Working with Asynchronous Events

Here, you'll learn how to handle asynchronous calls and store output while using AWS Lambda functions.

WE'LL COVER THE FOLLOWING

The Lambda interface is always the same, regardless of the event source or type. The conversion function will expect the same two parameters as all your Lambda functions so far:

- the event
- the context object

But unlike everything else we've done so far in the course, the Lambda function won't be able to just return the result back to the client. That's because S3 invokes the Lambda function in the background, separate from the client request going through API Gateway.

Synchronous and asynchronous calls

AWS Lambda functions support two types of call, synchronous and asynchronous:

- Synchronous calls expect the caller to wait until the Lambda function completes. The function reports the result directly to the caller. API Gateway uses this method.

- Asynchronous calls complete immediately, and the Lambda function keeps running in the background. The function can't send the result directly to the caller (it's 'fire and forget'). S3 and most other platform services use this method.

services use this method.

Storing the output with asynchronous calls

With asynchronous calls, the function also needs to be responsible for storing the output somewhere. The usual way to achieve this is to somehow correlate outputs with inputs, for example including the target output location in the event parameters, or deriving the result location from inputs. In this case, an easy solution for that would be to use a different bucket for results and store the output using the same key as the input. Buckets are effectively namespaces for files, and two different files can have the same keys in different buckets. You'll just need to tell the function the name of the result bucket. For that, you can use an environment variable. You can create a new file for the main Lambda function, for example, `index.js`, in the conversion function source directory.

```
const path = require('path'),
  os = require('os'),
  s3Util = require('./s3-util'),
  extractS3Info = require('./extract-s3-info'),
  silentRemove = require('./silent-remove'),
  OUTPUT_BUCKET = process.env.OUTPUT_BUCKET,
  supportedFormats = ['jpg', 'jpeg', 'png', 'gif'];

exports.handler = async (event, context) => {
  const s3Info = extractS3Info(event),
    id = context.awsRequestId,
    extension = path.extname(s3Info.key).toLowerCase(),
    tempFile = path.join(os.tmpdir(), id + extension),
    extensionWithoutDot = extension.slice(1),
    contentType = `image/${extensionWithoutDot}`;
  console.log('converting', s3Info.bucket, ':', s3Info.key, 'using', tempFile);
  if (!supportedFormats.includes(extensionWithoutDot)) {
    throw new Error(`unsupported file type ${extension}`);
  }
  await s3Util.downloadFileFromS3(s3Info.bucket, s3Info.key, tempFile);
  // in the next chapter, we'll do something useful with the tempfile here...
  await s3Util.uploadFileToS3(OUTPUT_BUCKET, s3Info.key, tempFile, contentType);
  await silentRemove(tempFile);
};
```

code/ch9/image-conversion/index.js

This function requires another bucket to store results, so you can go ahead and add it to the template. The following lines will be added to the `Resources` section of your template (at the same indentation level as `UploadS3Bucket`).

```
ThumbnailsS3Bucket:
  Type: AWS::S3::Bucket
  Properties:
```

```
    BucketEncryption:
      ServerSideEncryptionConfiguration:

        - ServerSideEncryptionByDefault:
            SSEAlgorithm: AES256
```

Line 27 to Line 33 of code/ch9/template.yaml

The web API and the form display function don't need to change, but the confirm upload function should now generate download signatures for the results bucket. You can keep the same source code, but you'll need to reconfigure it with the results bucket. You can change the `ConfirmUploadFunction` configuration to update the bucket references (lines 16 and 19 in the next code listing).

```
ConfirmUploadFunction:
  Type: AWS::Serverless::Function
  Properties:
    CodeUri: user-form/
    Handler: confirm-upload.lambdaHandler
    Runtime: nodejs12.x
    Events:
      ConfirmForm:
        Type: Api
        Properties:
          Path: /confirm
          Method: get
          RestApiId: !Ref WebApi
    Environment:
      Variables:
        UPLOAD_S3_BUCKET: !Ref ThumbnailsS3Bucket
    Policies:
      - S3ReadPolicy:
          BucketName: !Ref ThumbnailsS3Bucket
```

Line 58 to Line 76 of code/ch9/template.yaml

Next, you need to add a new function for the file conversion. You can create a new `AWS::Serverless::Function` resource according to the following listing in the `Resources` section of your template (at the same indentation level as `ConfirmUploadFunction`).

```
ConvertFileFunction:
  Type: AWS::Serverless::Function
  Properties:
    CodeUri: image-conversion/
    Handler: index.handler
    Runtime: nodejs12.x
    Events:
      FileUploaded:
```

```
      Type: S3
      Properties:
        Bucket: !Ref UploadS3Bucket

        Events: s3:ObjectCreated:*
  Timeout: 600
  Environment:
    Variables:
      OUTPUT_BUCKET: !Ref ThumbnailsS3Bucket
  Policies:
    - S3FullAccessPolicy:
        BucketName: !Ref ThumbnailsS3Bucket
```

Line 77 to Line 95 of code/ch9/template.yaml

Points to note

Here are some important settings:

- `CodeUri` (line 4) points to the directory with the function source code. In the previous sections, you used `image-conversion`.
- `Handler` (line 5) specifies the module and the function name. Earlier in this section, it was suggested that you save the main Lambda code in the `index` module (`index.js` file), in a function called `handler`, so the full `Handler` value should be `index.handler`.
- This function needs to work on relatively large files, so remember to set the `Timeout` property (line 13), for example to 600 seconds.
- The source code expects the results bucket name in the `OUTPUT_BUCKET` environment variable, so you need to configure that as well in the template (lines 14-16). You can use `!Ref` to turn the internal bucket reference into the actual bucket name.
- You also need to let the function write to the results bucket (lines 17-19).

In the next lesson, you will learn how to avoid circular references.