LESSON 3

In lesson 3 you are you going to begin building an image processor in AWS. Our slackbot will react to images uploaded in slack and replace all people's faces with smileys.

Over the course of the next few lessons we are going to:

- Allow users to upload images via Slack
- Save the uploaded image to an S3 bucket
- Detect faces in each image
- Determine the emotion of each face
- Replace the face with emoji
- Save a new photo to an S3 bucket
- Upload the new photo back to Slack
- Allow any user to recall any image via the botty

1. START YOUR ENGINES

Before you begin, create a copy of the Lesson 2 folder and name it Lesson 3. We are going to re-use a lot of the files.

2. EDIT SERVERLESS.YML

Over the next few steps you are going to create a new function that will orchestrate faceswap, create an S3 bucket for images and modify the existing echo function to download images files from Slack and in to an an S3 bucket. The process will be simple.

Step 1: When a user uploads a photo, Slack will invoke our Lambda function, this Lambda function will grab the image from Slack and upload it to an S3 bucket.

Step 2: That will kick of an S3 event that will automatically trigger a different Lambda function to process the image and swap faces with smileys based on people's look. There are a few more steps after but let's get this done first.

- Open serverless.yml and make sure it matches the following code snippet

```yaml
service: serverless-chatbot

provider:
  name: aws
  runtime: nodejs6.10
  profile: serverless-chatbot
  stage: dev
  iamRoleStatements:
    - Effect: "Allow"
      Action:
        - s3:GetObject
        - s3:PutObject
      Resource: arn:aws:s3:::${self:custom.uploadBucket}/*
custom:
  uploadBucket: ${self:service}-${self:provider.stage}-uploads
functions:
  hello:
    handler: handler.endpoint
    events:
      - http:
          path: echo
          method: post
    environment:
      POST_MESSAGE_URL: 'https://slack.com/api/chat.postMessage?'
      BOT_ACCESS_TOKEN: 'xoxb-159279836768-FOst5DLfEzmQgkz7cte5qiIv'
      BOT_ID: 'U4P87QLNL'
      VERIFICATION_TOKEN: 'wiM5uBXy2CcvIfTnyIdfhoVw'
  faceswap:
    handler: faceswap.execute
    events:
      - s3:
          bucket: ${self:custom.uploadBucket}
          event: s3:ObjectCreated:*
```

In the above code snipet we've added a faceswap function, a custom bucket name and explicitly specified a stage (dev as opposed to production). The process of deploying this severless.yml will automatically create the required function and S3 bucket and connect them together.

- If you look at the above code, you'll see that the bucket name has **YOURINITIALS** at the end. Bucket names are globally unique so we recommend putting your initials or a string of random characters at the very end. That will avoid clashes with other people.

3. EDIT HANDLER.JS

You will now repurpose handler.js. In the previous lesson the Lambda function had a very simple job. It would echo back anything that was written in Slack. Now it's going to do something a little bit more advanced. It will take any image file uploaded to Slack and then upload it to an S3 bucket we specified in serverless.yml (in the previous step).

- Open **handler.js** and replace its contents with the following code.

```
'use strict';

const https = require('https');
const fs = require('fs');
const aws = require('aws-sdk');
const qs = require('querystring');
const s3 = new aws.S3();

const uploadToBucket = function(filename) {
    var bodystream = fs.createReadStream(process.env.TEMP_FOLDER + filename);

    return new Promise((resolve, reject) => {
        s3.putObject({
           Bucket: process.env.UPLOAD_BUCKET,
           Key: filename,
           Body: bodystream
        }, function(error, data){
           if (error){
            return reject(error);
           }
           return resolve();
        });
    });
};

const downloadFileToSystem = function(path, filename) {
    var file = fs.createWriteStream(process.env.TEMP_FOLDER + filename);

    const options = {
        hostname: process.env.SLACK_HOSTNAME,
        path: path,
        headers: {
            authorization: 'Bearer ' + process.env.BOT_ACCESS_TOKEN
        }
    };

    return new Promise((resolve, reject) => {
        const request = https.get(options, (response) => {

            if (response.statusCode < 200 || response.statusCode > 299) {
                reject(new Error('Failure downloading file: ' + response.statusCode));
            }

            response.pipe(file);

            file.on('finish', function() {
                file.close(() => resolve());
            });
        });

        request.on('error', (err) => reject(err))
    })
};

const updateStatusInSlack = function(filename, channel) {
  return new Promise((resolve, reject) => {

    const response = {
        token: process.env.BOT_ACCESS_TOKEN,
        channel: channel,
        text: 'I am working on ' + filename + '... should be done soon.'
      }

      const URL = process.env.POST_MESSAGE_URL + qs.stringify(response);

      https.get(URL, (res) => {
        const statusCode = res.statusCode;
        resolve();
      })
  });
}

module.exports.endpoint = (event, context, callback) => {
  const request = JSON.parse(event.body);

  if (request.event.type && request.event.type === 'message' &&
      request.event.subtype && request.event.subtype === 'file_share') {

    const path = request.event.file.url_private_download;
    const filename = request.event.file.name;
    const channel = request.event.channel;

    downloadFileToSystem(path, filename)
      .then(() => uploadToBucket(filename))
      .then(() => updateStatusInSlack(filename, channel))
      .then(() => callback(null, {statusCode: 200}))
      .catch(() => callback(null, {statusCode: 500}));

      return;
  }

  callback(null, {statusCode: 200});
};
```

