Implementation

In this exercise, you will have to implement a Lambda function that processes newly uploaded images, creates a smaller version of the same image, and uploads an image to a different S3 bucket. It is important to upload an image to a different bucket; otherwise, a smaller image will trigger your Lambda function again.

Add an event sources for the "ResizeImage" function

The function should be connected to the same SNS topic as the `SendUploadNotifications` function:

```
events:
  - sns:
      arn:
        Fn::Join:
          - ':'
          - - arn:aws:sns
            - Ref: AWS::Region
            - Ref: AWS::AccountId
            - ${self:custom.topicName}
      topicName: ${self:custom.topicName}
```

Import "S3EventRecord" type

It might be handy to import `S3EventRecord` type that represents a single S3 event.

To do this make sure that the first import in the lambda handler's file looks like this:

```
import { SNSEvent, SNSHandler, S3EventRecord } from 'aws-lambda'
```

Process SNS events in the Lambda function

To implement the function, you would have to process SNS events. To do this, you can use the following snippet:

```
export const handler: SNSHandler = async (event: SNSEvent) => {
  console.log('Processing SNS event ', JSON.stringify(event))
  for (const snsRecord of event.Records) {
    const s3EventStr = snsRecord.Sns.Message
    console.log('Processing S3 event', s3EventStr)
    const s3Event = JSON.parse(s3EventStr)

    for (const record of s3Event.Records) {
      // "record" is an instance of S3EventRecord
      await processImage(record) // A function that should resize ea
    }
  }
}
```

This snippet should be already familiar to you since we used it in this lesson.

Get a key of an uploaded image in S3

To download a newly uploaded image, we first need to get its key. To get it, we need to use the following code:

```
const key = record.s3.object.key
```

Download an image

Now when we have a key of a new S3 object, we can download it. To do this, you can use the following snippet:

```
const response = await s3
  .getObject({
    Bucket: bucketName,
    Key: key
  })
  .promise()

const body: Buffer = response.Body
```

Notice that here we need to read from an S3 bucket that contains original images.

The body of a downloaded object will be of a type `Buffer` which is used to work with an array of bytes. You can read more about this type here.

Resize an image

Now once we have a body of an image, we can resize it. To do this, we will use the `Jimp` library:

```
// Read an image with the Jimp library
const image = await Jimp.read(body)

// Resize an image maintaining the ratio between the image's width a
image.resize(150, Jimp.AUTO)

// Convert an image to a buffer that we can write to a different buc
const convertedBuffer = await image.getBufferAsync(Jimp.AUTO)
```

Write an image to a different bucket

Once we have a resized image, we can write it back to S3. To do this, we can do the following:

```
await s3
  .putObject({
    Bucket: thumbnailBucketName,
    Key: `${key}.jpeg`,
    Body: convertedBuffer
  })
  .promise()
```

Notice that in this case we need to write an image to a separate S3 bucket for thumbnail images.

A note on the image processing library

The main reason why we use the `Jimp` library in this exercise is because it is implemented in pure JavaScript and does not rely on any native dependencies (libraries compiled to machine code). Native libraries can be used with AWS Lambda, but they are tricker to build (especially if you are using a non-Linux environment). You can read more about building a native binary package for AWS Lambda here.

Deployment

Before you deploy an application, keep in mind that names of S3 buckets should be globally unique across all AWS users. If you don't give your S3 buckets unique names a deployment will fail.

To ensure that your S3 buckets have unique names add a random string to the end of S3 bucket names in the `serverless.yml` file. Let's say you want to add a random string `ab4fe`. You would need to change the following section like this:

```
  environment:
    IMAGES_S3_BUCKET: serverless-udagram-images-ab4fe-${self:provide
    THUMBNAILS_S3_BUCKET: serverless-udagram-thumbnail-ab4fe-${self:
```

To deploy the whole project, you need to run following commands:

```
npm install
serverless deploy -v
```

Make sure that the `serverless` command is installed and configured to use correct IAM credentials.

