Working with Files

In this lesson, you'll have a look at how to work with files when using AWS Lambda.

In Chapter 7, you used a method of the AWS S3 SDK to upload the contents of a string into an S3 bucket. Although you could potentially work with images as in-memory strings or buffers, this would unnecessarily increase the memory footprint of the Lambda function, costing more than it should. The S3 SDK for most languages supports working with file streams.

Our conversion function needs to download an S3 object into a local file so you can later produce thumbnails. It will also need to upload the resulting file contents to S3. You'll create two utility methods for that. You can save the following code as `s3-util.js` in the conversion function code directory.

```
const aws = require('aws-sdk'),
  fs = require('fs'),
  s3 = new aws.S3(),
  downloadFileFromS3 = function (bucket, fileKey, filePath) {
    console.log('downloading', bucket, fileKey, filePath);
    return new Promise((resolve, reject) => {
      const file = fs.createWriteStream(filePath),
        stream = s3.getObject({
          Bucket: bucket,
          Key: fileKey
        }).createReadStream();
      stream.on('error', reject);
      file.on('error', reject);
      file.on('finish', () => {
        console.log('downloaded', bucket, fileKey);
        resolve(filePath);
      });
      stream.pipe(file);
    });
  },
  uploadFileToS3 = function (bucket, fileKey, filePath, contentType) {
    console.log('uploading', bucket, fileKey, filePath);
    return s3.upload({
      Bucket: bucket,
      Key: fileKey,
      Body: fs.createReadStream(filePath),
      ACL: 'private',
      ContentType: contentType
    }).promise();
  };
```

```
module.exports = {
  downloadFileFromS3: downloadFileFromS3,

  uploadFileToS3: uploadFileToS3
};
```

code/ch9/image-conversion/s3-util.js

Whenever you create files in the Lambda container, you also need to consider cleaning up. Although Lambda function calls are independent, they are not stateless. Lambda can choose to reuse a container for the same function configuration. If you get a huge amount of constant traffic, Lambda will reuse the same container over a longer period of time. Your function will download S3 objects into the local file system, so if you never clean up, the local disk might run out of space. You can add a utility function to delete temporary files. Any problems cleaning up shouldn't really abort processing, so this function can safely ignore errors. You can save the code from the next listing into `silent-remove.js` in the conversion function code directory.

```
const util = require('util'),
  fs = require('fs'),
  removeAsync = util.promisify(fs.unlink);
module.exports = async function silentRemove(file) {
  try {
    await removeAsync(file);
  } catch (e) {
    // ignore error
  }
};
```

code/ch9/image-conversion/silent-remove.js

In the next lesson, you will write the code to handle the asynchronous events. See you there!