This code does a few simple things:

- It detects when a file has been shared.
- It then downloads the file to the local filesystem.
- Then it uploads the file to an S3 bucket.
- And, then it writes a message to Slack.

You might also notice that this code refers to process.env.BOT_ACCESS_TOKEN and process.env.TEMP_FOLDER. As we've learnt previously, these are environment variables and we need to add them. Modify the **hello** function definition in **serverless.yml** so that it looks like the following.

```
functions:
  hello:
    handler: handler.endpoint
    events:
      - http:
          path: echo
          method: post
    environment:
      POST_MESSAGE_URL: 'https://slack.com/api/chat.postMessage?'
      BOT_ACCESS_TOKEN: 'xoxb-159279836768-FOst5DLfEzmQgkz7cte5qiIv'
      VERIFICATION_TOKEN: 'wiM5uBXy2CcvIfTnyIdfhoVw'
      UPLOAD_BUCKET: ${self:custom.uploadBucket}
      SLACK_HOSTNAME: 'files.slack.com'
      TEMP_FOLDER: '/tmp/'
```

4. FACEOFF!

Now you are going to create a function **faceswap**. This is the main function that will perform image analysis and replace faces with emojis. In this lesson, however, we are just going to create a shell for the function and then work on its implementation later.

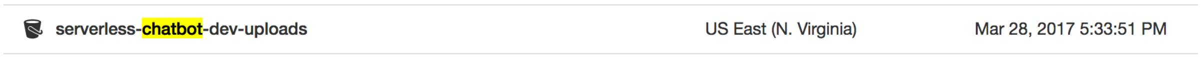
- Create a new file called **faceswap.js** and copy the following code to it

```
'use strict';

module.exports.execute = (event, context, callback) => {
  console.log(event);

  callback(null);
};
```

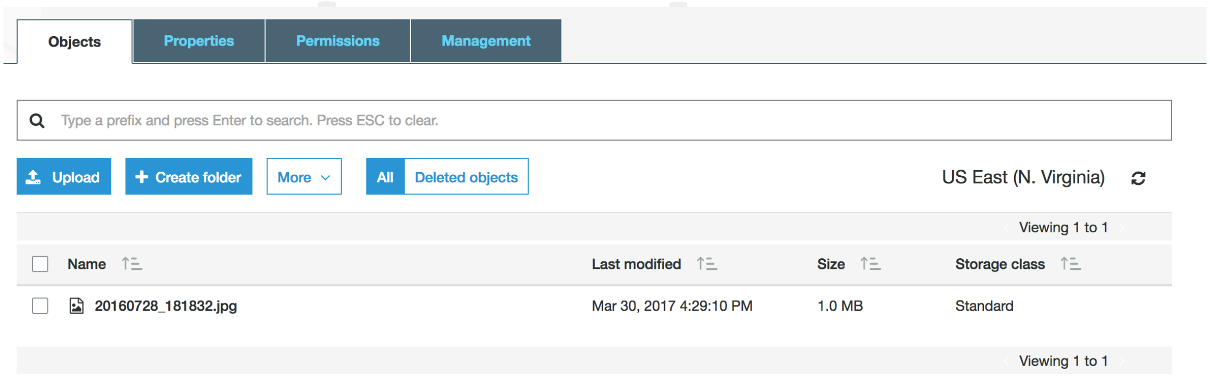
5. DEPLOY

Time to deploy what you have built.

- Deploy the two function by running **serverless deploy** from the terminal
- Check that the new **faceswap** function was created in the AWS Lambda console and have a look at the S3 console to see if you can find the new S3 bucket.


serverless-chatbot-dev-uploads US East (N. Virginia) Mar 28, 2017 5:33:51 PM

- Go to your slack and upload an image file. After a few seconds you should receive a message from *botty* that it is working on the upload.
- Go to the S3 console and click on your upload bucket. Check that the file is there. You can open it and have a look if it's the same file.



Name	Last modified	Size	Storage class
20160728_181832.jpg	Mar 30, 2017 4:29:10 PM	1.0 MB	Standard

This is it for lesson 3! It's lunch time so let's take a break and regroup after. But, if you've finished this and still have time left over, go on to the next lesson.