Preparations to test your function

To test your function you should do the following:

- Create a group
- Create an image
- Upload an image

You might do those steps using the React app provided with this exercise. Alternatively, you can use Postman or any other HTTP client.

Using React application

You need to configure and start the application. To configure your application, you need to go to `src/config.ts` file and change `apiEndpoint` to point to the API of your serverless application.

After the application is configured you can run the following commands:

```
npm install
npm run start
```

Then you need to go to the `localhost:3000` and you can use the application to upload a file.

Using Postman and AWS console

Create a group

After the deployment is finished, you need to create at least one group in the `Groups-dev` table. You can add the following item using the DynamoDB console:

```
{
    "id": "1",
    "name": "Dogs",
    "description": "Only dog images here!"
}
```

Create an image

Once a group is created you should be able to send a `POST` request to the following URL:

```
https://{{apiId}}.execute-api.us-east-1.amazonaws.com/dev/groups/1/images
```

As a body of the request, you can provide the following JSON:

```
{
    "title": "New image"
}
```

Notice that you need to replace `{{apiId}}` with an id of a deployed API (returned as the result of the `serverless` command). Also, make sure that a group with id `1` exists or provide an id of an existing group in the URL of a request you send.

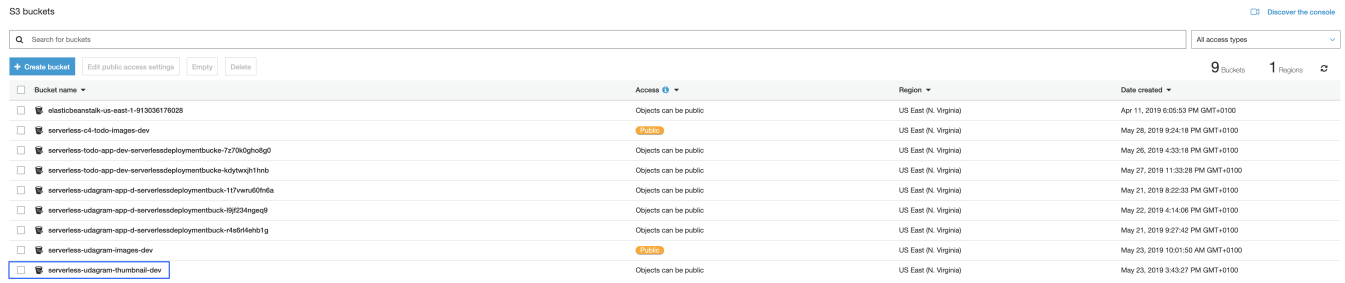
It should return an HTTP reply with a JSON object that contains field `uploadUrl`. This is a presigned URL we can use to upload an image.

Upload an image

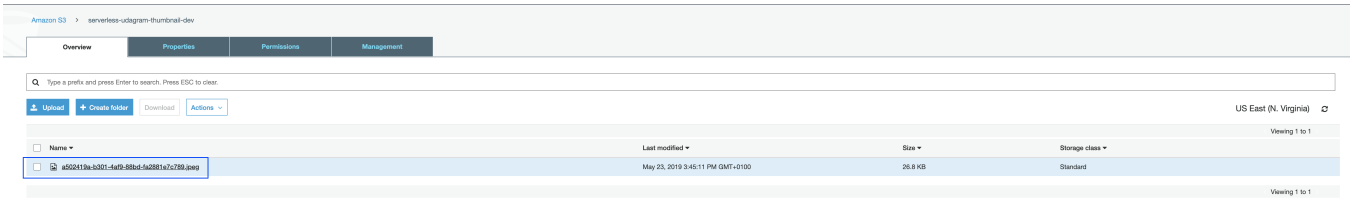
To upload an image, you need to send a `PUT` request to the URL returned on the previous step. A body of a request should be an image that you want to upload.

Expected result

Once an image is uploaded, you should see a resized image in the new S3 bucket. To find the uploaded image, first go to the S3 console and go to the bucket for generated thumbnails:



Click on the bucket, and you should see a new file for a generated thumbnail:



Download this file and check the function worked as expected. To do this, click on the checkbox near the file name and click on the `Download` button: