amazon S3 en el que se cargan las fotos.



El script de navegador usa el SDK para JavaScript para interactuar con un bucket de Amazon S3. El script utiliza el método [listObjects](#) de la clase del cliente de Amazon S3 para poder ver los álbumes de fotos.

Tareas previas necesarias

Para configurar y ejecutar este ejemplo, primero debe completar las tareas siguientes.

Note

En este ejemplo, debe utilizar la misma región de AWS tanto para el bucket de Amazon S3 como para el grupo de identidades de Amazon Cognito.

Creación del bucket

En la [consola de Amazon S3](#), cree un bucket de Amazon S3 donde pueda almacenar álbumes y fotos. Para obtener más información acerca de cómo usar la consola para crear un bucket de S3, consulte la sección de [creación de un bucket](#) en la Guía del usuario de la consola de Amazon Simple Storage Service.

Cuando cree el bucket de S3, asegúrese de hacer lo siguiente:

- Anote el nombre del bucket para que pueda utilizarlo en una tarea posterior, Configuración de los permisos del rol.
- Elija una región de AWS en la que crear el bucket. Debe ser la misma región que utilizará para crear un grupo de identidades de Amazon Cognito en una tarea posterior necesaria, Creación de un grupo de identidades.
- Configure permisos de buckets siguiendo [Configuración de permisos para el acceso a sitios web](#) en la Guía del usuario de Amazon Simple Storage Service.

Creación de un grupo de identidades

En la [consola de Amazon Cognito](#), cree un grupo de identidades de Amazon Cognito, tal y como se describe en [the section called "Paso 1: Creación de un grupo de identidades en Amazon Cognito"](#) del tema Introducción a los scripts de navegador.

Según cree el grupo de identidades, anote el nombre del grupo de identidades, así como el nombre del rol para la identidad sin autenticar.

Configuración de los permisos del rol

Para permitir la visualización de álbumes y fotos, debe añadir permisos a un rol de IAM del grupo de identidades que acaba de crear. Comience creando una política tal y como se indica a continuación.

1. Abra la [consola de IAM](#).
2. En el panel de navegación de la izquierda, elija Políticas (Políticas) y, a continuación, elija el botón Create Policy (Crear política).
3. En la pestaña JSON, escriba la siguiente definición de JSON, pero sustituya BUCKET_NAME por el nombre del bucket.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:ListBucket"
      ],
      "Resource": [
        "arn:aws:s3:::BUCKET_NAME"
      ]
    }
  ]
}
```

4. Elija el botón Review policy (Revisar política), asigne un nombre a la política y proporcione una descripción (si lo desea); a continuación, seleccione el botón Create policy (Crear política).

Asegúrese de anotar el nombre para que pueda encontrarlo y asociarlo al rol de IAM más adelante.

Una vez que se ha creado la política, vuelva a la [consola de IAM](#). Busque el rol de IAM de la identidad sin autenticar que creó Amazon Cognito en la tarea previa necesaria, Creación de un grupo de identidades. Puede utilizar la política que acaba de crear para añadir permisos a esta identidad.

Aunque el flujo de trabajo de esta tarea normalmente es el mismo que el de [the section called “Paso 2: Añadir una política al rol de IAM creado”](#) del tema Introducción a los script de navegador, hay algunas diferencias que se deben tener en cuenta:

- Utilice la nueva política que acaba de crear, no una política de Amazon Polly.
- En la página Attach Permissions (Asociar permisos), para encontrar rápidamente la nueva política, abra la lista Filter policies (Filtrar políticas) y elija Customer managed (Administradas por el cliente).

Para obtener más información sobre cómo crear un rol de IAM, consulte [Creación de un rol para delegar permisos a un servicio de AWS](#) en la Guía del usuario de IAM.

Configuración de CORS

Para que el script de navegador pueda obtener acceso al bucket de Amazon S3, tiene que definir su [configuración de CORS](#) tal y como se indica a continuación.

Important

En la nueva consola S3, la configuración de CORS debe 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>
```

Creación de álbumes y carga de fotos

Como este ejemplo solo permite a los usuarios ver las fotos que ya están en el bucket, debe crear algunos álbumes en el bucket y cargar fotos en ellos.

Note

En este ejemplo, los nombres de archivo de los archivos de fotos deben comenzar por un guion bajo ("_"). Este carácter es importante para poder filtrar las fotos posteriormente. Además, asegúrese de respetar los derechos de autor de los propietarios de las fotos.

1. En la [consola de Amazon S3](#), abra el bucket que creó anteriormente.
2. En la pestaña Overview (Información general), eija el botón Create folder (Crear carpeta) para crear carpetas. En este ejemplo, llame a las carpetas "album1", "album2" y "album3".
3. Para album1 y después para album2, seleccione la carpeta y cargue fotos en ella como se indica a continuación:
 - a. Elija el botón Upload (Cargar).

- b. Arrastre o elija los archivos de foto que desea usar y después elija Next (Siguiente).
 - c. En Manage public permissions (Administrar permisos públicos), elija Grant public read access to this object(s) (Conceder acceso de lectura a este objeto).
 - d. Elija el botón Upload (Cargar) (en la esquina inferior izquierda).
4. Deje album3 vacío.

Definición de la página web

El código HTML de la aplicación para ver fotos consta de un elemento `<div>` en el que el script de navegador crea la interfaz de visualización. El primer elemento `<script>` añade el SDK al script de navegador. El segundo elemento `<script>` añade el archivo JavaScript externo que contiene el código de script de navegador.

En este ejemplo, el archivo se denomina `PhotoViewer.js` y se encuentra en la misma carpeta que el archivo HTML. Para encontrar el `SDK_VERSION_NUMBER` actual, consulte la referencia de API para el SDK para JavaScript en la [Guía de referencia de la API de AWS SDK para 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>
```

Configuración del SDK

Obtenga las credenciales que necesita para configurar el SDK llamando al método `CognitoIdentityCredentials`. Debe proporcionar ID del grupo de identidades de Amazon Cognito. A continuación, cree un objeto del servicio `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");
}
```

El resto del código de este ejemplo define las siguientes funciones para recopilar y presentar información sobre los álbumes y fotos del bucket.

- `listAlbums`
- `viewAlbum`

Listas de álbumes en el bucket

Para obtener una lista de todos los álbumes existentes en el bucket, la función `listAlbums` de la aplicación llama al método `listObjects` del objeto del servicio `AWS.S3`. La función utiliza la propiedad `CommonPrefixes` para que la llamada devuelva solo los objetos que se utilizan como álbumes (es decir, las carpetas).

El resto de la función toma la lista de álbumes del bucket de Amazon S3 y genera el código HTML necesario para mostrar la lista de álbumes en la página 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);
    }
  });
}
```

Visualización de un álbum

Para mostrar el contenido de un álbum en el bucket de Amazon S3, la función `viewAlbum` de la aplicación toma un nombre de álbum y crea la clave de Amazon S3 para dicho álbum. Luego la función llama al método `listObjects` del objeto del servicio AWS . S3 para obtener una lista de todos los objetos (las fotos) del álbum.

El resto de la función toma la lista de objetos del álbum y genera el código HTML necesario para mostrar las fotos en la página 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/>",
        '',
        "</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;");
});
}

```

Visualización de fotos de un bucket de Amazon S3: código completo

Esta sección contiene todo el código HTML y JavaScript del ejemplo que muestra cómo ver las fotos de un bucket de Amazon S3. Consulte la [sección principal](#) para obtener más información y conocer los requisitos previos.

El HTML del ejemplo:

```

<!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 ejemplo se puede encontrar [aquí en GitHub](#).

El código de script de navegador del ejemplo:

```
//
// 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/>",
        ' ',
        "</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 ejemplo se puede encontrar [aquí en GitHub](#).

Carga de fotos en Amazon S3 desde un navegador



Este ejemplo de código de script de navegador muestra:

- Cómo crear una aplicación de navegador que permita a los usuarios crear álbumes de fotos en un bucket de Amazon S3 y cargar fotos en los álbumes.

El escenario

En este ejemplo, una página HTML sencilla proporciona una aplicación basada en navegador para crear álbumes de fotos en un bucket de Amazon S3 en el que puede cargar fotos. La aplicación le permite eliminar fotos y álbumes que añade.



El script de navegador usa el SDK para JavaScript para interactuar con un bucket de Amazon S3. Utilice los siguientes métodos de la clase de cliente de Amazon S3 para habilitar la aplicación de álbum de fotos:

- [listObjects](#)
- [headObject](#)
- [putObject](#)
- [upload](#)
- [deleteObject](#)
- [deleteObjects](#)

Tareas previas necesarias

Para configurar y ejecutar este ejemplo, primero debe completar estas tareas:

- En la [consola de Amazon S3](#) cree un bucket de Amazon S3 que utilizará para almacenar las fotos en el álbum. Para obtener más información acerca de la creación de un bucket, consulte la sección de [creación de un bucket](#) en la Guía del usuario de Amazon Simple Storage Service. Asegúrese de que tiene permisos de lectura y escritura en Objects (Objetos). Para obtener más información acerca de los permisos en buckets, consulte [Configurar permisos para el acceso a sitios web](#).
- En la [consola de Amazon Cognito](#) cree un grupo de identidades de Amazon Cognito usando identidades federadas con acceso habilitado para usuarios no autenticados en la misma región que el bucket de Amazon S3. Debe incluir el ID del grupo de identidades en el código para obtener las credenciales para el script del navegador. Para obtener más información sobre las identidades federadas de Amazon Cognito, consulte [Grupos de identidades de Amazon Cognito \(identidades federadas\)](#) en la Guía para desarrolladores de Amazon Cognito.
- En la [consola de IAM](#) busque el rol de IAM creado por Amazon Cognito para los usuarios sin autenticar. Añada la política siguiente para conceder permisos de lectura y escritura a un bucket de Amazon S3. Para obtener más información sobre cómo crear un rol de IAM, consulte [Creación de un rol para delegar permisos a un servicio de AWS](#) en la Guía del usuario de IAM.

Utilice esta política de roles para el rol de IAM creado por Amazon Cognito para usuarios sin autenticar.

Warning

Si habilita el acceso a usuarios sin autenticar, concederá acceso de escritura al bucket y a todos los objetos del bucket a todas las personas del mundo. Este enfoque de seguridad es útil en este ejemplo para centrarnos en los principales objetivos del ejemplo. Sin embargo, en muchas situaciones de la vida real, es muy recomendable aplicar una seguridad más estricta, como usuarios autenticados y titularidad de los objetos.

JSON

```
{  
  "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/*"
    ]
  }
]
```

Configuración de CORS

Para que el script de navegador pueda obtener acceso al bucket de Amazon S3, primero tiene que configurar su [configuración de CORS](#) tal y como se indica a continuación.

Important

En la nueva consola S3, la configuración de CORS debe ser JSON.

JSON

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



```
    listAlbums();
  </script>
</head>
<body>
  <h1>My Photo Albums App</h1>
  <div id="app"></div>
</body>
</html>
```

Configuración del SDK

Obtenga las credenciales necesarias para configurar el SDK llamando al método `CognitoIdentityCredentials`, indicando el ID de grupo de identidades de Amazon Cognito. A continuación cree un objeto de servicio `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 },
});
```

Prácticamente el resto del código de este ejemplo está organizado en una serie de funciones que reúnen y presentan información sobre los álbumes en el bucket, cargan y muestran fotos cargadas en álbumes, y eliminan fotos y álbumes. Dichas funciones son las siguientes:

- `listAlbums`
- `createAlbum`
- `viewAlbum`
- `addPhoto`
- `deleteAlbum`

- deletePhoto

Listas de álbumes en el bucket

La aplicación crea álbumes en el bucket de Amazon S3 como objetos cuyas claves comienzan con una barra inclinada para indicar que el objeto funciona como una carpeta. Para obtener una lista de todos los álbumes existentes en el bucket, la función `listAlbums` de la aplicación llama al método `listObjects` del objeto de servicio `AWS.S3` al utilizar `commonPrefix` por lo que la llamada devuelve solo objetos que se utilizan como álbumes.

El resto de la función toma la lista de álbumes del bucket de Amazon S3 y genera el código HTML necesario para mostrar la lista de álbumes en la página web. También habilitar la eliminación y apertura de álbumes individuales.

```
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);
}
});
}
```

Creación de un álbum en el bucket

Para crear un álbum en el bucket de Amazon S3, la función `createAlbum` de la aplicación primero valida el nombre que se da al nuevo álbum para asegurarse de que los caracteres que contiene son adecuados. A continuación, la función crea una clave de objeto de Amazon S3 y la pasa al método `headObject` del objeto de servicio de Amazon S3. Este método devuelve los metadatos de la clave especificada, por lo que si se devuelven datos significa que ya existe un objeto con dicha clave.

Si el álbum no existe todavía, la función llama al método `putObject` del objeto de servicio `AWS.S3` para crear el álbum. A continuación llama a la función `viewAlbum` para mostrar el nuevo álbum vacío.

```
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);
  });
});
}
```

Visualización de un álbum

Para mostrar el contenido de un álbum en el bucket de Amazon S3, la función `viewAlbum` de la aplicación toma un nombre de álbum y crea la clave de Amazon S3 para dicho álbum. Luego la función llama al método `listObjects` del objeto de servicio `AWS.S3` para obtener una lista de todos los objetos (fotos) del álbum.

El resto de la función toma la lista de objetos (fotos) del álbum y genera el código HTML necesario para mostrar las fotos en la página web. También permite eliminar fotos individuales y volver a la lista de álbumes.

```
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>",
        '',
        "</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);
});
}

```

Añadir fotos a un álbum

Para cargar una foto a un álbum en el bucket de Amazon S3, la función `addPhoto` de la aplicación utiliza un elemento selector de archivos en la página web para identificar un archivo para cargarlo. A continuación forma una clave para la foto que se va a cargar a partir del nombre del álbum actual y el nombre del archivo.

La función llama al método `upload` del objeto de servicio de Amazon S3 para cargar la foto. Después de cargar la foto, la función vuelve a mostrar el álbum de modo que la foto cargada aparezca.

```

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);
  }
);
}
```

Eliminación de una foto

Para eliminar una foto de un álbum en el bucket de Amazon S3, la función `deletePhoto` de la aplicación llama al método `deleteObject` del objeto de servicio de Amazon S3. Esto elimina la foto que se ha especificado con el valor `photoKey` que se ha transferido a la función.

```
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);
  });
}
```

Eliminación de un álbum

Para eliminar un álbum en el bucket de Amazon S3, la función `deleteAlbum` de la aplicación llama al método `deleteObjects` del objeto de servicio de 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();
      }
    );
  });
}
```

Carga de fotos en Amazon S3: código completo

Esta sección contiene todo el código HTML y JavaScript del ejemplo de carga de fotos en un álbum de fotos de Amazon S3. Consulte la [sección principal](#) para obtener más información y conocer los requisitos previos.

El HTML del ejemplo:

```
<!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 ejemplo se puede encontrar [aquí en GitHub](#).

El código de script de navegador del ejemplo:

```
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>",
                '',
                "</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 ejemplo se puede encontrar [aquí en GitHub](#).

Ejemplos de Node.js de Amazon S3

Los siguientes temas muestran ejemplos de cómo se puede usar AWS SDK para JavaScript para interactuar con buckets de Amazon S3 mediante Node.js.

Temas

- [Creación y uso de buckets de Amazon S3](#)
- [Configuración de buckets de Amazon S3](#)
- [Administración de permisos de acceso a los buckets de Amazon S3](#)
- [Uso de políticas de bucket de Amazon S3](#)
- [Uso de un bucket de Amazon S3 como un host web estático](#)

Creación y uso de buckets de Amazon S3



Este ejemplo de código de Node.js muestra:

- Cómo obtener y mostrar una lista de buckets de Amazon S3 en su cuenta.
- Cómo crear un bucket de Amazon S3.
- Cómo cargar un objeto en un bucket especificado.

El escenario

En este ejemplo, se utiliza una serie de módulos de Node.js para obtener una lista de los buckets de Amazon S3 existentes, crear un bucket y cargar un archivo en un bucket especificado. Estos

módulos de Node.js utilizan el SDK para JavaScript para obtener información y cargar archivos en un bucket de Amazon S3 con estos métodos de la clase de cliente de Amazon S3:

- [listBuckets](#)
- [createBucket](#)
- [listObjects](#)
- [upload](#)
- [deleteBucket](#)

Tareas previas necesarias

Para configurar y ejecutar este ejemplo, primero debe completar estas tareas:

- Instale Node.js. Para obtener más información acerca de la instalación de Node.js consulte el [sitio web de Node.js](#).
- Cree un archivo de configuraciones compartidas con sus credenciales de usuario. Para obtener más información acerca de cómo crear un archivo de credenciales compartidas, consulte [Carga de credenciales en Node.js desde el archivo de credenciales compartidas](#).

Configuración del SDK

Configure el SDK para JavaScript creando un objeto de configuración global y luego configurando la región para su código. En este ejemplo, la región está establecida en `us-west-2`.

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

Visualización de una lista de buckets de Amazon S3

Cree un módulo de Node.js con el nombre de archivo `s3_listbuckets.js`. Configure el SDK como le hemos mostrado anteriormente. Para acceder a Amazon Simple Storage Service, cree un objeto de servicio de `AWS.S3`. Llame al método `listBuckets` del objeto de servicio de Amazon S3 para recuperar una lista de sus buckets. El parámetro `data` de la función de devolución de llamada tiene una propiedad `Buckets` que contiene una matriz de mapas para representar los buckets. Muestre la lista de buckets registrándola en la consola.

```
// 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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos.

```
node s3_listbuckets.js
```

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Creación del bucket de Amazon S3

Cree un módulo de Node.js con el nombre de archivo `s3_createbucket.js`. Configure el SDK como le hemos mostrado anteriormente. Cree un objeto de servicio `AWS.S3`. El módulo tomará un argumento de línea de comandos único para especificar un nombre para el nuevo bucket.

Añada una variable para almacenar los parámetros que se utilizan para llamar al método `createBucket` del objeto de servicio de Amazon S3, incluido el nombre del bucket que se acaba de crear. La función de devolución de llamada registra la ubicación del nuevo bucket después de que Amazon S3 lo cree correctamente.

```
// 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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos.

```
node s3_createbucket.js BUCKET_NAME
```

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Carga de un archivo en un bucket de Amazon S3

Cree un módulo de Node.js con el nombre de archivo `s3_upload.js`. Configure el SDK como le hemos mostrado anteriormente. Cree un objeto de servicio `AWS.S3`. El módulo ocupará dos argumentos de la línea de comandos, la primera para especificar el bucket de destino y la segunda para especificar el archivo que desea cargar.

Cree una variable con los parámetros necesarios para llamar al método `upload` del objeto de servicio de Amazon S3. Proporcione el nombre del bucket de destino en el parámetro de `Bucket`. El parámetro `Key` se establece en el nombre del archivo seleccionado, que puede obtener mediante el módulo `path` de Node.js. El parámetro `Body` se establece en el contenido del archivo, que puede obtener mediante `createReadStream` del módulo `fs` de 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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos.

```
node s3_upload.js BUCKET_NAME FILE_NAME
```

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Obtención de listas de objetos en un bucket de Amazon S3

Cree un módulo de Node.js con el nombre de archivo `s3_listobjects.js`. Configure el SDK como le hemos mostrado anteriormente. Cree un objeto de servicio `AWS.S3`.

Añada una variable para almacenar los parámetros que se utilizan para llamar al método `listObjects` del objeto de servicio de Amazon S3, incluido el nombre del bucket que se va a leer. La función de devolución de llamada registra una lista de objetos (archivos) o un mensaje de 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" });

// 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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos.

```
node s3_listobjects.js
```

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Eliminación de un bucket de Amazon S3

Cree un módulo de Node.js con el nombre de archivo `s3_deletebucket.js`. Configure el SDK como le hemos mostrado anteriormente. Cree un objeto de servicio `AWS.S3`.

Añada una variable para almacenar los parámetros que se utilizan para llamar al método `createBucket` del objeto de servicio de Amazon S3, incluido el nombre del bucket que se va a eliminar. El bucket tiene que estar vacío para eliminarlo. La función de devolución de llamada registra un mensaje de éxito o de 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" });

// 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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos.

```
node s3_deletebucket.js
```

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Configuración de buckets de Amazon S3



Este ejemplo de código de Node.js muestra:

- Cómo configurar los permisos de uso compartido de recursos entre orígenes (CORS) para un bucket.

El escenario

En este ejemplo, se usan una serie de módulos Node.js para generar una lista de buckets de Amazon S3 y para configurar el registro de buckets y CORS. Los módulos de Node.js usan el SDK para JavaScript para configurar un bucket de Amazon S3 seleccionado utilizando estos métodos de la clase de cliente de Amazon S3:

- [getBucketCors](#)
- [putBucketCors](#)

Para obtener más información sobre cómo usar la configuración de CORS con un bucket de Amazon S3, consulte [Uso compartido de recursos entre orígenes \(CORS\)](#) en la Guía del usuario de Amazon Simple Storage Service.

Tareas previas necesarias

Para configurar y ejecutar este ejemplo, primero debe completar estas tareas:

- Instale Node.js. Para obtener más información acerca de la instalación de Node.js consulte el [sitio web de Node.js](#).
- Cree un archivo de configuraciones compartidas con sus credenciales de usuario. Para obtener más información acerca de cómo crear un archivo de credenciales compartidas, consulte [Carga de credenciales en Node.js desde el archivo de credenciales compartidas](#).

Configuración del SDK

Configure el SDK para JavaScript creando un objeto de configuración global y luego configurando la región para su código. En este ejemplo, la región está establecida en `us-west-2`.

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

Recuperación de una configuración CORS de bucket

Cree un módulo de Node.js con el nombre de archivo `s3_getcors.js`. El módulo tomará un argumento de línea de comandos única para especificar el bucket cuya configuración de CORS desea. Configure el SDK como le hemos mostrado anteriormente. Cree un objeto de servicio `AWS.S3`.

El único parámetro que necesita pasar es el nombre del bucket seleccionado al llamar al método `getBucketCors`. Si el bucket tiene actualmente una configuración CORS, Amazon S3 devuelve dicha configuración como la propiedad `CORSRules` del parámetro `data` que se pasa a la función de devolución de llamada.

Si el bucket seleccionado no tiene configuración CORS, dicha información se devuelve a la función de devolución de llamada en el parámetro `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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos.

```
node s3_getcors.js BUCKET_NAME
```

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Configuración de CORS de bucket

Cree un módulo de Node.js con el nombre de archivo `s3_setcors.js`. El módulo acepta varios argumentos de línea de comandos, el primero de los cuales especifica el bucket cuya configuración CORS desea establecer. Los argumentos adicionales enumeran los métodos HTTP (POST, GET, PUT, PATCH, DELETE, POST) que desea permitir para el bucket. Configure el SDK como le hemos mostrado anteriormente.

Cree un objeto de servicio `AWS.S3`. Luego cree un objeto JSON para almacenar los valores para la configuración CORS según las necesidades del método `putBucketCors` del objeto de servicio `AWS.S3`. Especifique "Authorization" para el valor `AllowedHeaders` y "*" para el valor `AllowedOrigins`. Establezca el valor de `AllowedMethods` como matriz vacía inicialmente.

Especifique los métodos permitidos como parámetros de línea de comandos en el módulo Node.js, añadiendo cada uno de los métodos que coincidan con uno de los parámetros. Añada la configuración CORS obtenida a la matriz de configuraciones contenida en el parámetro `CORSRules`. Especifique el bucket que desea configurar para CORS en el parámetro `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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos que incluye uno o varios métodos HTTP, tal como se muestra.

```
node s3_setcors.js BUCKET_NAME get put
```

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Administración de permisos de acceso a los buckets de Amazon S3



Este ejemplo de código de Node.js muestra:

- Cómo recuperar o definir la lista de control de acceso de un bucket de Amazon S3.

El escenario

En este ejemplo, se utiliza un módulo Node.js para mostrar la lista de control de acceso (ACL) de un bucket seleccionado y aplicar cambios a la ACL de un bucket seleccionado. El módulo de Node.js usa el SDK para JavaScript para administrar permisos de acceso a buckets de Amazon S3 mediante estos métodos de la clase de cliente de Amazon S3:

- [getBucketAcl](#)
- [putBucketAcl](#)

Para obtener más información acerca de las listas de control de acceso de buckets de Amazon S3, consulte [Administración de acceso con ACL](#) en la Guía del usuario de Amazon Simple Storage Service.

Tareas previas necesarias

Para configurar y ejecutar este ejemplo, primero debe completar estas tareas:

- Instale Node.js. Para obtener más información acerca de la instalación de Node.js consulte el [sitio web de Node.js](#).
- Cree un archivo de configuraciones compartidas con sus credenciales de usuario. Para obtener más información acerca de cómo crear un archivo de credenciales compartidas, consulte [Carga de credenciales en Node.js desde el archivo de credenciales compartidas](#).

Configuración del SDK

Configure el SDK para JavaScript creando un objeto de configuración global y luego configurando la región para su código. En este ejemplo, la región está establecida en `us-west-2`.

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

Recuperación de la lista de control de acceso de un bucket actual

Cree un módulo de Node.js con el nombre de archivo `s3_getbucketacl.js`. El módulo tomará un argumento de línea de comandos único para especificar el bucket cuya configuración de ACL desea. Configure el SDK como le hemos mostrado anteriormente.

Cree un objeto de servicio AWS . S3. El único parámetro que necesita pasar es el nombre del bucket seleccionado al llamar al método `getBucketAcl`. Amazon S3 devuelve la configuración de la lista de control de acceso en el parámetro `data` que se pasa a la función de devolución de llamada.

```
// 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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos.

```
node s3_getbucketacl.js BUCKET_NAME
```

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Uso de políticas de bucket de Amazon S3



Este ejemplo de código de Node.js muestra:

- Cómo recuperar la política de buckets de un bucket de Amazon S3.
- Cómo añadir o actualizar la política de buckets de un bucket de Amazon S3.
- Cómo eliminar la política de buckets de un bucket de Amazon S3.

El escenario

En este ejemplo, se usan una serie de módulos de Node.js para recuperar, configurar o eliminar una política de bucket de un bucket de Amazon S3. Los módulos de Node.js usan el SDK para JavaScript para configurar la política de un bucket de Amazon S3 seleccionado utilizando estos métodos de la clase de cliente de Amazon S3:

- [getBucketPolicy](#)
- [putBucketPolicy](#)
- [deleteBucketPolicy](#)

Para obtener más información acerca de las políticas de bucket para Amazon S3, consulte [Uso de políticas de bucket y políticas de usuario](#) en la Guía del usuario de Amazon Simple Storage Service.

Tareas previas necesarias

Para configurar y ejecutar este ejemplo, primero debe completar estas tareas:

- Instale Node.js. Para obtener más información acerca de la instalación de Node.js consulte el [sitio web de Node.js](#).
- Cree un archivo de configuraciones compartidas con sus credenciales de usuario. Para obtener más información acerca de cómo crear un archivo de credenciales compartidas, consulte [Carga de credenciales en Node.js desde el archivo de credenciales compartidas](#).

Configuración del SDK

Configure el SDK para JavaScript creando un objeto de configuración global y luego configurando la región para su código. En este ejemplo, la región está establecida en `us-west-2`.

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

Recuperación de la política de buckets actual

Cree un módulo de Node.js con el nombre de archivo `s3_getbucketpolicy.js`. El módulo toma un argumento de línea de comandos único que especifica el bucket cuya política quiere. Configure el SDK como le hemos mostrado anteriormente.

Cree un objeto de servicio AWS . S3. El único parámetro que necesita pasar es el nombre del bucket seleccionado al llamar al método `getBucketPolicy`. Si el bucket tiene actualmente una política, Amazon S3 la devuelve en el parámetro `data` que se pasa a la función de devolución de llamada.

Si el bucket seleccionado no tiene política, dicha información se devuelve a la función de devolución de llamada en el parámetro `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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos.

```
node s3_getbucketpolicy.js BUCKET_NAME
```

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Configuración una política de bucket sencilla

Cree un módulo de Node.js con el nombre de archivo `s3_setbucketpolicy.js`. El módulo toma un argumento de línea de comandos único que especifica el bucket cuya política quiere aplicar.

Configure el SDK como le hemos mostrado anteriormente.

Cree un objeto de servicio AWS . S3. Las políticas de bucket se especifican en JSON. Primero cree un objeto JSON que contenga todos los valores para especificar la política, excepto el valor `Resource` que identifica el bucket.

Formatee la cadena `Resource` que la política exige e incorpore el nombre del bucket seleccionado. Inserte dicha cadena en el objeto JSON. Prepare los parámetros para el método `putBucketPolicy`, incluido el nombre del bucket y la política de JSON convertida en un valor de cadena.

```
// 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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos.

```
node s3_setbucketpolicy.js BUCKET_NAME
```

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Eliminación de una política de bucket

Cree un módulo de Node.js con el nombre de archivo `s3_deletebucketpolicy.js`. El módulo toma un argumento de línea de comandos único que especifica el bucket cuya política quiere eliminar. Configure el SDK como le hemos mostrado anteriormente.

Cree un objeto de servicio AWS.S3. El único parámetro que necesita pasar al llamar al método `deleteBucketPolicy` es el nombre del bucket seleccionado.

```
// 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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos.

```
node s3_deletebucketpolicy.js BUCKET_NAME
```

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Uso de un bucket de Amazon S3 como un host web estático



Este ejemplo de código de Node.js muestra:

- Cómo configurar un bucket de Amazon S3 como host web estático.

El escenario

En este ejemplo, se utilizan una serie de módulos Node.js para configurar cualquiera de los buckets para que funcione como host web estático. Los módulos de Node.js usan el SDK para JavaScript para configurar un bucket de Amazon S3 seleccionado utilizando estos métodos de la clase de cliente de Amazon S3:

- [getBucketWebsite](#)
- [putBucketWebsite](#)
- [deleteBucketWebsite](#)

Para obtener más información acerca de cómo utilizar un bucket de Amazon S3 como un host web estático, consulte [Alojamiento de un sitio web estático en Amazon S3](#) en la Guía del usuario de Amazon Simple Storage Service.

Tareas previas necesarias

Para configurar y ejecutar este ejemplo, primero debe completar estas tareas:

- Instale Node.js. Para obtener más información acerca de la instalación de Node.js consulte el [sitio web de Node.js](#).
- Cree un archivo de configuraciones compartidas con sus credenciales de usuario. Para obtener más información acerca de cómo crear un archivo de credenciales compartidas, consulte [Carga de credenciales en Node.js desde el archivo de credenciales compartidas](#).

Configuración del SDK

Configure el SDK para JavaScript creando un objeto de configuración global y luego configurando la región para su código. En este ejemplo, la región está establecida en `us-west-2`.

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

Recuperación de la configuración de sitio web del bucket actual

Cree un módulo de Node.js con el nombre de archivo `s3_getbucketwebsite.js`. El módulo toma un argumento de línea de comandos único que especifica el bucket cuya configuración de sitio web quiere. Configure el SDK como le hemos mostrado anteriormente.

Cree un objeto de servicio `AWS.S3`. Cree una función que recupere la configuración de sitio web de bucket actual para el bucket seleccionado en la lista de buckets. El único parámetro que necesita pasar es el nombre del bucket seleccionado al llamar al método `getBucketWebsite`. Si el bucket tiene actualmente una configuración de sitio web, Amazon S3 devuelve dicha configuración en el parámetro `data` que se pasa a la función de devolución de llamada.

Si el bucket seleccionado no tiene configuración de sitio web, dicha información se devuelve a la función de devolución de llamada en el parámetro `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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos.

```
node s3_getbucketwebsite.js BUCKET_NAME
```

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Establecimiento de la configuración de sitio web de un bucket

Cree un módulo de Node.js con el nombre de archivo `s3_setbucketwebsite.js`. Configure el SDK como le hemos mostrado anteriormente. Cree un objeto de servicio AWS . S3.

Cree una función que aplica una configuración de sitio web de bucket. La configuración permite que el bucket seleccionado sirva como host web estático. Las configuraciones de sitio web se especifican en JSON. Primero cree un objeto JSON que contenga todos los valores para especificar la configuración del sitio web, excepto el valor `Key` que identifica el documento de error y el valor `Suffix` que identifica el documento de índice.

Inserte los valores de los elementos de entrada de texto en el objeto JSON. Prepare los parámetros para el método `putBucketWebsite`, incluido el nombre del bucket y la configuración de sitio web 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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos.

```
node s3_setbucketwebsite.js BUCKET_NAME INDEX_PAGE ERROR_PAGE
```

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Eliminación de la configuración de sitio web de un bucket

Cree un módulo de Node.js con el nombre de archivo `s3_deletebucketwebsite.js`. Configure el SDK como le hemos mostrado anteriormente. Cree un objeto de servicio `AWS.S3`.

Cree una función que elimine la configuración de sitio web del bucket seleccionado. El único parámetro que necesita pasar al llamar al método `deleteBucketWebsite` es el nombre del bucket seleccionado.

```
// 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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos.

```
node s3_deletebucketwebsite.js BUCKET_NAME
```

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Ejemplos de Amazon Simple Email Service

Amazon Simple Email Service (Amazon SES) es un servicio de envío de correos electrónicos basado en la nube y diseñado para ayudar a los responsables de marketing digital y a los desarrolladores de aplicaciones a enviar correos electrónicos de marketing, notificaciones y transacciones. Se trata de un servicio de confianza y rentable para negocios de todos los tamaños que usan el correo electrónico para mantenerse en contacto con sus clientes.



La API de JavaScript para Amazon SES se expone a través de la clase de cliente `AWS.SES`. Para obtener más información acerca de cómo usar la clase de cliente de Amazon SES, consulte [Class: AWS.SES](#) en la referencia de la API.

Temas

- [Administración de identidades en Amazon SES](#)
- [Uso de plantillas de correo electrónico en Amazon SES](#)
- [Envío de correo electrónico con Amazon SES](#)
- [Uso de filtros de direcciones IP para recibir correos electrónicos en Amazon SES](#)
- [Uso de reglas de recepción en Amazon SES](#)

Administración de identidades en Amazon SES



Este ejemplo de código de Node.js muestra:

- Cómo verificar las direcciones de correo electrónico y los dominios que se usan con Amazon SES.
- Cómo asignar una política de IAM a sus identidades de Amazon SES.
- Cómo generar una lista de todas las identidades de Amazon SES para su cuenta de AWS.
- Cómo eliminar identidades que se usan con Amazon SES.

En Amazon SES, una identidad es una dirección de correo electrónico o dominio que se utiliza para enviar correos electrónicos. Amazon SES requiere que verifique su dirección de correo electrónico o dominio para confirmar que es de su propiedad e impedir que otras personas los utilicen.

Para obtener información detallada acerca de cómo verificar direcciones de correo electrónico y dominios en Amazon SES, consulte [Verificación de direcciones de correo electrónico y dominios en Amazon SES](#) en la Guía para desarrolladores de Amazon Simple Email Service. Para obtener información acerca de cómo enviar autorizaciones en Amazon SES, consulte [Información general sobre el envío de autorizaciones en Amazon SES](#).

El escenario

En este ejemplo, va a utilizar una serie de módulos de Node.js para verificar y administrar identidades de Amazon SES. Los módulos de Node.js utilizan el SDK para JavaScript para verificar las direcciones de correo electrónico y los dominios, por medio de los métodos siguientes de la clase de cliente de Amazon SES:

- [listIdentities](#)
- [deleteIdentity](#)
- [verifyEmailIdentity](#)
- [verifyDomainIdentity](#)

Tareas previas necesarias

Para configurar y ejecutar este ejemplo, primero debe completar estas tareas:

- Instale Node.js. Para obtener más información acerca de la instalación de Node.js consulte el [sitio web de Node.js](#).
- Cree un archivo de configuraciones compartidas con sus credenciales de usuario. Para obtener más información acerca de cómo proporcionar un archivo JSON de credenciales, consulte [Carga de credenciales en Node.js desde el archivo de credenciales compartidas](#).

Configuración del SDK

Configure el SDK para JavaScript creando un objeto de configuración global y luego configurando la región para su código. En este ejemplo, la región está establecida en us-west-2.

```
// Load the SDK for JavaScript
var AWS = require('aws-sdk');

// Set the Region
AWS.config.update({region: 'us-west-2'});
```

Generación de un listado de entidades

En este ejemplo, utilice un módulo de Node.js para obtener una lista de direcciones de correo electrónico y dominios que se usarán con Amazon SES. Cree un módulo de Node.js con el nombre de archivo `ses_listidentities.js`. Configure el SDK como le hemos mostrado anteriormente.

Cree un objeto para transferir el parámetro `IdentityType` y otros parámetros para el método `listIdentities` de la clase de cliente de Amazon SES. Para llamar al método `listIdentities`, cree una promesa para invocar un objeto de servicio de Amazon SES transfiriendo el objeto de parámetros.

Luego gestione la response en la devolución de llamada de la promesa. Los data que la promesa devuelve contienen una matriz de identidades de dominio tal y como se especifica con el parámetro `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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos.

```
node ses_listidentities.js
```

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Verificación de una identidad de dirección de correo electrónico

En este ejemplo, utilice un módulo de Node.js para comprobar los remitentes de direcciones de correo electrónico que se usarán con Amazon SES. Cree un módulo de Node.js con el nombre de archivo `ses_verifyemailidentity.js`. Configure el SDK como le hemos mostrado anteriormente. Para obtener acceso a Amazon SES, cree un objeto de servicio `AWS.SES`.

Cree un objeto para transferir el parámetro `EmailAddress` para el método `verifyEmailIdentity` de la clase de cliente de Amazon SES. Para llamar al método `verifyEmailIdentity`, cree una promesa para invocar un objeto de servicio de Amazon SES transfiriendo los parámetros. Luego gestione la `response` en la devolución de llamada de la promesa.

```
// 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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos. El dominio se añade a Amazon SES para que se compruebe.

```
node ses_verifyemailidentity.js
```

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Comprobación de una identidad de dominio

En este ejemplo, utilice un módulo de Node.js para comprobar los dominios de correo electrónico que se usarán con Amazon SES. Cree un módulo de Node.js con el nombre de archivo `ses_verifydomainidentity.js`. Configure el SDK como le hemos mostrado anteriormente.

Cree un objeto para transferir el parámetro `Domain` para el método `verifyDomainIdentity` de la clase de cliente de Amazon SES. Para llamar al método `verifyDomainIdentity`, cree una

promesa para invocar un objeto de servicio de Amazon SES transfiriendo el objeto de parámetros. Luego gestione la response en la devolución de llamada de la promesa.

```
// 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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos. El dominio se añade a Amazon SES para que se compruebe.

```
node ses_verifydomainidentity.js
```

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Eliminación de identidades

En este ejemplo, utilice un módulo de Node.js para eliminar direcciones de correo electrónico o dominios que se usan con Amazon SES. Cree un módulo de Node.js con el nombre de archivo `ses_deleteidentity.js`. Configure el SDK como le hemos mostrado anteriormente.

Cree un objeto para transferir el parámetro `Identity` para el método `deleteIdentity` de la clase de cliente de Amazon SES. Para llamar al método `deleteIdentity`, cree una `request` para invocar un objeto de servicio de Amazon SES transfiriendo los parámetros. Luego gestione la response en la devolución de llamada de la promesa.

```
// 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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos.

```
node ses_deleteidentity.js
```

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Uso de plantillas de correo electrónico en Amazon SES



Este ejemplo de código de Node.js muestra:

- Obtener una lista de todas sus plantillas de correo electrónico.
- Recuperar y actualizar plantillas de correo electrónico.
- Crear y eliminar plantillas de correo electrónico.

Con Amazon SES, puede enviar mensajes de correo electrónico personalizados mediante plantillas de correo electrónico. Para obtener más información sobre cómo crear y usar plantillas de correo electrónico en Amazon Simple Email Service, consulte [Envío de correos electrónicos personalizados utilizando la API de Amazon SES](#) en la Guía para desarrolladores de Amazon Simple Email Service.

El escenario

En este ejemplo, va a utilizar una serie de módulos de Node.js para trabajar con plantillas de correo electrónico. Los módulos de Node.js utilizan el SDK para JavaScript para crear y usar plantillas de correo electrónico mediante los métodos siguientes de la clase de cliente de Amazon SES:

- [listTemplates](#)
- [createTemplate](#)
- [getTemplate](#)
- [deleteTemplate](#)
- [updateTemplate](#)

Tareas previas necesarias

Para configurar y ejecutar este ejemplo, primero debe completar estas tareas:

- Instale Node.js. Para obtener más información acerca de la instalación de Node.js consulte el [sitio web de Node.js](#).
- Cree un archivo de configuraciones compartidas con sus credenciales de usuario. Para obtener más información acerca de cómo crear un archivo de credenciales, consulte [Carga de credenciales en Node.js desde el archivo de credenciales compartidas](#).

Generación de listas de plantillas de correo electrónico

En este ejemplo, utilice un módulo de Node.js para crear una plantilla de correo electrónico que se usará con Amazon SES. Cree un módulo de Node.js con el nombre de archivo `ses_listtemplates.js`. Configure el SDK como le hemos mostrado anteriormente.

Cree un objeto para transferir los parámetros para el método `listTemplates` de la clase de cliente de Amazon SES. Para llamar al método `listTemplates`, cree una promesa para invocar un objeto de servicio de Amazon SES transfiriendo los parámetros. Luego gestione la `response` en la devolución de llamada de la promesa.

```
// 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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos. Amazon SES devuelve la lista de plantillas.

```
node ses_listtemplates.js
```

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Obtención de una plantilla de correo electrónico

En este ejemplo, utilice un módulo de Node.js para obtener una plantilla de correo electrónico que se usará con Amazon SES. Cree un módulo de Node.js con el nombre de archivo `ses_gettemplate.js`. Configure el SDK como le hemos mostrado anteriormente.

Cree un objeto para transferir el parámetro `TemplateName` para el método `getTemplate` de la clase de cliente de Amazon SES. Para llamar al método `getTemplate`, cree una promesa para invocar un objeto de servicio de Amazon SES transfiriendo los parámetros. Luego gestione la `response` en la devolución de llamada de la promesa.

```
// 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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos. Amazon SES devuelve los detalles de la plantilla.

```
node ses_gettemplate.js
```

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Creación de una plantilla de correo electrónico

En este ejemplo, utilice un módulo de Node.js para crear una plantilla de correo electrónico que se usará con Amazon SES. Cree un módulo de Node.js con el nombre de archivo `ses_createtemplate.js`. Configure el SDK como le hemos mostrado anteriormente.

Cree un objeto para transferir los parámetros para el método `createTemplate` de la clase de cliente de Amazon SES, incluidos `TemplateName`, `HtmlPart`, `SubjectPart` y `TextPart`. Para llamar al método `createTemplate`, cree una promesa para invocar un objeto de servicio de Amazon SES transfiriendo los parámetros. Luego gestione la respuesta en la devolución de llamada de la promesa.

```
// 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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos. La plantilla se añade a Amazon SES.

```
node ses_createtemplate.js
```

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Actualización de una plantilla de correo electrónico

En este ejemplo, utilice un módulo de Node.js para crear una plantilla de correo electrónico que se usará con Amazon SES. Cree un módulo de Node.js con el nombre de archivo `ses_updatetemplate.js`. Configure el SDK como le hemos mostrado anteriormente.

Cree un objeto para transferir los valores de parámetro `Template` que desea actualizar en la plantilla, con el parámetro `TemplateName` obligatorio transferido al método `updateTemplate` de la clase de cliente de Amazon SES. Para llamar al método `updateTemplate`, cree una promesa para invocar un objeto de servicio de Amazon SES transfiriendo los parámetros. Luego gestione la `response` en la devolución de llamada de la promesa.

```
// 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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos. Amazon SES devuelve los detalles de la plantilla.

```
node ses_updatetemplate.js
```

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Eliminación de una plantilla de correo electrónico

En este ejemplo, utilice un módulo de Node.js para crear una plantilla de correo electrónico que se usará con Amazon SES. Cree un módulo de Node.js con el nombre de archivo `ses_deletetemplate.js`. Configure el SDK como le hemos mostrado anteriormente.

Cree un objeto para transferir el parámetro `TemplateName` obligatorio al método `deleteTemplate` de la clase de cliente de Amazon SES. Para llamar al método `deleteTemplate`, cree una promesa para invocar un objeto de servicio de Amazon SES transfiriendo los parámetros. Luego gestione la response en la devolución de llamada de la promesa.

```
// 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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos. Amazon SES devuelve los detalles de la plantilla.

```
node ses_deletetemplate.js
```

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Envío de correo electrónico con Amazon SES



Este ejemplo de código de Node.js muestra:

- Enviar un correo electrónico en HTML o de texto.
- Enviar mensajes de correo electrónico basados en una plantilla de correo electrónico.
- Enviar mensajes de correo electrónico masivos basados en una plantilla de correo electrónico.

La API de Amazon SES ofrece dos formas distintas de enviar los correos electrónicos, según el nivel de control que desee tener sobre la composición del mensaje de correo electrónico: con o sin formato. Para obtener información detallada, consulte [Envío de correo electrónico con formato](#)

[utilizando la API de Amazon SES](#) y [Envío de correo electrónico sin formato usando la API de Amazon SES](#).

El escenario

En este ejemplo, va a utilizar una serie de módulos de Node.js para enviar correos electrónicos de distintas maneras. Los módulos de Node.js utilizan el SDK para JavaScript para crear y usar plantillas de correo electrónico mediante los métodos siguientes de la clase de cliente de Amazon SES:

- [sendEmail](#)
- [sendTemplatedEmail](#)
- [sendBulkTemplatedEmail](#)

Tareas previas necesarias

- Instale Node.js. Para obtener más información acerca de la instalación de Node.js consulte el [sitio web de Node.js](#).
- Cree un archivo de configuraciones compartidas con sus credenciales de usuario. Para obtener más información acerca de cómo proporcionar un archivo JSON de credenciales, consulte [Carga de credenciales en Node.js desde el archivo de credenciales compartidas](#).

Requisitos para enviar un mensaje de correo electrónico

Amazon SES crea un mensaje de correo electrónico e inmediatamente lo pone en la cola para su envío. Para enviar correo electrónico por medio del método `SES.sendEmail`, el mensaje debe cumplir los requisitos siguientes:

- Tiene que enviar el mensaje desde un dominio o una dirección de correo electrónico que se haya verificado. Si intenta enviar un correo electrónico mediante una dirección o un dominio que no se haya verificado, la operación generará un error "Email address not verified".
- Si su cuenta está todavía en el entorno de pruebas de Amazon SES, solo podrá realizar envíos a direcciones o dominios verificados, o a direcciones de correo electrónico que estén asociadas al simulador de bandeja de correo de Amazon SES. Para obtener más información, consulte [Verificación de direcciones de correo electrónico y dominios](#) en la Guía para desarrolladores de Amazon Simple Email Service.
- El tamaño total del mensaje, los archivos adjuntos incluidos, debe ser inferior a 10 MB.

- El mensaje tiene que incluir al menos una dirección de correo electrónico de destinatario. La dirección del destinatario puede ser una dirección A:, una dirección CC: o una dirección CCO. Si la dirección de correo electrónico de un destinatario no es válida (es decir, no sigue el formato `UserName@[SubDomain.]Domain.TopLevelDomain`), se rechazará todo el mensaje, aunque este contenga otros destinatarios que sí son válidos.
- El mensaje no puede incluir más de 50 destinatarios entre los campos A:, CC: y CCO: en total. Si necesita enviar un mensaje de correo electrónico a más destinatarios, puede dividir la lista de destinatarios en grupos de 50 o menos y luego llamar al método `sendEmail` varias veces para enviar el mensaje a cada grupo.

Envío de un correo electrónico

En este ejemplo, utilice un módulo de Node.js para enviar un correo electrónico con Amazon SES. Cree un módulo de Node.js con el nombre de archivo `ses_sendemail.js`. Configure el SDK como le hemos mostrado anteriormente.

Cree un objeto para transferir los valores de parámetros que definen el correo electrónico que se va a enviar, como las direcciones del remitente y el receptor, el cuerpo del correo electrónico en formatos de texto y HTML, al método `sendEmail` de la clase de cliente de Amazon SES. Para llamar al método `sendEmail`, cree una promesa para invocar un objeto de servicio de Amazon SES transfiriendo los parámetros. Luego gestione la respuesta en la devolución de llamada de la promesa.

```
// 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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos. El correo electrónico se pone en cola para que Amazon SES lo envíe.

```
node ses_sendemail.js
```

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Envío de un correo electrónico mediante una plantilla

En este ejemplo, utilice un módulo de Node.js para enviar un correo electrónico con Amazon SES. Cree un módulo de Node.js con el nombre de archivo `ses_sendtemplatedemail.js`. Configure el SDK como le hemos mostrado anteriormente.

Cree un objeto para transferir los valores de parámetros que definen el correo electrónico que se va a enviar, como las direcciones del remitente y el receptor, el cuerpo del correo electrónico en formatos de texto y HTML, al método `sendTemplatedEmail` de la clase de cliente de Amazon SES. Para llamar al método `sendTemplatedEmail`, cree una promesa para invocar un objeto de servicio de Amazon SES transfiriendo los parámetros. Luego gestione la `response` en la devolución de llamada de la promesa.

```
// 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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos. El correo electrónico se pone en cola para que Amazon SES lo envíe.

```
node ses_sendtemplatedemail.js
```

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Envío masivo de correos electrónicos mediante una plantilla

En este ejemplo, utilice un módulo de Node.js para enviar un correo electrónico con Amazon SES. Cree un módulo de Node.js con el nombre de archivo `ses_sendbulktemplatedemail.js`. Configure el SDK como le hemos mostrado anteriormente.

Cree un objeto para transferir los valores de parámetros que definen el correo electrónico que se va a enviar, como las direcciones del remitente y el receptor, el cuerpo del correo electrónico en formatos de texto y HTML, al método `sendBulkTemplatedEmail` de la clase de cliente de Amazon SES. Para llamar al método `sendBulkTemplatedEmail`, cree una promesa para invocar un objeto de servicio de Amazon SES transfiriendo los parámetros. Luego gestione la `response` en la devolución de llamada de la promesa.

```
// 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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos. El correo electrónico se pone en cola para que Amazon SES lo envíe.

```
node ses_sendbulktemplatedemail.js
```

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Uso de filtros de direcciones IP para recibir correos electrónicos en Amazon SES



Este ejemplo de código de Node.js muestra:

- Crear filtros de direcciones IP para aceptar o rechazar correo que proceda de una dirección IP o de un rango de direcciones IP.
- Generar una lista de filtros de direcciones IP actuales.
- Eliminar un filtro de direcciones IP.

En Amazon SES, un filtro es una estructura de datos que contiene un nombre y un intervalo de direcciones IP y que indica si se permite o se bloquea correo electrónico procedente de dicho rango. Las direcciones IP que quiere bloquear o permitir se indican por medio de una única dirección IP o un intervalo de direcciones IP en notación de enrutamiento entre dominios sin clases (CIDR). Para obtener más información sobre cómo Amazon SES recibe el correo electrónico, consulte [Conceptos de recepción de correo electrónico de Amazon SES](#) en la Guía para desarrolladores de Amazon Simple Email Service.

El escenario

En este ejemplo, se usan una serie de módulos de Node.js para enviar correos electrónicos de distintas maneras. Los módulos de Node.js utilizan el SDK para JavaScript para crear y usar plantillas de correo electrónico mediante los métodos siguientes de la clase de cliente de Amazon SES:

- [createReceiptFilter](#)
- [listReceiptFilters](#)
- [deleteReceiptFilter](#)

Tareas previas necesarias

Para configurar y ejecutar este ejemplo, primero debe completar estas tareas:

- Instale Node.js. Para obtener más información acerca de la instalación de Node.js consulte el [sitio web de Node.js](#).
- Cree un archivo de configuraciones compartidas con sus credenciales de usuario. Para obtener más información acerca de cómo crear un archivo de credenciales compartidas, consulte [Carga de credenciales en Node.js desde el archivo de credenciales compartidas](#).

Configuración del SDK

Configure el SDK para JavaScript creando un objeto de configuración global y luego configurando la región para su código. En este ejemplo, la región está establecida en `us-west-2`.

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

Creación de un filtro de direcciones IP

En este ejemplo, utilice un módulo de Node.js para enviar un correo electrónico con Amazon SES. Cree un módulo de Node.js con el nombre de archivo `ses_createreceiptfilter.js`. Configure el SDK como le hemos mostrado anteriormente.

Cree un objeto para transferir los valores de los parámetros que definen el filtro de IP, como el nombre del filtro, una dirección IP o un intervalo de direcciones que deben filtrarse y si se permitirá o se bloqueará el tráfico de correo electrónico de las direcciones filtradas. Para llamar al método `createReceiptFilter`, cree una promesa para invocar un objeto de servicio de Amazon SES transfiriendo los parámetros. Luego gestione la `response` en la devolución de llamada de la promesa.

```
// 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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos. El filtro se crea en Amazon SES.

```
node ses_createreceiptfilter.js
```

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Generación de una lista de sus filtros de direcciones IP

En este ejemplo, utilice un módulo de Node.js para enviar un correo electrónico con Amazon SES. Cree un módulo de Node.js con el nombre de archivo `ses_listreceiptfilters.js`. Configure el SDK como le hemos mostrado anteriormente.

Cree un objeto de parámetros vacío. Para llamar al método `listReceiptFilters`, cree una promesa para invocar un objeto de servicio de Amazon SES transfiriendo los parámetros. Luego gestione la `response` en la devolución de llamada de la promesa.

```
// 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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos. Amazon SES devuelve la lista de filtros.

```
node ses_listreceiptfilters.js
```

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Eliminación de un filtro de direcciones IP

En este ejemplo, utilice un módulo de Node.js para enviar un correo electrónico con Amazon SES. Cree un módulo de Node.js con el nombre de archivo `ses_deletereciptfilter.js`. Configure el SDK como le hemos mostrado anteriormente.

Cree un objeto para transferir el nombre del filtro IP que desea eliminar. Para llamar al método `deleteReceiptFilter`, cree una promesa para invocar un objeto de servicio de Amazon SES transfiriendo los parámetros. Luego gestione la `response` en la devolución de llamada de la promesa.

```
// 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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos. El filtro se elimina de Amazon SES.

```
node ses_deletereciptfilter.js
```

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Uso de reglas de recepción en Amazon SES



Este ejemplo de código de Node.js muestra:

- Crear y eliminar reglas de recepción.
- Organizar reglas de recepción en conjuntos de reglas de recepción.

Las reglas de recepción de Amazon SES especifican qué debe hacerse con el correo electrónico que recibe de las direcciones de correo electrónico o los dominios de su propiedad. Una regla de recepción contiene una condición y una lista de acciones que siguen un orden. Si el destinatario de un correo electrónico entrante coincide con un destinatario que se ha especificado en las condiciones de la regla de recepción, Amazon SES ejecutará las acciones indicadas en dicha regla de recepción.

Para utilizar Amazon SES como receptor de correo electrónico, debe tener al menos un conjunto de reglas de recepción activo. Un conjunto de reglas de recepción es una colección ordenada de reglas de recepción que especifican cómo debe proceder Amazon SES con los correos electrónicos que recibe a través de dominios verificados. Para obtener más información, consulte [Creación de reglas de recepción para recepción de correos electrónicos de Amazon SES](#) y [Creación de un](#)

[conjunto de reglas de recepción para recibir correos electrónicos de Amazon SES](#) en la Guía para desarrolladores de Amazon Simple Email Service.

El escenario

En este ejemplo, se usan una serie de módulos de Node.js para enviar correos electrónicos de distintas maneras. Los módulos de Node.js utilizan el SDK para JavaScript para crear y usar plantillas de correo electrónico mediante los métodos siguientes de la clase de cliente de Amazon SES:

- [createReceiptRule](#)
- [deleteReceiptRule](#)
- [createReceiptRuleSet](#)
- [deleteReceiptRuleSet](#)

Tareas previas necesarias

Para configurar y ejecutar este ejemplo, primero debe completar estas tareas:

- Instale Node.js. Para obtener más información acerca de la instalación de Node.js consulte el [sitio web de Node.js](#).
- Cree un archivo de configuraciones compartidas con sus credenciales de usuario. Para obtener más información acerca de cómo proporcionar un archivo JSON de credenciales, consulte [Carga de credenciales en Node.js desde el archivo de credenciales compartidas](#).

Creación de una regla de recepción de Amazon S3

Cada regla de recepción de Amazon SES contiene una lista ordenada de acciones. En este ejemplo se crea una regla de recepción con una acción de Amazon S3, que entrega el mensaje de correo electrónico a un bucket de Amazon S3. Para obtener más información sobre las acciones de las reglas de recepción, consulte [Opciones de acción](#) en la Guía para desarrolladores de Amazon Simple Email Service.

Para que Amazon SES escriba correo electrónico en un bucket de Amazon S3, cree una política de bucket que conceda permiso PutObject a Amazon SES. Para obtener información sobre la creación de esta política de bucket, consulte [Conceder permiso a Amazon SES para escribir en su bucket de Amazon S3](#) en la Guía para desarrolladores de Amazon Simple Email Service.

En este ejemplo, utilice un módulo de Node.js para crear una regla de recepción en Amazon SES y guardar los mensajes recibidos en un bucket de Amazon S3. Cree un módulo de Node.js con el nombre de archivo `ses_createreceiptrule.js`. Configure el SDK como le hemos mostrado anteriormente.

Crear un objeto de parámetros para transferir los valores necesarios para crear un conjunto de reglas de recepción. Para llamar al método `createReceiptRuleSet`, cree una promesa para invocar un objeto de servicio de Amazon SES transfiriendo los parámetros. Luego gestione la `response` en la devolución de llamada de la promesa.

```
// 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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos. Amazon SES crea la regla de recepción.

```
node ses_createreciptrule.js
```

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Eliminación de una regla de recepción

En este ejemplo, utilice un módulo de Node.js para enviar un correo electrónico con Amazon SES. Cree un módulo de Node.js con el nombre de archivo `ses_deletereciptrule.js`. Configure el SDK como le hemos mostrado anteriormente.

Crear un objeto de parámetros para transferir el nombre de la regla de recepción que se desea eliminar. Para llamar al método `deleteReceiptRule`, cree una promesa para invocar un objeto de servicio de Amazon SES transfiriendo los parámetros. Luego gestione la `response` en la devolución de llamada de la promesa.

```
// 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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos. Amazon SES crea la lista de conjuntos de reglas de recepción.

```
node ses_deleterecepitrule.js
```

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Creación de un conjunto de reglas de recepción

En este ejemplo, utilice un módulo de Node.js para enviar un correo electrónico con Amazon SES. Cree un módulo de Node.js con el nombre de archivo `ses_createrecepitruleset.js`. Configure el SDK como le hemos mostrado anteriormente.

Cree un objeto de parámetros para transferir el nombre del nuevo conjunto de reglas de recepción. Para llamar al método `createReceiptRuleSet`, cree una promesa para invocar un objeto de servicio de Amazon SES transfiriendo los parámetros. Luego gestione la respuesta en la devolución de llamada de la promesa.

```
// 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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos. Amazon SES crea la lista de conjuntos de reglas de recepción.

```
node ses_createrecepitruleset.js
```

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Eliminación de un conjunto de reglas de recepción

En este ejemplo, utilice un módulo de Node.js para enviar un correo electrónico con Amazon SES. Cree un módulo de Node.js con el nombre de archivo `ses_deleterecepitruleset.js`. Configure el SDK como le hemos mostrado anteriormente.

Cree un objeto para transferir el nombre del conjunto de reglas de recepción que desea eliminar. Para llamar al método `deleteReceiptRuleSet`, cree una promesa para invocar un objeto de servicio de Amazon SES transfiriendo los parámetros. Luego gestione la respuesta en la devolución de llamada de la promesa.

```
// 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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos. Amazon SES crea la lista de conjuntos de reglas de recepción.

```
node ses_deleterecepitruleset.js
```

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Ejemplos de Amazon Simple Notification Service

Amazon Simple Notification Service (Amazon SNS) es un servicio web que coordina y gestiona la entrega o el envío de mensajes a los puntos de enlace o clientes suscritos.

En Amazon SNS, existen dos tipos de clientes, publicadores y suscriptores, también conocidos como productores y consumidores.



Los publicadores se comunican de forma asíncrona con los suscriptores generando y enviando un mensaje a un tema, que es un punto de acceso lógico y un canal de comunicación. Los suscriptores (por ejemplo, servidores web, direcciones de correo electrónico, colas de Amazon SQS o funciones de Lambda) consumen o reciben el mensaje o la notificación a través de uno de los protocolos admitidos (Amazon SQS, HTTP/HTTPS, correo electrónico, SMS o AWS Lambda) cuando están suscritos al tema.

La API de JavaScript para Amazon SNS se expone a través de [Class: AWS.SNS](#).

Temas

- [Administración de temas en Amazon SNS](#)

- [Publicación de mensajes en Amazon SNS](#)
- [Administración de suscripciones en Amazon SNS](#)
- [Envío de mensajes SMS con Amazon SNS](#)

Administración de temas en Amazon SNS



Este ejemplo de código de Node.js muestra:

- Cómo crear temas en Amazon SNS en los que pueda publicar notificaciones.
- Cómo eliminar temas creados en Amazon SNS.
- Cómo obtener una lista de los temas disponibles.
- Cómo obtener y establecer atributos de temas.

El escenario

En este ejemplo, va a utilizar una serie de módulos de Node.js para crear, enumerar y eliminar temas de Amazon SNS y para gestionar atributos de los temas. Los módulos de Node.js usan el SDK para JavaScript para administrar temas mediante los métodos de clase de cliente AWS . SNS siguientes:

- [createTopic](#)
- [listTopics](#)
- [deleteTopic](#)
- [getTopicAttributes](#)
- [setTopicAttributes](#)

Tareas previas necesarias

Para configurar y ejecutar este ejemplo, primero debe completar estas tareas:

- Instale Node.js. Para obtener más información acerca de la instalación de Node.js consulte el [sitio web de Node.js](#).

- Cree un archivo de configuraciones compartidas con sus credenciales de usuario. Para obtener más información acerca de cómo proporcionar un archivo JSON de credenciales, consulte [Carga de credenciales en Node.js desde el archivo de credenciales compartidas](#).

Creación de un tema

En este ejemplo, utilice un módulo de Node.js para crear un tema de Amazon SNS. Cree un módulo de Node.js con el nombre de archivo `sns_createtopic.js`. Configure el SDK como le hemos mostrado anteriormente.

Cree un objeto para transferir el `Name` del nuevo tema al método `createTopic` de la clase de cliente `AWS.SNS`. Para llamar al método `createTopic`, cree una promesa para invocar un objeto de servicio de Amazon SNS pasando el objeto de parámetros. Luego gestione la `response` en la devolución de llamada de la promesa. Los `data` que la promesa devuelve contienen el ARN del tema.

```
// 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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos.

```
node sns_createtopic.js
```

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Generación de una lista de sus temas

En este ejemplo, utilice un módulo de Node.js para generar una lista de todos los temas de Amazon SNS. Cree un módulo de Node.js con el nombre de archivo `sns_listtopics.js`. Configure el SDK como le hemos mostrado anteriormente.

Cree un objeto vacío para transferirlo al método `listTopics` de la clase de cliente `AWS.SNS`. Para llamar al método `listTopics`, cree una promesa para invocar un objeto de servicio de Amazon SNS pasando el objeto de parámetros. Luego gestione la `response` en la devolución de llamada de la promesa. Los `data` que la promesa devuelve contienen una matriz de ARN de su tema.

```
// 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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos.

```
node sns_listtopics.js
```

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Eliminación de un tema

En este ejemplo, utilice un módulo de Node.js para eliminar un tema de Amazon SNS. Cree un módulo de Node.js con el nombre de archivo `sns_deletetopic.js`. Configure el SDK como le hemos mostrado anteriormente.

Cree un objeto que contenga el `TopicArn` del tema que se va a eliminar para transferirlo al método `deleteTopic` de la clase de cliente `AWS.SNS`. Para llamar al método `deleteTopic`, cree una promesa para invocar un objeto de servicio de Amazon SNS pasando el objeto de parámetros. Luego gestione la `response` en la devolución de llamada de la promesa.

```
// 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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos.

```
node sns_deletetopic.js
```

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Obtención de atributos de temas

En este ejemplo, utilice un módulo de Node.js para recuperar atributos de un tema de Amazon SNS. Cree un módulo de Node.js con el nombre de archivo `sns_gettopicattributes.js`. Configure el SDK como le hemos mostrado anteriormente.

Cree un objeto que contenga el `TopicArn` de un tema que se vaya a eliminar para transferirlo al método `getTopicAttributes` de la clase de cliente `AWS.SNS`. Para llamar al método `getTopicAttributes`, cree una promesa para invocar un objeto de servicio de Amazon SNS pasando el objeto de parámetros. Luego gestione la `response` en la devolución de llamada de la promesa.

```
// 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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos.

```
node sns_gettopicattributes.js
```

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Configuración de los atributos de un tema

En este ejemplo, utilice un módulo de Node.js para establecer los atributos mutables de un tema de Amazon SNS. Cree un módulo de Node.js con el nombre de archivo `sns_settopicattributes.js`. Configure el SDK como le hemos mostrado anteriormente.

Cree un objeto que contenga los parámetros para realizar la actualización del atributo, como el `TopicArn` del tema cuyos atributos desea establecer, el nombre del atributo que se va a establecer y el nuevo valor para dicho atributo. Solo puede establecer los atributos `Policy`, `DisplayName` y `DeliveryPolicy`. Transfiera los parámetros al método `setTopicAttributes` de la clase de cliente `AWS.SNS`. Para llamar al método `setTopicAttributes`, cree una promesa para invocar un objeto de servicio de Amazon SNS pasando el objeto de parámetros. Luego gestione la respuesta en la devolución de llamada de la promesa.

```
// 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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos.

```
node sns_settopicattributes.js
```

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Publicación de mensajes en Amazon SNS



Este ejemplo de código de Node.js muestra:

- Cómo publicar mensajes en un tema de Amazon SNS.

El escenario

En este ejemplo va a utilizar una serie de módulos de Node.js para publicar mensajes de Amazon SNS en puntos de conexión de temas, correos electrónicos o números de teléfono. Los módulos de Node.js usan el SDK para JavaScript para enviar mensajes mediante este método de la clase de cliente de AWS.SNS:

- [publish](#)

Tareas previas necesarias

Para configurar y ejecutar este ejemplo, primero debe completar estas tareas:

- Instale Node.js. Para obtener más información acerca de la instalación de Node.js consulte el [sitio web de Node.js](#).
- Cree un archivo de configuraciones compartidas con sus credenciales de usuario. Para obtener más información acerca de cómo proporcionar un archivo JSON de credenciales, consulte [Carga de credenciales en Node.js desde el archivo de credenciales compartidas](#).

Publicación de un mensaje en un tema de Amazon SNS

En este ejemplo, utilice un módulo de Node.js para publicar un mensaje en un tema de Amazon SNS. Cree un módulo de Node.js con el nombre de archivo `sns_publish_topic.js`. Configure el SDK como le hemos mostrado anteriormente.

Cree un objeto que contenga los parámetros para publicar un mensaje, como el texto del mensaje y el ARN del tema de Amazon SNS. Para obtener información detallada sobre los atributos de SMS, consulte [SetSMSAttributes](#).

Transfiera los parámetros al método `publish` de la clase de cliente `AWS.SNS`. Cree una promesa para invocar un objeto de servicio de Amazon SNS transfiriendo el objeto de parámetros. Luego gestione la respuesta en la devolución de llamada de la promesa.

```
// 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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos.

```
node sns_publishtopic.js
```

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Administración de suscripciones en Amazon SNS



Este ejemplo de código de Node.js muestra:

- Cómo mostrar una lista de todas las suscripciones en un tema de Amazon SNS.
- Cómo suscribir una dirección de correo electrónico, un punto de conexión de una aplicación o una función de Lambda a un tema de Amazon SNS.
- Cómo cancelar una suscripción a temas de Amazon SNS.

El escenario

En este ejemplo, va a utilizar una serie de módulos de Node.js para publicar mensajes de notificación en temas de Amazon SNS. Los módulos de Node.js usan el SDK para JavaScript para administrar temas mediante los métodos de clase de cliente AWS . SNS siguientes:

- [subscribe](#)
- [confirmSubscription](#)
- [listSubscriptionsByTopic](#)
- [unsubscribe](#)

Tareas previas necesarias

Para configurar y ejecutar este ejemplo, primero debe completar estas tareas:

- Instale Node.js. Para obtener más información acerca de la instalación de Node.js consulte el [sitio web de Node.js](#).
- Cree un archivo de configuraciones compartidas con sus credenciales de usuario. Para obtener más información acerca de cómo proporcionar un archivo JSON de credenciales, consulte [Carga de credenciales en Node.js desde el archivo de credenciales compartidas](#).

Generación de una lista de suscripciones a un tema

En este ejemplo, utilice un módulo de Node.js para generar una lista de todas las suscripciones a un tema de Amazon SNS. Cree un módulo de Node.js con el nombre de archivo `sns_listsubscriptions.js`. Configure el SDK como le hemos mostrado anteriormente.

Cree un objeto que contenga el parámetro `TopicArn` para el tema cuyas suscripciones desee incluir en la lista. Transfiera los parámetros al método `listSubscriptionsByTopic` de la clase de cliente `AWS.SNS`. Para llamar al método `listSubscriptionsByTopic`, cree una promesa para invocar un objeto de servicio de Amazon SNS pasando el objeto de parámetros. Luego gestione la `response` en la devolución de llamada de la promesa.

```
// 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 sublistPromise = new AWS.SNS({ apiVersion: "2010-03-31" })
  .listSubscriptionsByTopic(params)
  .promise();

// Handle promise's fulfilled/rejected states
sublistPromise
  .then(function (data) {
    console.log(data);
  })
  .catch(function (err) {
    console.error(err, err.stack);
  });
```

Para ejecutar el ejemplo, escriba lo siguiente en la línea de comandos.

```
node sns_listsubscriptions.js
```

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Suscripción de una dirección de correo electrónico a un tema

En este ejemplo, va a utilizar un módulo de Node.js para suscribir una dirección de correo electrónico para que reciba mensajes de correo electrónico SMTP de un tema de Amazon SNS. Cree un módulo de Node.js con el nombre de archivo `sns_subscribeemail.js`. Configure el SDK como le hemos mostrado anteriormente.

Cree un objeto que contenga el parámetro `Protocol` para especificar el protocolo de email, el `TopicArn` del tema al que se suscribirá y una dirección de correo electrónico como el mensaje `Endpoint`. Transfiera los parámetros al método `subscribe` de la clase de cliente `AWS.SNS`. Puede utilizar el método `subscribe` para suscribir varios puntos de conexión diferentes a un tema de Amazon SNS, en función de los valores que se utilicen para los parámetros que se transfieran, tal y como se verá en otros ejemplos de este tema.

Para llamar al método `subscribe`, cree una promesa para invocar un objeto de servicio de Amazon SNS pasando el objeto de parámetros. Luego gestione la `response` en la devolución de llamada de la promesa.

```
// 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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos.

```
node sns_subscribeemail.js
```

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Suscripción de un punto de conexión de aplicación a un tema

En este ejemplo, va a utilizar un módulo de Node.js para suscribir un punto de conexión de una aplicación móvil para que reciba notificaciones de un tema de Amazon SNS. Cree un módulo de Node.js con el nombre de archivo `sns_subscribeapp.js`. Configure el SDK como le hemos mostrado anteriormente.

Cree un objeto que contenga el parámetro `Protocol` para especificar el protocolo de la `application`, el `TopicArn` del tema al que va a suscribirse y el ARN del punto de enlace de la

aplicación móvil para el parámetro `Endpoint`. Transfiera los parámetros al método `subscribe` de la clase de cliente `AWS.SNS`.

Para llamar al método `subscribe`, cree una promesa para invocar un objeto de servicio de Amazon SNS pasando el objeto de parámetros. Luego gestione la respuesta en la devolución de llamada de la promesa.

```
// 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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos.

```
node sns_subscribeapp.js
```

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Suscripción de una función de Lambda a un tema

En este ejemplo, va a utilizar un módulo de Node.js para suscribir una función AWS Lambda para que reciba notificaciones de un tema de Amazon SNS. Cree un módulo de Node.js con el nombre de archivo `sns_subscribe_lambda.js`. Configure el SDK como le hemos mostrado anteriormente.

Cree un objeto que contenga el parámetro `Protocol` y que especifique el protocolo `lambda`, el `TopicArn` del tema al que va a suscribirse y el ARN de una función AWS Lambda como el parámetro `Endpoint`. Transfiera los parámetros al método `subscribe` de la clase de cliente `AWS.SNS`.

Para llamar al método `subscribe`, cree una promesa para invocar un objeto de servicio de Amazon SNS pasando el objeto de parámetros. Luego gestione la `response` en la devolución de llamada de la promesa.

```
// 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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos.

```
node sns_subscribelambda.js
```

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Cancelación de la suscripción a un tema

En este ejemplo, utilice un módulo de Node.js para cancelar la suscripción a un tema de Amazon SNS. Cree un módulo de Node.js con el nombre de archivo `sns_unsubscribe.js`. Configure el SDK como le hemos mostrado anteriormente.

Cree un objeto que contenga el parámetro `SubscriptionArn` que especifique el ARN de la suscripción que desea cancelar. Transfiera los parámetros al método `unsubscribe` de la clase de cliente `AWS.SNS`.

Para llamar al método `unsubscribe`, cree una promesa para invocar un objeto de servicio de Amazon SNS pasando el objeto de parámetros. Luego gestione la respuesta en la devolución de llamada de la promesa.

```
// 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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos.

```
node sns_unsubscribe.js
```

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Envío de mensajes SMS con Amazon SNS



Este ejemplo de código de Node.js muestra:

- Cómo obtener y establecer las preferencias de mensajería de SMS para Amazon SNS.
- Cómo comprobar un número de teléfono para ver si se ha desactivado la opción de recepción de mensajes SMS.
- Cómo obtener una lista de números de teléfono con la opción de recepción de mensajes SMS desactivada
- Cómo enviar un mensaje SMS.

El escenario

Puede utilizar Amazon SNS para enviar mensajes de texto o mensajes SMS a dispositivos habilitados para recibir SMS. Dispone de la capacidad de enviar un mensaje directamente a un número de teléfono o de enviar un mensaje a varios números de teléfono a la vez suscribiendo dichos números de teléfono a un tema y enviando su mensaje al tema.

En este ejemplo, va a utilizar una serie de módulos de Node.js para publicar mensajes de texto SMS de Amazon SNS a dispositivos habilitados para SMS. Los módulos de Node.js usan el SDK para JavaScript para publicar mensajes SMS mediante los métodos de clase de cliente de AWS . SNS siguientes:

- [getSMSAttributes](#)
- [setSMSAttributes](#)
- [checkIfPhoneNumberIsOptedOut](#)
- [listPhoneNumbersOptedOut](#)
- [publish](#)

Tareas previas necesarias

Para configurar y ejecutar este ejemplo, primero debe completar estas tareas:

- Instale Node.js. Para obtener más información acerca de la instalación de Node.js consulte el [sitio web de Node.js](#).
- Cree un archivo de configuraciones compartidas con sus credenciales de usuario. Para obtener más información acerca de cómo proporcionar un archivo JSON de credenciales, consulte [Carga de credenciales en Node.js desde el archivo de credenciales compartidas](#).

Obtención de atributos de SMS

Utilice Amazon SNS para especificar las preferencias de mensajería SMS, como la forma en que se optimizan sus envíos (por coste o por fiabilidad de la entrega), su límite de gasto mensual, cómo se registran los envíos de mensajes y si desea suscribirse a informes de uso de SMS diarios. Estas preferencias se recuperan y se establecen como atributos SMS para Amazon SNS.

En este ejemplo, va a utilizar un módulo de Node.js para obtener los atributos de SMS actuales en Amazon SNS. Cree un módulo de Node.js con el nombre de archivo `sns_getsmstype.js`. Configure el SDK como le hemos mostrado anteriormente. Cree un objeto que contenga los parámetros para obtener atributos de SMS, como los nombres de los atributos individuales que desea obtener. Para obtener más información sobre los atributos de SMS disponibles, consulte [SetSMSAttributes](#) en la Referencia de la API de Amazon Simple Notification Service.

Este ejemplo obtiene el atributo `DefaultSMSType` que controla si se envían mensajes SMS como `Promotional`, que optimiza la entrega de mensajes para conseguir el costo más bajo, o como `Transactional`, que optimiza el envío de mensajes para conseguir la máxima fiabilidad. Transfiera los parámetros al método `setTopicAttributes` de la clase de cliente `AWS.SNS`. Para llamar al método `getSMSAttributes`, cree una promesa para invocar un objeto de servicio de Amazon SNS pasando el objeto de parámetros. Luego gestione la `response` en la devolución de llamada de la promesa.

```
// 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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos.

```
node sns_getsmstype.js
```

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Configuración de atributos de SMS

En este ejemplo, va a utilizar un módulo de Node.js para obtener los atributos de SMS actuales en Amazon SNS. Cree un módulo de Node.js con el nombre de archivo `sns_setsmstype.js`. Configure el SDK como le hemos mostrado anteriormente. Cree un objeto que contenga los parámetros para establecer atributos de SMS, como los nombres de los atributos individuales que desea establecer y los valores que se establecerán para cada uno de ellos. Para todos los atributos de SMS, consulte [SetSMSAttributes](#) en la Referencia de la API de Amazon Simple Notification Service.

Este ejemplo establece el atributo `DefaultSMSType` en `Transactional`, que optimiza el envío de mensajes para conseguir la máxima fiabilidad. Transfiera los parámetros al método `setTopicAttributes` de la clase de cliente `AWS.SNS`. Para llamar al método

`getSMSAttributes`, cree una promesa para invocar un objeto de servicio de Amazon SNS pasando el objeto de parámetros. Luego gestione la `response` en la devolución de llamada de la promesa.

```
// 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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos.

```
node sns_setsmstype.js
```

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Comprobación de si se ha desactivado un número de teléfono

En este ejemplo, va a utilizar un módulo de Node.js para comprobar si un número de teléfono tiene desactivada la opción de recibir mensajes SMS. Cree un módulo de Node.js con el nombre de

archivo `sns_checkphoneoptout.js`. Configure el SDK como le hemos mostrado anteriormente. Cree un objeto que contenga el número de teléfono que se desea comprobar como parámetro.

Este ejemplo establece el parámetro `PhoneNumber` para especificar el número de teléfono que se va a comprobar. Transfiera el objeto al método `checkIfPhoneNumberIsOptedOut` de la clase de cliente `AWS.SNS`. Para llamar al método `checkIfPhoneNumberIsOptedOut`, cree una promesa para invocar un objeto de servicio de Amazon SNS pasando el objeto de parámetros. Luego gestione la `response` en la devolución de llamada de la promesa.

```
// 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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos.

```
node sns_checkphoneoptout.js
```

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Generación de una lista de los números de teléfono desactivados

En este ejemplo, va a utilizar un módulo de Node.js para obtener una lista de números de teléfono que tienen desactivada la opción de recibir mensajes SMS. Cree un módulo de Node.js con el nombre de archivo `sns_listnumbersoptedout.js`. Configure el SDK como le hemos mostrado anteriormente. Cree un objeto vacío como parámetro.

Transfiera el objeto al método `listPhoneNumbersOptedOut` de la clase de cliente `AWS.SNS`. Para llamar al método `listPhoneNumbersOptedOut`, cree una promesa para invocar un objeto de servicio de Amazon SNS pasando el objeto de parámetros. Luego gestione la `response` en la devolución de llamada de la promesa.

```
// 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 phonenumberlistPromise = new AWS.SNS({ apiVersion: "2010-03-31" })
  .listPhoneNumbersOptedOut({})
  .promise();

// Handle promise's fulfilled/rejected states
phonenumberlistPromise
  .then(function (data) {
    console.log(data);
  })
  .catch(function (err) {
    console.error(err, err.stack);
  });
```

Para ejecutar el ejemplo, escriba lo siguiente en la línea de comandos.

```
node sns_listnumbersoptedout.js
```

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Publicación de un mensaje SMS

En este ejemplo, utilice un módulo de Node.js para enviar un mensaje SMS a un número de teléfono. Cree un módulo de Node.js con el nombre de archivo `sns_publishsms.js`. Configure el SDK como le hemos mostrado anteriormente. Cree un objeto que contenga los parámetros `Message` y `PhoneNumber`.

Cuando envíe un mensaje SMS, especifique el número de teléfono usando la formato E.164. E.164 es un estándar de estructura de número de teléfono utilizado para las telecomunicaciones internacionales. Los números que aplican este formato pueden tener un máximo de 15 dígitos y van

prefijados con el carácter (+) y el código de país. Por ejemplo, un número de teléfono de los EE. UU. en formato E.164 se mostraría como +1001XXX5550100.

En este ejemplo se establece el parámetro `PhoneNumber` para especificar el número de teléfono al que se enviará el mensaje. Transfiera el objeto al método `publish` de la clase de cliente `AWS.SNS`. Para llamar al método `publish`, cree una promesa para invocar un objeto de servicio de Amazon SNS pasando el objeto de parámetros. Luego gestione la `response` en la devolución de llamada de la promesa.

```
// 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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos.

```
node sns_publishsms.js
```

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Ejemplos de Amazon SQS

Amazon Simple Queue Service (Amazon SQS) es un servicio de colas de mensajes rápido, de confianza, escalable y totalmente administrado. Amazon SQS permite desacoplar componentes de una aplicación en la nube. Amazon SQS incluye colas estándar con un alto rendimiento y procesamiento “al menos una vez”, y colas FIFO que ofrecen distribución FIFO (primero en entrar, primero en salir) y procesamiento “exactamente una vez”.



La API de JavaScript para Amazon SQS se expone a través de la clase de cliente `AWS.SQS`. Para obtener más información acerca de cómo usar la clase de cliente de Amazon SQS, consulte [Class: AWS.SQS](#) en la referencia de la API.

Temas

- [Uso de colas en Amazon SQS](#)
- [Envío y recepción de mensajes en Amazon SQS](#)
- [Administración del tiempo de espera de visibilidad en Amazon SQS](#)
- [Habilitación del sondeo largo en Amazon SQS](#)
- [Uso de colas de mensajes fallidos en Amazon SQS](#)

Uso de colas en Amazon SQS



Este ejemplo de código de Node.js muestra:

- Cómo obtener una lista de todas sus colas de mensajes.
- Cómo obtener la URL de una cola en concreto.
- Cómo crear y eliminar colas.

Acerca del ejemplo

En este ejemplo, se utilizan una serie de módulos de Node.js para trabajar con colas. Los módulos de Node.js utilizan SDK para JavaScript para permitir que las colas llamen a los siguientes métodos de la clase de cliente `AWS.SQS`:

- [listQueues](#)
- [createQueue](#)
- [getQueueUrl](#)
- [deleteQueue](#)

Para obtener más información acerca de los mensajes de Amazon SQS, consulte [Cómo funcionan las colas](#) en la Guía para desarrolladores de Amazon Simple Queue Service.

Tareas previas necesarias

Para configurar y ejecutar este ejemplo, primero debe completar estas tareas:

- Instale Node.js. Para obtener más información acerca de la instalación de Node.js consulte el [sitio web de Node.js](#).
- Cree un archivo de configuraciones compartidas con sus credenciales de usuario. Para obtener más información acerca de cómo crear un archivo de credenciales compartidas, consulte [Carga de credenciales en Node.js desde el archivo de credenciales compartidas](#).

Generación de una lista de sus colas

Cree un módulo de Node.js con el nombre de archivo `sqs_listqueues.js`. Asegúrese de configurar el SDK tal y como se ha indicado anteriormente. Para obtener acceso a Amazon SQS, cree un objeto de servicio de `AWS.SQS`. Cree un objeto JSON que contenga los parámetros necesarios para generar una lista de sus colas; de forma predeterminada es un objeto vacío. Llame al método `listQueues` para recuperar la lista de colas. La devolución de llamada devuelve las direcciones URL de todas las colas.

```
// 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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos.

```
node sqs_listqueues.js
```

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Creación de una cola

Cree un módulo de Node.js con el nombre de archivo `sqs_createqueue.js`. Asegúrese de configurar el SDK tal y como se ha indicado anteriormente. Para obtener acceso a Amazon SQS, cree un objeto de servicio de `AWS.SQS`. Cree un objeto JSON que contenga los parámetros necesarios para obtener una lista de las colas que incluya el nombre de la cola que se ha creado. Los parámetros también pueden contener atributos de la cola como, por ejemplo, cuántos segundos se tardará en entregar el mensaje o cuántos segundos se conservará un mensaje recibido. Llame al método `createQueue`. La devolución de llamada devuelve la dirección URL de la cola creada.

```
// 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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos.

```
node sqs_createqueue.js
```

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Obtención de la dirección URL de una cola

Cree un módulo de Node.js con el nombre de archivo `sqs_getqueueurl.js`. Asegúrese de configurar el SDK tal y como se ha indicado anteriormente. Para obtener acceso a Amazon SQS, cree un objeto de servicio de `AWS.SQS`. Cree un objeto JSON que contenga los parámetros necesarios para obtener una lista de colas que incluya el nombre de la cola cuya dirección URL desea obtener. Llame al método `getQueueUrl`. La devolución de llamada devuelve la dirección URL de la cola 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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos.

```
node sqs_getqueueurl.js
```

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Eliminación de una cola

Cree un módulo de Node.js con el nombre de archivo `sqs_deletequeue.js`. Asegúrese de configurar el SDK tal y como se ha indicado anteriormente. Para obtener acceso a Amazon SQS, cree un objeto de servicio de `AWS.SQS`. Cree un objeto JSON que contenga los parámetros necesarios para eliminar una cola y que conste de la dirección URL de la cola que desea eliminar. Llame al 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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos.

```
node sqs_deletequeue.js
```

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Envío y recepción de mensajes en Amazon SQS



Este ejemplo de código de Node.js muestra:

- Cómo enviar mensajes en una cola.
- Cómo recibir mensajes en una cola.
- Cómo eliminar mensajes de una cola.

El escenario

En este ejemplo, se utilizan una serie de módulos de Node.js para enviar y recibir mensajes. Los módulos de Node.js utilizan el SDK para JavaScript para enviar y recibir mensajes mediante los métodos de la clase de cliente AWS .SQS siguientes:

- [sendMessage](#)
- [receiveMessage](#)
- [deleteMessage](#)

Para obtener más información acerca de los mensajes de Amazon SQS, consulte [Envío de un mensaje a una cola de Amazon SQS](#) y [Recepción y eliminación de un mensaje de una cola de Amazon SQS](#) en la Guía para desarrolladores de Amazon Simple Queue Service.

Tareas previas necesarias

Para configurar y ejecutar este ejemplo, primero debe completar estas tareas:

- Instale Node.js. Para obtener más información acerca de la instalación de Node.js consulte el [sitio web de Node.js](#).
- Cree un archivo de configuraciones compartidas con sus credenciales de usuario. Para obtener más información acerca de cómo crear un archivo de credenciales compartidas, consulte [Carga de credenciales en Node.js desde el archivo de credenciales compartidas](#).
- Crear una cola de Amazon SQS. Para ver un ejemplo de cómo crear una cola, consulte [Uso de colas en Amazon SQS](#).

Envío de un mensaje a una cola

Cree un módulo de Node.js con el nombre de archivo `sqs_sendmessage.js`. Asegúrese de configurar el SDK tal y como se ha indicado anteriormente. Para obtener acceso a Amazon SQS, cree un objeto de servicio de AWS.SQS. Cree un objeto JSON que contenga los parámetros necesarios para su mensaje y que incluya la dirección URL de la cola a la que desea enviar este mensaje. En este ejemplo, el mensaje proporciona detalles (título, autor y número de semanas en la lista) sobre un libro de una lista de libros más vendidos de ficción.

Llame al método `sendMessage`. La devolución de llamada devuelve el ID único del mensaje.

```
// 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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos.

```
node sqs_sendmessage.js
```

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Recepción y eliminación de mensajes de una cola

Cree un módulo de Node.js con el nombre de archivo `sqs_receivemessage.js`. Asegúrese de configurar el SDK tal y como se ha indicado anteriormente. Para obtener acceso a Amazon SQS, cree un objeto de servicio de `AWS.SQS`. Cree un objeto JSON que contenga los parámetros necesarios para su mensaje y que incluya la dirección URL de donde desee recibir mensajes. En este ejemplo, los parámetros especifican la recepción de todos los atributos de mensajes, así como la recepción de un máximo de 10 mensajes.

Llame al método `receiveMessage`. La devolución de llamada devuelve una matriz de objetos de `Message` desde la que puede recuperar el `ReceiptHandle` de cada mensaje que puede utilizar más adelante para eliminar dicho mensaje. Cree otro objeto JSON que contenga los parámetros necesarios para eliminar el mensaje, que son la dirección URL de la cola y el valor de `ReceiptHandle`. Llame al método `deleteMessage` para eliminar el mensaje que ha recibido.

```
// 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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos.

```
node sqs_receivemessage.js
```

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Administración del tiempo de espera de visibilidad en Amazon SQS



Este ejemplo de código de Node.js muestra:

- Cómo especificar el intervalo de tiempo durante el cual los mensajes que una cola recibe no son visibles.

El escenario

En este ejemplo, se utiliza un módulo de Node.js para administrar el tiempo de espera de visibilidad. El módulo de Node.js usa el SDK para JavaScript para administrar el tiempo de espera de visibilidad mediante este método de la clase de cliente AWS .SQS:

- [changeMessageVisibility](#)

Para obtener más información sobre tiempos de espera de visibilidad de Amazon SQS, consulte [Tiempo de espera de visibilidad](#) en la Guía para desarrolladores de Amazon Simple Queue Service.

Tareas previas necesarias

Para configurar y ejecutar este ejemplo, primero debe completar estas tareas:

- Instale Node.js. Para obtener más información acerca de la instalación de Node.js consulte el [sitio web de Node.js](#).
- Cree un archivo de configuraciones compartidas con sus credenciales de usuario. Para obtener más información acerca de cómo crear un archivo de credenciales compartidas, consulte [Carga de credenciales en Node.js desde el archivo de credenciales compartidas](#).
- Crear una cola de Amazon SQS. Para ver un ejemplo de cómo crear una cola, consulte [Uso de colas en Amazon SQS](#).
- Envíe un mensaje a la cola. Para ver un ejemplo de cómo enviar un mensaje a una cola, consulte [Envío y recepción de mensajes en Amazon SQS](#).

Cambio del tiempo de espera de visibilidad

Cree un módulo de Node.js con el nombre de archivo `sqs_changingvisibility.js`. Asegúrese de configurar el SDK tal y como se ha indicado anteriormente. Para acceder a Amazon Simple Queue Service, cree un objeto de servicio de `AWS.SQS`. Recibir el mensaje de la cola

Al recibir el mensaje de la cola, cree un objeto JSON que contenga los parámetros necesarios para configurar el tiempo de espera, como la dirección URL de la cola que contiene el mensaje, el `ReceiptHandle` que se devuelve cuando el mensaje se recibe y el nuevo tiempo de espera en segundos. Llame al 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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos.

```
node sqs_changingvisibility.js
```

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Habilitación del sondeo largo en Amazon SQS



Este ejemplo de código de Node.js muestra:

- Cómo habilitar los sondeos largos para una cola que se acaba de crear.
- Cómo habilitar un sondeo largo para una cola ya existente.
- Cómo habilitar un sondeo largo al recibir un mensaje.

El escenario

El sondeo largo reduce el número de respuestas vacías al permitir que Amazon SQS espere un tiempo determinado a que haya un mensaje disponible en la cola antes de enviar una respuesta. Asimismo, el sondeo largo elimina las respuestas vacías falsas, ya que consulta todos los servidores en lugar de solo una muestra de servidores. Para habilitar el sondeo largo, debe especificar un tiempo de espera diferente de cero para los mensajes recibidos. Puede hacerlo estableciendo el parámetro `ReceiveMessageWaitTimeSeconds` de una cola o estableciendo el parámetro `WaitTimeSeconds` en un mensaje al recibirse este.

En este ejemplo, se utilizan una serie de módulos de Node.js para habilitar un sondeo largo. Los módulos de Node.js usan el SDK para JavaScript para habilitar un sondeo largo con los métodos de la clase de cliente AWS .SQS siguientes:

- [setQueueAttributes](#)
- [receiveMessage](#)
- [createQueue](#)

Para obtener más información sobre el sondeo largo de Amazon SQS, consulte [Sondeo largo](#) en la Guía para desarrolladores de Amazon Simple Queue Service.

Tareas previas necesarias

Para configurar y ejecutar este ejemplo, primero debe completar estas tareas:

- Instale Node.js. Para obtener más información acerca de la instalación de Node.js consulte el [sitio web de Node.js](#).
- Cree un archivo de configuraciones compartidas con sus credenciales de usuario. Para obtener más información acerca de cómo crear un archivo de credenciales compartidas, consulte [Carga de credenciales en Node.js desde el archivo de credenciales compartidas](#).

Habilitación de un sondeo largo al crear una cola

Cree un módulo de Node.js con el nombre de archivo `sqs_longpolling_createqueue.js`. Asegúrese de configurar el SDK tal y como se ha indicado anteriormente. Para obtener acceso a Amazon SQS, cree un objeto de servicio de AWS .SQS. Cree un objeto JSON que contenga los parámetros necesarios para crear una cola, como un valor distinto de cero para el parámetro `ReceiveMessageWaitTimeSeconds`. Llame al método `createQueue`. El sondeo largo se habilita entonces para la cola.

```
// 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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos.

```
node sqs_longpolling_createqueue.js
```

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Habilitar el sondeo largo en una cola existente

Cree un módulo de Node.js con el nombre de archivo `sqs_longpolling_existingqueue.js`. Asegúrese de configurar el SDK tal y como se ha indicado anteriormente. Para acceder a Amazon Simple Queue Service, cree un objeto de servicio de `AWS.SQS`. Cree un objeto JSON que contenga los parámetros necesarios para establecer los atributos de la cola, como un valor distinto de cero para el parámetro `ReceiveMessageWaitTimeSeconds` y la dirección URL de la cola. Llame al método `setQueueAttributes`. El sondeo largo se habilita entonces para la cola.

```
// 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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos.

```
node sqs_longpolling_existingqueue.js
```

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Habilitar el sondeo largo al recibir un mensaje

Cree un módulo de Node.js con el nombre de archivo `sqs_longpolling_receivemessage.js`. Asegúrese de configurar el SDK tal y como se ha indicado anteriormente. Para acceder a Amazon Simple Queue Service, cree un objeto de servicio de `AWS.SQS`. Cree un objeto JSON que contenga los parámetros necesarios para recibir mensajes, como un valor distinto de cero para el parámetro `WaitTimeSeconds` y la dirección URL de la cola. Llame al 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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos.

```
node sqs_longpolling_receivemessage.js
```

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Uso de colas de mensajes fallidos en Amazon SQS



Este ejemplo de código de Node.js muestra:

- Cómo utilizar una cola para recibir y conservar mensajes de otras colas que las colas no pueden procesar.

El escenario

Una cola de mensajes fallidos es una cola a la que otras colas (de origen) pueden enviar mensajes que no se han podido procesar correctamente. Puede apartar y aislar estos mensajes en la cola de mensajes fallidos para determinar por qué no se procesaron correctamente. Debe configurar individualmente cada cola de origen que envía mensajes a una cola de mensajes fallidos. Varias colas pueden dirigirse a una única cola de mensajes fallidos.

En este ejemplo, se utiliza un módulo de Node.js para dirigir mensajes a una cola de mensajes fallidos. El módulo de Node.js usa el SDK para JavaScript para usar las colas de mensajes fallidos mediante este método de la clase de cliente de AWS .SQS:

- [setQueueAttributes](#)

Para obtener más información sobre las colas de mensajes fallidos en Amazon SQS, consulte [Uso de colas de mensajes fallidos en Amazon SQS](#) en la Guía para desarrolladores de Amazon Simple Queue Service.

Tareas previas necesarias

Para configurar y ejecutar este ejemplo, primero debe completar estas tareas:

- Instale Node.js. Para obtener más información acerca de la instalación de Node.js consulte el [sitio web de Node.js](#).
- Cree un archivo de configuraciones compartidas con sus credenciales de usuario. Para obtener más información acerca de cómo crear un archivo de credenciales compartidas, consulte [Carga de credenciales en Node.js desde el archivo de credenciales compartidas](#).
- Cree una cola de Amazon SQS para usarla como cola de mensajes fallidos. Para ver un ejemplo de cómo crear una cola, consulte [Uso de colas en Amazon SQS](#).

Configuración de colas de origen

Después de crear una cola para que sirva de cola de mensajes fallidos, deberá configurar las demás colas que dirigen los mensajes no procesados a la cola de mensajes fallidos. Para ello, especifique una política de redireccionamiento que identifique la cola que debe utilizarse como cola de mensajes fallidos y el número máximo de recepciones por mensaje individual antes de que se enruten hacia la cola de mensajes fallidos.

Cree un módulo de Node.js con el nombre de archivo `sqs_deadletterqueue.js`. Asegúrese de configurar el SDK tal y como se ha indicado anteriormente. Para obtener acceso a Amazon SQS, cree un objeto de servicio de `AWS.SQS`. Cree un objeto JSON que contenga los parámetros necesarios para actualizar atributos de cola, como el parámetro `RedrivePolicy` que especifica tanto el ARN de la cola de mensajes fallidos y el valor de `maxReceiveCount`. Especifique también la cola de origen de la URL que desea configurar. Llame al 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 ejecutar el ejemplo, escriba lo siguiente en la línea de comandos.

```
node sqs_deadletterqueue.js
```

Este código de ejemplo se puede encontrar [aquí en GitHub](#).

Tutoriales

En los siguientes tutoriales aprenderá a realizar diferentes tareas relacionadas con el uso de AWS SDK para JavaScript.

Temas

- [Tutorial: Configuración de Node.js en una instancia de Amazon EC2](#)

Tutorial: Configuración de Node.js en una instancia de Amazon EC2

Un escenario común para utilizar Node.js con el SDK para JavaScript consiste en configurar y ejecutar una aplicación web de Node.js en una instancia de Amazon Elastic Compute Cloud (Amazon EC2). En este tutorial, creará una instancia Linux, se conectará a ella mediante SSH y, a continuación, instalará Node.js para ejecutarse en dicha instancia.

Requisitos previos

En este tutorial se presupone que ya ha lanzado una instancia Linux con un nombre de DNS público al que se puede tener acceso desde Internet y al que se puede conectar a través de SSH. Para obtener más información, consulte [Paso 1: Lanzamiento de una instancia](#) en la Guía del usuario de Amazon EC2.

Important

Utilice Imagen de máquina de Amazon (AMI) de Amazon Linux 2023 al ejecutar una nueva instancia de Amazon EC2.

También debe haber configurado el grupo de seguridad para que permita las conexiones SSH (puerto 22), HTTP (puerto 80) y HTTPS (puerto 443). Para obtener más información sobre estos requisitos previos, consulte [Configuración con Amazon EC2](#) en la Guía del usuario de Amazon EC2.

Procedimiento

El siguiente procedimiento le será útil para instalar Node.js en una instancia Amazon Linux. Puede utilizar este servidor para alojar una aplicación web de Node.js.

Para configurar Node.js en su instancia Linux

1. Conecte su instancia Linux como `ec2-user` mediante SSH.
2. Instale el administrador de versiones de nodos (nvm) escribiendo lo siguiente en la línea de comandos.

Warning

AWS no controla el siguiente código. Antes de ejecutarlo, asegúrese de comprobar su autenticidad e integridad. Para obtener más información sobre este código, visite el repositorio [nvm](#) de GitHub.

```
curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.39.7/install.sh | bash
```

Utilizaremos nvm para instalar Node.js, ya que nvm puede instalar varias versiones de Node.js y permitirle alternar entre ellas.

3. Cargue nvm escribiendo lo siguiente en la línea de comandos.

```
source ~/.bashrc
```

4. Utilice nvm para instalar la última versión LTS de Node.js escribiendo lo siguiente en la línea de comandos.

```
nvm install --lts
```

Si instala Node.js, también instalará el administrador de paquetes de nodos (npm) para poder instalar módulos adicionales según sea necesario.

5. Compruebe que Node.js esté instalado y ejecutándose correctamente, escribiendo lo siguiente en la línea de comandos.

```
node -e "console.log('Running Node.js ' + process.version)"
```

Esto presenta el siguiente mensaje que muestra la versión de Node.js que se está ejecutando.

Running Node.js *VERSION*

Note

La instalación del nodo solo se aplica a la sesión actual de Amazon EC2. Si reinicia la sesión de CLI, debe usar `nvm` para habilitar la versión de nodo instalada. Si la instancia finaliza, necesitará instalar de nuevo el nodo. La alternativa es crear una imagen de máquina de Amazon (AMI) de la instancia de Amazon EC2 cuando tenga la configuración que desea conservar, tal y como se describe en la siguiente sección.

Creación de una imagen de Amazon Machine (AMI).

Después de instalar Node.js en una instancia de Amazon EC2, puede crear una imagen de máquina de Amazon (AMI) a partir de dicha instancia. Si crea una AMI le será más fácil aprovisionar varias instancias de Amazon EC2 con la misma instalación de Node.js. Para obtener información acerca de cómo crear una AMI a partir de una instancia existente, consulte [Creación de una AMI de Linux con el respaldo de Amazon EBS](#) en la Guía del usuario de Amazon EC2.

Activos relacionados

Para obtener más información acerca de los comandos y el software que se utilizan en este tema, visite las siguientes páginas web:

- administrador de versiones de nodos (nvm): consulte el [repositorio nvm en GitHub](#).
- administrador de paquetes de nodos (npm): consulte el [sitio web de npm](#).

Referencia de la API de JavaScript

Los temas de referencia de la API para la versión más reciente del SDK para JavaScript se encuentran en:

[Guía de referencia de la API de AWS SDK para JavaScript.](#)

Registro de cambios del SDK en GitHub

El registro de cambios para las publicaciones de la versión 2.4.8 y versiones posteriores se encuentra en:

[Registro de cambios.](#)

Migración a la versión 3 de AWS SDK para JavaScript

La versión 3 de AWS SDK para JavaScript es una reescritura importante de la versión 2. Para obtener más información sobre la migración a la versión 3, consulte [Migración de la versión 2.x a la 3.x de AWS SDK para JavaScript](#) en la Guía para desarrolladores de la versión 3 de AWS SDK para JavaScript.

Seguridad de este producto o servicio de AWS

La seguridad en la nube de Amazon Web Services (AWS) es la máxima prioridad. Como cliente de AWS, se beneficia de una arquitectura de red y un centro de datos que se han diseñado para satisfacer los requisitos de seguridad de las organizaciones más exigentes. La seguridad es una responsabilidad compartida entre AWS y usted. En el [modelo de responsabilidad compartida](#), se habla de “seguridad de la nube” y “seguridad en la nube”:

Seguridad de la nube: AWS es responsable de proteger la infraestructura que ejecuta todos los servicios ofrecidos en la nube de AWS y de proporcionar servicios que puede utilizar de forma segura. Nuestra responsabilidad de seguridad es la mayor prioridad en AWS y auditores externos prueban y verifican la eficacia de nuestra seguridad con frecuencia como parte de los [programas de conformidad de AWS](#).

Seguridad en la nube: su responsabilidad viene determinada por el servicio de AWS que usa y por otros factores, como la confidencialidad de los datos, los requisitos de la organización, y las normas y los reglamentos aplicables.

Este producto o servicio de AWS sigue el [modelo de responsabilidad compartida](#) a través de los servicios específicos de Amazon Web Services (AWS) que admite. Para obtener información sobre la seguridad de los servicios de AWS, consulte la [página de documentación sobre la seguridad de los servicios de AWS](#) y los [servicios de AWS sujetos a las medidas de conformidad de AWS de cada programa de conformidad](#).

Temas

- [Protección de datos de este producto o servicio de AWS](#)
- [Gestión de identidad y acceso](#)
- [Validación de la conformidad de este producto o servicio de AWS](#)
- [Resiliencia de este producto o servicio de AWS](#)
- [Seguridad de la infraestructura de este producto o servicio de AWS](#)
- [Aplicación de una versión mínima de TLS](#)

Protección de datos de este producto o servicio de AWS

El [modelo de responsabilidad compartida](#) de AWS se aplica a la protección de datos de este producto o servicio de AWS. Como se describe en este modelo, AWS es responsable de proteger

la infraestructura global que ejecuta toda la Nube de AWS. Eres responsable de mantener el control sobre el contenido alojado en esta infraestructura. También eres responsable de las tareas de administración y configuración de seguridad para los Servicios de AWS que utiliza. Para obtener más información sobre la privacidad de los datos, consulte las [Preguntas frecuentes sobre la privacidad de datos](#). Para obtener información sobre la protección de datos en Europa, consulte la publicación de blog sobre el [Modelo de responsabilidad compartida de AWS y GDPR](#) en el Blog de seguridad de AWS.

Con fines de protección de datos, recomendamos proteger las credenciales de la Cuenta de AWS y configurar cuentas de usuario individuales con AWS IAM Identity Center o AWS Identity and Access Management (IAM). De esta manera, solo se otorgan a cada usuario los permisos necesarios para cumplir sus obligaciones laborales. También recomendamos proteger sus datos de la siguiente manera:

- Utiliza la autenticación multifactor (MFA) en cada cuenta.
- Utiliza SSL/TLS para comunicarse con los recursos de AWS. Exigimos TLS 1.2 y recomendamos TLS 1.3.
- Configure los registros de API y de actividad de los usuarios con AWS CloudTrail. Para obtener información sobre cómo utilizar registros de seguimiento de CloudTrail para capturar actividades de AWS, consulte [Working with CloudTrail trails](#) en la Guía del usuario de AWS CloudTrail.
- Utiliza las soluciones de cifrado de AWS, junto con todos los controles de seguridad predeterminados dentro de los servicios de Servicios de AWS.
- Utiliza servicios de seguridad administrados avanzados, como Amazon Macie, que lo ayuden a detectar y proteger la información confidencial almacenada en Amazon S3.
- Si necesita módulos criptográficos validados FIPS 140-3 al acceder a AWS a través de una interfaz de la línea de comandos o una API, utiliza un punto de conexión de FIPS. Para obtener más información sobre los puntos de conexión de FIPS disponibles, consulte [Estándar de procesamiento de la información federal \(FIPS\) 140-3](#).

Se recomienda encarecidamente no introducir nunca información confidencial o sensible, como por ejemplo, direcciones de correo electrónico de clientes, en etiquetas o campos de formato libre, tales como el campo Nombre. Esto incluye cuando trabaje con este producto o servicio de AWS u otros Servicios de AWS a través de la consola, la API, AWS CLI o SDK de AWS. Cualquier dato que introduzca en etiquetas o campos de formato libre utilizados para los nombres se pueden emplear para los registros de facturación o diagnóstico. Si proporciona una URL a un servidor externo,

recomendamos encarecidamente que no incluya información de credenciales en la URL a fin de validar la solicitud para ese servidor.

Gestión de identidad y acceso

AWS Identity and Access Management (IAM) es un Servicio de AWS que ayuda a los administradores a controlar de forma segura el acceso a los recursos de AWS. Los administradores de IAM controlan quién está autenticado (ha iniciado sesión) y autorizado (tiene permisos) para utilizar recursos de AWS. IAM es un Servicio de AWS que se puede utilizar sin cargo adicional.

Temas

- [Público](#)
- [Autenticación con identidades](#)
- [Administración del acceso con políticas](#)
- [Cómo funcionan los Servicios de AWS con IAM](#)
- [Solución de problemas de identidades y accesos en AWS](#)

Público

La forma en que utilice AWS Identity and Access Management (IAM) difiere en función del trabajo que realice en AWS.

Usuario de servicio: si utiliza Servicios de AWS para realizar el trabajo, el administrador le proporciona las credenciales y los permisos que necesita. A medida que utilice más características de AWS para realizar su trabajo, es posible que necesite permisos adicionales. Entender cómo se administra el acceso puede ayudarlo a solicitar los permisos correctos al administrador. Si no puede acceder a una característica en AWS, consulte [Solución de problemas de identidades y accesos en AWS](#) o la Guía del usuario del Servicio de AWS que esté usando.

Administrador de servicio: si está a cargo de los recursos de AWS en su empresa, probablemente tenga acceso completo a AWS. Su trabajo consiste en determinar a qué características y recursos de AWS deben acceder los usuarios del servicio. A continuación, debe enviar solicitudes a su administrador de IAM para cambiar los permisos de los usuarios de sus servicios. Revise la información de esta página para conocer los conceptos básicos de IAM. Para obtener más información sobre cómo la empresa puede utilizar IAM con AWS, consulte la Guía del usuario del Servicio de AWS que esté usando.

Administrador de IAM: si es un administrador de IAM, es posible que quiera conocer más detalles sobre cómo escribir políticas para administrar el acceso a AWS. Para consultar ejemplos de políticas basadas en la identidad de AWS que puede utilizar en IAM, consulte la Guía del usuario del Servicio de AWS que esté usando.

Autenticación con identidades

La autenticación es la manera de iniciar sesión en AWS mediante credenciales de identidad. Debe autenticarse como el Usuario raíz de la cuenta de AWS, como un usuario de IAM o asumiendo un rol de IAM.

Se puede iniciar sesión como una identidad federada con las credenciales de un origen de identidad, como AWS IAM Identity Center (IAM Identity Center), la autenticación de inicio de sesión único o las credenciales de Google/Facebook. Para obtener más información sobre el inicio de sesión, consulte [Cómo iniciar sesión en la Cuenta de AWS](#) en la Guía del usuario de AWS Sign-In.

Para el acceso mediante programación, AWS proporciona un SDK y una CLI para firmar criptográficamente las solicitudes. Para obtener más información, consulte [AWS Signature Version 4 para solicitudes de API](#) en la Guía del usuario de IAM.

Usuario raíz de la Cuenta de AWS

Cuando se crea una Cuenta de AWS, se comienza con una identidad de inicio de sesión denominada usuario raíz de la Cuenta de AWS que tiene acceso completo a todos los Servicios de AWS y recursos. Se recomienda encarecidamente que no utilice el usuario raíz para las tareas diarias. Para ver las tareas que requieren credenciales de usuario raíz, consulte [Tareas que requieren credenciales de usuario raíz](#) en la Guía del usuario de IAM.

Identidad federada

Como práctica recomendada, exija a los usuarios humanos que utilicen la federación con un proveedor de identidades para acceder a Servicios de AWS con credenciales temporales.

Una identidad federada es un usuario del directorio empresarial, proveedor de identidad web o Directory Service que accede a los Servicios de AWS mediante credenciales de un origen de identidad. Las identidades federadas asumen roles que proporcionan credenciales temporales.

Para una administración de acceso centralizada, se recomienda AWS IAM Identity Center. Para obtener más información, consulte [¿Qué es el Centro de identidades de IAM?](#) en la Guía del usuario de AWS IAM Identity Center.

Usuarios y grupos de IAM

Un [usuario de IAM](#) es una identidad con permisos específicos para una sola persona o aplicación. Recomendamos el uso de credenciales temporales en lugar de usuarios de IAM con credenciales de larga duración. Para obtener más información, consulte [Exigir que los usuarios humanos utilicen la federación con un proveedor de identidades para acceder a AWS con credenciales temporales](#) en la Guía del usuario de IAM.

Un [grupo de IAM](#) especifica un conjunto de usuarios de IAM y facilita la administración de los permisos para grupos grandes de usuarios. Para obtener más información, consulte [Casos de uso para usuarios de IAM](#) en la Guía del usuario de IAM.

Roles de IAM

Un [Rol de IAM](#) es una identidad con permisos específicos que proporciona credenciales temporales. Se puede asumir un rol [cambiando de un usuario a un rol de IAM \(consola\)](#) o llamando a una operación de la API de la AWS CLI o AWS. Para obtener más información, consulte [Métodos para asumir un rol](#) en la Guía del usuario de IAM.

Los roles de IAM son útiles para el acceso de usuario federado, los permisos de usuario de IAM temporales, el acceso entre cuentas, el acceso entre servicios y las aplicaciones que se ejecutan en Amazon EC2. Para obtener más información, consulte [Acceso a recursos entre cuentas en IAM](#) en la Guía del usuario de IAM.

Administración del acceso con políticas

Para controlar el acceso en AWS, se crean políticas y se adjuntan a identidades o recursos de AWS. Una política define los permisos cuando se asocia a una identidad o un recurso. AWS evalúa estas políticas cuando una entidad principal realiza una solicitud. La mayoría de las políticas se almacenan en AWS como documentos JSON. Para obtener más información sobre los documentos de políticas de JSON, consulte [Información general de políticas de JSON](#) en la Guía del usuario de IAM.

Mediante las políticas, los administradores especifican quién tiene acceso a qué, definiendo qué entidad principal puede realizar acciones sobre qué recursos y en qué condiciones.

De forma predeterminada, los usuarios y los roles no tienen permisos. Un administrador de IAM crea políticas de IAM y las agrega a roles, que los usuarios pueden asumir posteriormente. Las políticas de IAM definen permisos independientemente del método que se utilice para realizar la operación.

Políticas basadas en identidades

Las políticas basadas en identidad son documentos de política de permisos JSON que asocia a una identidad (usuario, grupo o rol). Estas políticas controlan qué acciones pueden realizar las identidades, en qué recursos y en qué condiciones. Para obtener más información sobre cómo crear una política basada en la identidad, consulte [Definición de permisos de IAM personalizados con políticas administradas por el cliente](#) en la Guía del usuario de IAM.

Las políticas basadas en identidad pueden ser políticas insertadas (incrustadas directamente en una sola identidad) o políticas administradas (políticas independientes asociadas a varias identidades). Para obtener información sobre cómo elegir entre políticas administradas e insertadas, consulte [Selección entre políticas administradas y políticas insertadas](#) en la Guía del usuario de IAM.

Políticas basadas en recursos

Las políticas basadas en recursos son documentos de políticas JSON que se asocian a un recurso. Los ejemplos incluyen las Políticas de confianza de roles de IAM y las Políticas de bucket de Amazon S3. En los servicios que admiten políticas basadas en recursos, los administradores de servicios pueden utilizarlos para controlar el acceso a un recurso específico. Debe [especificar una entidad principal](#) en una política basada en recursos.

Las políticas basadas en recursos son políticas insertadas que se encuentran en ese servicio. No se puede utilizar políticas de IAM administradas de AWS en una política basada en recursos.

Listas de control de acceso (ACL)

Las listas de control de acceso (ACL) controlan qué entidades principales (miembros de cuentas, usuarios o roles) tienen permisos para acceder a un recurso. Las ACL son similares a las políticas basadas en recursos, aunque no utilizan el formato de documento de políticas JSON.

Amazon S3, AWS WAF y Amazon VPC son ejemplos de servicios que admiten las ACL. Para obtener más información sobre las ACL, consulta [Información general de Lista de control de acceso \(ACL\)](#) en la Guía para desarrolladores de Amazon Simple Storage Service.

Otros tipos de políticas

AWS admite tipos de políticas adicionales que pueden establecer el máximo de permisos concedidos por los tipos de políticas más frecuentes:

- Límites de permisos: establecen los permisos máximos que una política basada en identidad puede conceder a una entidad de IAM. Para obtener más información, consulte [Límites de permisos para las entidades de IAM](#) en la Guía del usuario de IAM.
- Políticas de control de servicios (SCP): especifican los permisos máximos para una organización o unidad organizativa en AWS Organizations. Para obtener más información, consulte [Políticas de control de servicios](#) en la Guía del usuario de AWS Organizations.
- Políticas de control de recursos (RCP): definen los permisos máximos disponibles para los recursos de las cuentas. Para obtener más información, consulte [Políticas de control de recursos \(RCP\)](#) en la Guía del usuario de AWS Organizations.
- Políticas de sesión: políticas avanzadas que se pasan como parámetro cuando se crea una sesión temporal para un rol o un usuario federado. Para obtener más información, consulte [Políticas de sesión](#) en la Guía del usuario de IAM.

Varios tipos de políticas

Cuando se aplican varios tipos de políticas a una solicitud, los permisos resultantes son más complicados de entender. Para obtener información acerca de cómo AWS decide si permitir o no una solicitud cuando hay varios tipos de políticas implicados, consulte [Lógica de evaluación de políticas](#) en la Guía del usuario de IAM.

Cómo funcionan los Servicios de AWS con IAM

Para obtener una perspectiva general sobre cómo funcionan los Servicios de AWS con la mayoría de las características de IAM, consulte [Servicios de AWS que funcionan con IAM](#) en la Guía del usuario de IAM.

Para obtener información sobre cómo usar un Servicio de AWS específico con IAM, consulte la sección de seguridad de la Guía del usuario del servicio correspondiente.

Solución de problemas de identidades y accesos en AWS

Utilice la siguiente información para diagnosticar y solucionar los problemas comunes que puedan surgir cuando trabaje con AWS e IAM.

Temas

- [No tengo autorización para realizar una acción en AWS](#)
- [No tengo autorización para realizar la operación iam:PassRole](#)

- [Quiero permitir a personas externas a mi Cuenta de AWS el acceso a mis recursos de AWS](#)

No tengo autorización para realizar una acción en AWS

Si recibe un error que indica que no tiene autorización para realizar una acción, las políticas se deben actualizar para permitirle realizar la acción.

En el siguiente ejemplo, el error se produce cuando el usuario de IAM `mateojackson` intenta utilizar la consola para consultar los detalles acerca de un recurso ficticio `my-example-widget`, pero no tiene los permisos ficticios `awes:GetWidget`.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
awes:GetWidget on resource: my-example-widget
```

En este caso, la política del usuario `mateojackson` debe actualizarse para permitir el acceso al recurso `my-example-widget` mediante la acción `awes:GetWidget`.

Si necesita ayuda, contacte con su administrador de AWS. El administrador es la persona que le proporcionó las credenciales de inicio de sesión.

No tengo autorización para realizar la operación iam:PassRole

Si recibe un error que indica que no tiene autorización para realizar la acción `iam:PassRole`, las políticas deben actualizarse a fin de permitirle pasar un rol a AWS.

Algunos Servicios de AWS le permiten transferir un rol existente a dicho servicio en lugar de crear un nuevo rol de servicio o uno vinculado al servicio. Para ello, debe tener permisos para transferir el rol al servicio.

En el siguiente ejemplo, el error se produce cuando un usuario de IAM denominado `marymajor` intenta utilizar la consola para realizar una acción en AWS. Sin embargo, la acción requiere que el servicio cuente con permisos que otorguen un rol de servicio. Mary no tiene permisos para transferir el rol al servicio.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
iam:PassRole
```

En este caso, las políticas de Mary se deben actualizar para permitirle realizar la acción `iam:PassRole`.

Si necesita ayuda, póngase en contacto con su administrador de AWS. El administrador es la persona que le proporcionó las credenciales de inicio de sesión.

Quiero permitir a personas externas a mi Cuenta de AWS el acceso a mis recursos de AWS

Se puede crear un rol que los usuarios de otras cuentas o las personas externas a la organización puedan utilizar para acceder a sus recursos. Se puede especificar una persona de confianza para que asuma el rol. En el caso de los servicios que admitan las políticas basadas en recursos o las listas de control de acceso (ACL), puede utilizar dichas políticas para conceder a las personas acceso a sus recursos.

Para obtener más información, consulte lo siguiente:

- Para obtener información acerca de si AWS admite estas características, consulte [Cómo funcionan los Servicios de AWS con IAM](#).
- Para obtener información acerca de cómo proporcionar acceso a los recursos de las Cuenta de AWS de su propiedad, consulte [Proporcionar acceso a un usuario de IAM a otra Cuentas de AWS de la que es propietario](#) en la Guía del usuario de IAM.
- Para obtener información acerca de cómo proporcionar acceso a sus recursos a Cuentas de AWS de terceros, consulte [Proporcionar acceso a Cuentas de AWS que son propiedad de terceros](#) en la Guía del usuario de IAM.
- Para obtener información sobre cómo proporcionar acceso mediante una federación de identidades, consulte [Proporcionar acceso a usuarios autenticados externamente \(identidad federada\)](#) en la Guía del usuario de IAM.
- Para conocer sobre la diferencia entre las políticas basadas en roles y en recursos para el acceso entre cuentas, consulte [Acceso a recursos entre cuentas en IAM](#) en la Guía del usuario de IAM.

Validación de la conformidad de este producto o servicio de AWS

Para saber si un Servicio de AWS está incluido en el ámbito de programas de conformidad específicos, consulte [Servicios de AWS incluidos por programa de conformidad](#) y elija el programa de conformidad que le interese. Para obtener información general, consulte [Programas de conformidad de AWS](#).

Puede descargar los informes de auditoría de terceros utilizando AWS Artifact. Para obtener más información, consulte [Descarga de informes en AWS Artifact](#).

Su responsabilidad de conformidad al utilizar Servicios de AWS se determina en función de la confidencialidad de los datos, los objetivos de conformidad de la empresa, así como de la legislación y los reglamentos aplicables. Para obtener más información sobre su responsabilidad en cuanto a la conformidad al utilizar los Servicios de AWS, consulte [Documentación de seguridad de AWS](#).

Este producto o servicio de AWS sigue el [modelo de responsabilidad compartida](#) a través de los servicios específicos de Amazon Web Services (AWS) que admite. Para obtener información sobre la seguridad de los servicios de AWS, consulte la [página de documentación sobre la seguridad de los servicios de AWS](#) y los [servicios de AWS sujetos a las medidas de conformidad de AWS de cada programa de conformidad](#).

Resiliencia de este producto o servicio de AWS

La infraestructura global de AWS está conformada por Regiones de AWS y zonas de disponibilidad.

Las Regiones de AWS proporcionan varias zonas de disponibilidad físicamente independientes y aisladas que se encuentran conectadas mediante redes con un alto nivel de rendimiento y redundancia, además de baja latencia.

Con las zonas de disponibilidad, puede diseñar y utilizar aplicaciones y bases de datos que realizan una conmutación por error automática entre las zonas sin interrupciones. Las zonas de disponibilidad tienen una mayor disponibilidad, tolerancia a errores y escalabilidad que las infraestructuras tradicionales de centros de datos únicos o múltiples.

Para obtener más información sobre las regiones y zonas de disponibilidad de AWS, consulte [Infraestructura global de AWS](#).

Este producto o servicio de AWS sigue el [modelo de responsabilidad compartida](#) a través de los servicios específicos de Amazon Web Services (AWS) que admite. Para obtener información sobre la seguridad de los servicios de AWS, consulte la [página de documentación sobre la seguridad de los servicios de AWS](#) y los [servicios de AWS sujetos a las medidas de conformidad de AWS de cada programa de conformidad](#).

Seguridad de la infraestructura de este producto o servicio de AWS

Este producto o servicio de AWS utiliza servicios administrados y, por lo tanto, está protegido por la seguridad de red global de AWS. Para obtener información sobre los servicios de seguridad de AWS y sobre cómo AWS protege la infraestructura, consulte [Seguridad en la nube de AWS](#). Para diseñar

su entorno de AWS siguiendo las prácticas recomendadas de seguridad de infraestructura, consulte [Protección de la infraestructura](#) en Portal de seguridad de AWS Well-Architected Framework.

Puede utilizar llamadas a la API publicadas en AWS para acceder a este producto o servicio de AWS a través de la red. Los clientes deben admitir lo siguiente:

- Seguridad de la capa de transporte (TLS). Exigimos TLS 1.2 y recomendamos TLS 1.3.
- Conjuntos de cifrado con confidencialidad directa total (PFS) como DHE (Ephemeral Diffie-Hellman) o ECDHE (Elliptic Curve Ephemeral Diffie-Hellman). La mayoría de los sistemas modernos como Java 7 y posteriores son compatibles con estos modos.

Además, las solicitudes deben estar firmadas mediante un ID de clave de acceso y una clave de acceso secreta que esté asociada a una entidad principal de IAM. También puedes utilizar [AWS Security Token Service](#) (AWS STS) para generar credenciales de seguridad temporales para firmar solicitudes.

Este producto o servicio de AWS sigue el [modelo de responsabilidad compartida](#) a través de los servicios específicos de Amazon Web Services (AWS) que admite. Para obtener información sobre la seguridad de los servicios de AWS, consulte la [página de documentación sobre la seguridad de los servicios de AWS](#) y los [servicios de AWS sujetos a las medidas de conformidad de AWS de cada programa de conformidad](#).

Aplicación de una versión mínima de TLS

Para aumentar la seguridad al comunicarse con los servicios de AWS, configure AWS SDK for JavaScript para usar TLS 1.2 o una versión posterior.

Transport Layer Security (TLS) es un protocolo que utilizan los navegadores web y otras aplicaciones para garantizar la privacidad e integridad de los datos intercambiados a través de una red.

Important

A partir del 10 de junio de 2024, [anunciamos](#) que TLS 1.3 está disponible en los puntos de conexión de la API de servicio de AWS en cada una de las regiones de AWS. AWS SDK para JavaScript v2 no negocia la versión de TLS por sí mismo. En su lugar, utiliza la versión de TLS determinada por Node.js, que se puede configurar mediante `https.Agent`. AWS recomienda utilizar la versión de LTS activa actual de Node.js.

Verificar y aplicar TLS en Node.js

Cuando se utiliza AWS SDK for JavaScript con Node.js, la capa de seguridad de Node.js subyacente se utiliza para establecer la versión de TLS.

Node.js 12.0.0 y las versiones posteriores utilizan una versión mínima de OpenSSL 1.1.1, que admite TLS 1.3. AWS SDK para JavaScript v2 usa TLS 1.3 de forma predeterminada cuando está disponible, pero usa una versión inferior si es necesaria de forma predeterminada.

Verificar la versión de OpenSSL y TLS

Para obtener la versión de OpenSSL que usa Node.js en su equipo, ejecute el siguiente comando.

```
node -p process.versions
```

La versión de OpenSSL de la lista es la versión que utiliza Node.js, como se muestra en el siguiente ejemplo.

```
openssl: '1.1.1b'
```

Para obtener la versión de TLS que usa Node.js en su equipo, inicie el shell de Node y ejecute los siguientes comandos, en orden.

```
> var tls = require("tls");  
> var tlsSocket = new tls.TLSocket();  
> tlsSocket.getProtocol();
```

El último comando genera la versión de TLS, como se muestra en el siguiente ejemplo.

```
'TLSv1.3'
```

Node.js utiliza de forma predeterminada esta versión de TLS e intenta negociar otra versión de TLS si una llamada no se realiza correctamente.

Comprobación de las versiones mínima y máxima de TLS compatibles

Los desarrolladores pueden comprobar las versiones mínima y máxima de TLS compatibles en Node.js mediante el siguiente script:

```
var tls = require("tls");
```

```
console.log("Supported TLS versions:", tls.DEFAULT_MIN_VERSION + " to " +  
  tls.DEFAULT_MAX_VERSION);
```

El último comando genera la versión de TLS mínima y máxima predeterminada, como se muestra en el siguiente ejemplo.

```
Supported TLS versions: TLSv1.2 to TLSv1.3
```

Aplicar una versión mínima de TLS

Node.js negocia una versión de TLS cuando se produce un error en una llamada. Puede aplicar la versión de TLS mínima permitida durante esta negociación, al ejecutar un script desde la línea de comandos o a petición en el código de JavaScript.

Para especificar la versión mínima de TLS desde la línea de comandos, debe utilizar Node.js versión 11.4.0 o posterior. Para instalar una versión específica de Node.js, primero instale Node Version Manager (nvm) siguiendo los pasos que se encuentran en la información sobre [instalación y actualización de Node Version Manager](#). A continuación, ejecute los siguientes comandos para instalar y usar una versión específica de Node.js.

```
nvm install 11  
nvm use 11
```

Enforcing TLS 1.2

Para forzar que TLS 1.2 sea la versión mínima permitida, especifique el argumento `--tls-min-v1.2` al ejecutar el script, como se muestra en el siguiente ejemplo.

```
node --tls-min-v1.2 yourScript.js
```

Para especificar la versión de TLS mínima permitida para una solicitud específica en el código de JavaScript, utilice el parámetro `httpOptions` para especificar el protocolo, como se muestra en el ejemplo siguiente.

```
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 forzar que TLS 1.3 sea la versión mínima permitida, especifique el argumento `--tls-min-v1.3` al ejecutar el script, como se muestra en el siguiente ejemplo.

```
node --tls-min-v1.3 yourScript.js
```

Para especificar la versión de TLS mínima permitida para una solicitud específica en el código de JavaScript, utilice el parámetro `httpOptions` para especificar el protocolo, como se muestra en el ejemplo siguiente.

```
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 y aplicar TLS en un script de navegador

Al utilizar el SDK para JavaScript en un script de navegador, la configuración del navegador controla la versión de TLS que se utiliza. La versión de TLS que utiliza el navegador no se puede detectar ni

establecer mediante script y el usuario debe configurarla. Para verificar y aplicar la versión de TLS que se utiliza en un script de navegador, consulte las instrucciones de su navegador específico.

Microsoft Internet Explorer

1. Abra Internet Explorer.
2. En la barra de menús, seleccione la pestaña Herramientas ▶ Opciones de Internet ▶ Avanzadas.
3. Desplácese hacia abajo hasta la categoría Seguridad y marque manualmente la casilla de opción Usar TLS 1.2.
4. Haga clic en OK (Aceptar).
5. Cierre el navegador y reinicie Internet Explorer.

Microsoft Edge

1. En el cuadro de búsqueda del menú de Windows, escriba *Opciones de Internet*.
2. En Mejor coincidencia, haga clic en Opciones de Internet.
3. En la ventana Propiedades de Internet, en la pestaña Avanzadas, desplácese hacia abajo hasta la sección Seguridad.
4. Marque la casilla de verificación Usar TLS 1.2.
5. Haga clic en OK (Aceptar).

Google Chrome

1. Abra Google Chrome.
2. Haga clic en Alt F y seleccione Configuración.
3. Desplácese hacia abajo y seleccione Mostrar configuración avanzada...
4. Desplácese hasta la sección Sistema y haga clic en Abrir configuración de proxy...
5. Seleccione la pestaña Avanzado.
6. Desplácese hacia abajo hasta la categoría Seguridad y marque manualmente la casilla de opción Usar TLS 1.2.
7. Haga clic en OK (Aceptar).
8. Cierre el navegador y reinicie Google Chrome.

Mozilla Firefox

1. Abra Firefox.
2. En la barra de direcciones, escriba `about:config` y pulse Intro.
3. En el campo Buscar, escriba `tls`. Busque la entrada `security.tls.version.min` y haga doble clic en ella.
4. Establezca el valor entero en 3 para hacer que el protocolo de TLS 1.2 sea el predeterminado.
5. Haga clic en OK (Aceptar).
6. Cierre el navegador y reinicie Mozilla Firefox.

Apple Safari

No hay opciones para activar protocolos SSL. Si utiliza la versión 7 o superior de Safari, TLS 1.2 se habilita automáticamente.

Recursos adicionales

Los siguientes enlaces proporcionan recursos adicionales que puede usar con [AWS SDK para JavaScript](#).

Guía de referencia de las herramientas y los SDK de AWS

La [Guía de referencia de las herramientas y los SDK de AWS](#) también contiene configuraciones, características y otros conceptos fundamentales comunes a muchos de los SDK de AWS.

Foro de SDK para JavaScript

Encontrará preguntas y debates sobre temas de interés para los usuarios del SDK para JavaScript en el [Foro de SDK para JavaScript](#).

SDK para JavaScript y guía para desarrolladores en GitHub

Existen varios repositorios en GitHub para SDK para JavaScript.

- El SDK para JavaScript actual está disponible en el [repositorio de SDK](#).
- La Guía para desarrolladores de SDK para JavaScript (el presente documento) está disponible en formato Markdown en su propio [repositorio de la documentación](#).
- Parte del código de muestra incluido en esta guía está disponible en el [repositorio del código de muestra del SDK](#).

SDK de JavaScript en Gitter

También puede encontrar preguntas y debates acerca de SDK para JavaScript en la comunidad de [SDK para JavaScript](#) en Gitter.

Historial de documento de AWS SDK para JavaScript

- Versión del SDK: consulte [Referencia de la API de JavaScript](#)
- Última actualización importante de la documentación: 31 de marzo de 2022

Historial de documentos

En la siguiente tabla se describen los cambios importantes de cada versión de la AWS SDK para JavaScript posteriores a mayo de 2018. Para obtener notificaciones sobre las actualizaciones de esta documentación, puede suscribirse a una [fuente RSS](#).

Cambio	Descripción	Fecha
Ahora se admite TLS 1.3 en todos los puntos de conexión de la API de servicio de AWS en todas las regiones	Se han actualizado la versión de TLS compatible y el método para registrar la versión de TLS.	10 de abril de 2025
Aplicación de una versión mínima de TLS	Se ha añadido información sobre TLS 1.3.	31 de marzo de 2022
Visualización de fotos en un bucket de Amazon S3 desde un navegador	Se ha añadido un ejemplo que muestra cómo ver fotos en álbumes de fotos existentes.	13 de mayo de 2019
Configuración de las credenciales en Node.js, nuevas opciones de carga de credenciales	Se ha añadido información acerca de las credenciales que se cargan desde el proveedor de credenciales de ECS o un proceso de credenciales configurado.	25 de abril de 2019
Credenciales con un proceso de credenciales configurado	Se ha añadido información acerca de las credenciales que se cargan desde	25 de abril de 2019

un proceso de credenciales configurado.

[Nueva versión de Introducción a los script de navegador](#)

Se ha reformulado la sección Introducción a los script de navegador para simplificar el ejemplo y para obtener acceso al servicio Amazon Polly para enviar texto y devolver voz sintetizada que se puede reproducir en el navegador . Consulte la sección de [introducción en un script de navegador](#) para ver el nuevo contenido.

14 de julio de 2018

[Nueva versión de Muestras de código de Amazon SNS](#)

Se han añadido cuatro nuevas muestras de código Node.js para trabajar con Amazon SNS. Consulte [Ejemplos de Amazon SNS](#) para ver el código de muestra.

29 de junio de 2018

[Nueva versión de Introducción a Node.js](#)

Introducción a Node.js se ha reescrito para utilizar código de muestra actualizado y para dar más detalles sobre cómo crear el archivo `package.json`, así como el mismo código Node.js. Consulte la sección de [introducción a Node.js](#) para ver el nuevo contenido.

4 de junio de 2018

Actualizaciones anteriores

En la siguiente tabla se describen los cambios importantes de cada versión de la AWS SDK para JavaScript anteriores a junio de 2018.

Cambio	Descripción	Fecha
Nuevas muestras de código de AWS Elemental MediaConvert	Se han añadido tres muestras nuevas de código de Node.js con AWS Elemental MediaConvert. Consulte AWS Elemental MediaConvert Ejemplos de para ver el código de muestra.	21 de mayo de 2018
Nueva edición en el botón de GitHub	Ahora, en el encabezado de cada tema se ofrece un botón que le lleva a la versión de Markdown del mismo tema en GitHub de forma que pueda realizar ediciones para mejorar la exactitud y la exhaustividad de la guía.	21 de febrero de 2018
Nuevo tema sobre puntos de enlace personalizados	Se ha añadido información sobre el formato y el uso de puntos de enlace personalizados para realizar llamadas a la API. Consulte Especificación de puntos de enlace personalizados .	20 de febrero de 2018
SDK para JavaScript y Guía para desarrolladores en GitHub	La guía para desarrolladores del SDK para JavaScript está disponible en formato Markdown en su propio repositorio de la documentación . Puede publicar	16 de febrero de 2018

Cambio	Descripción	Fecha
	problemas que le interesaría que la guía aborde o enviar solicitudes de extracción para enviar cambios propuestos.	
Nuevo código de muestra de Amazon DynamoDB	Se ha añadido un nuevo código de muestra Node.js para actualizar una tabla de DynamoDB mediante el cliente de documentos. Consulte Uso del cliente de documentos de DynamoDB para ver el código de muestra.	14 de febrero de 2018
Nuevo tema sobre el registro de SDK	Se ha añadido un tema donde se describe cómo registrar llamadas a la API con el SDK para JavaScript; se incluye información acerca del uso de un registrador externo. Consulte Registro de llamadas del AWS SDK para JavaScript .	5 de febrero de 2018
Tema actualizado sobre la configuración de la región	Se ha actualizado y ampliado el tema que describe cómo configurar la región que se utiliza con el SDK, incluida información sobre el orden de prioridad para configurar la región. Consulte Configuración de la región de AWS .	12 de diciembre de 2017

Cambio	Descripción	Fecha
Nuevos ejemplos de código para Amazon SES	Se ha actualizado la sección con ejemplos de código de SDK, que incluyen cinco nuevos ejemplos para trabajar con Amazon SES. Para obtener más información sobre estos ejemplos de código, consulte Ejemplos de Amazon Simple Email Service .	9 de noviembre de 2017

Cambio	Descripción	Fecha
Mejoras de uso	<p>Basándose en pruebas de uso recientes, se han introducido varios cambios para mejorar el uso de la documentación.</p> <ul style="list-style-type: none">• El destino de las muestras de código, ya sea ejecución en navegador o en Node.js, se identifica más claramente.• Los enlaces a los temas ya no saltan inmediatamente a otro contenido web, como la referencia de la API.• En la introducción hay más enlaces a detalles sobre cómo obtener las credenciales de AWS.• Proporciona más información sobre características de Node.js comunes necesarias para utilizar el SDK. Para obtener más información, consulte Consideraciones de Node.js.	9 de agosto de 2017

Cambio	Descripción	Fecha
Nueva versión de Ejemplos de código de DynamoDB	Se ha actualizado la sección con ejemplos de código de SDK para reescribir los dos ejemplos anteriores, así como para añadir tres ejemplos recién creados para trabajar con DynamoDB. Para obtener más información sobre estos ejemplos de código, consulte Ejemplos de Amazon DynamoDB .	21 de junio de 2017
Nueva versión de Ejemplos de código de IAM	Se ha actualizado la sección con ejemplos de código de SDK para incluir cinco nuevos ejemplos para trabajar con IAM. Para obtener más información sobre estos ejemplos de código, consulte Ejemplos de AWS IAM .	23 de diciembre de 2016
Nuevos ejemplos de código de CloudWatch y Amazon SQS	Se ha actualizado la sección para incluir nuevos ejemplos de código de SDK para trabajar con CloudWatch y con Amazon SQS. Para obtener más información sobre estos ejemplos de código, consulte Ejemplos de Amazon CloudWatch y Ejemplos de Amazon SQS .	20 de diciembre de 2016

Cambio	Descripción	Fecha
Nuevos ejemplos de código de Amazon EC2	Se ha actualizado la sección para incluir cinco nuevos ejemplos de código de SDK para trabajar con Amazon EC2. Para obtener más información sobre estos ejemplos de código, consulte Ejemplos de Amazon EC2 .	15 de diciembre de 2016
La lista de navegadores compatibles está más visible	La lista de navegadores compatibles con el SDK para JavaScript, que anteriormente se encontraba en el tema sobre requisitos previos, tiene ahora su propio tema para que sea más visible en el índice.	16 de noviembre de 2016
Publicación inicial de la nueva guía del desarrollador	La guía del desarrollador anterior ya no está disponible. La nueva guía del desarrollador se ha reorganizado para que sea más fácil encontrar la información. Cuando haya escenarios de JavaScript de navegador o de Node.js con consideraciones especiales, dichas consideraciones se identificarán según sea más adecuado. La guía también proporciona ejemplos de código adicionales que están mejor organizados para que sea más fácil y más rápido encontrarlos.	28 de octubre de 2016