

## LESSON 4

In Lesson 4, you are going to continue building your image process. You've done the S3 upload function, so now you need to write the **faceswap** function. You already have the file (**faceswap.js**); now you just need to write the implementation.

### 1. PACKAGE.JSON

In this lesson, we are going to install a dependency that will be packaged up and sent along with our Lambda function to AWS. This dependency is **GraphicsMagick** for nodejs.

- Make a copy of your **lesson3** folder and name it **lesson4** (`$ cp -r lesson3 lesson4`)
- Navigate to the **lesson4** directory.
- Run **npm init** from inside the **lesson4** directory on the command line. This will create a **package.json** file.
- At the prompt, name your function **serverless-chatbot**. Accept all other default settings.

After the **npm init** process is finished, we will install **GraphicsMagick**.

- From your **lesson4** folder, run the following Terminal command: **npm install gm --save**
- After the download is finished, you can open **package.json** to verify that **GraphicsMagick** ("gm") is there as a dependency.

```
{
  "name": "serverless-chatbot",
  "version": "1.0.0",
  "description": "Serverless Chatbot",
  "main": "faceswap.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC",
  "dependencies": {
    "gm": "^1.23.0"
  }
}
```

d Guru

### 2. FACESWAP.JS

Open **faceswap.js** and replace its contents with the following implementation:

```
'use strict';

const aws = require('aws-sdk');
const fs = require('fs');

const gm = require('gm').subClass({
  imageMagick: true
});

const rekognition = new aws.Rekognition();
const s3 = new aws.S3();

function getEmojiBasedOnSentiment(emotion) {
  if (emotion) {
    return 'emoji/' + emotion[0].Type.toLowerCase() + '.png';
  } else {
    return 'emoji/happy.png';
  }
}
```

```

const detectFaces = function(bucket, filename) {
  return new Promise((resolve, reject) => {
    var params = {
      Image: {
        S3Object: {
          Bucket: bucket,
          Name: filename
        }
      },
      Attributes: ['ALL']
    };

    rekognition.detectFaces(params, function(err, data) {
      if (err) {
        reject(err);
      }
      else {
        resolve(data);
      }
    })
  });
}

const saveFileToSystem = function(bucket, key, facedata) {
  var file = fs.createWriteStream(process.env.TEMP_FOLDER + key);

  return new Promise((resolve, reject) => {
    var stream = s3.getObject({Bucket: bucket, Key: key})
      .createReadStream()
      .pipe(file);

    stream.on('error', function(error){
      reject(error);
    });

    stream.on('close', function(data){
      resolve();
    });
  });
};

const analyseImage = function(key) {
  return new Promise((resolve, reject) => {
    var image = gm(process.env.TEMP_FOLDER + key);

    image.size(function(err, size) {
      if (err) {
        reject(err);
      }
      else {

```

```

        resolve({image: image, size: size});
    }
    });
});
}

const processFaces = function(key, imgdata, facedata) {
    const image = imgdata.image;
    const size = imgdata.size;

    return new Promise((resolve, reject) => {
        for (var i = 0; i < facedata.FaceDetails.length; i++) {
            var box = facedata.FaceDetails[i].BoundingBox;

            const left = parseInt(box.Left * size.width, 10);
            const top = parseInt(box.Top * size.height, 10);

            const width = parseInt(size.width * box.Width, 10);
            const height = parseInt(size.height * box.Height, 10);

            var dimensions = `${left}` + ',' + `${top}` + ',' + `${width}` + ',' + `${height}`;
            var emoji = getEmojiBasedOnSentiment(facedata.FaceDetails[i].Emotions);
            image.draw('image Over ' + dimensions + ' ' + emoji);
        }
        resolve(image);
    });
}

const saveNewImageToSystem = function(image, key) {
    return new Promise((resolve, reject) => {
        image.write(process.env.TEMP_FOLDER + process.env.OUTPUT_PREFIX + key, function (error){
            if (error) {
                reject(error);
            } else {
                resolve();
            }
        });
    });
}

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

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

```

```

        } else {
            resolve();
        }
    });
});
});

module.exports.execute = (event, context, callback) => {
    const bucket = event.Records[0].s3.bucket.name;
    const key = decodeURIComponent(event.Records[0].s3.object.key.replace(/\+/g, " "));

    var fd = null;

    detectFaces(bucket, key)
        .then((facedata) => {fd = facedata; return saveFileToSystem(bucket, key, facedata)})
        .then(() => analyseImage(key))
        .then((imagedata) => processFaces(key, imagedata, fd))
        .then((image) => saveNewImageToSystem(image, key))
        .then(() => uploadToBucket(key))
        .then(() => callback(null, 'Success'))
        .catch((err) => callback(err))
};

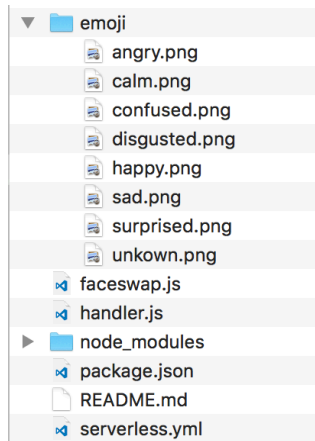
```

Wow, that is a lot of code for a simple Lambda function. So, how does it work?

- This Lambda function is invoked by an S3 event, which (in our case) is a file uploaded to the S3 bucket (in the previous lesson).
- The **faceswap** function runs and proceeds to do the following:
  - Analyse the image using the AWS Rekognition service (detectFaces)
  - Save the image file the local system (saveFileToSystem)
  - Analyse the image and get its width and height (analyseImage)
  - Process the image and replace faces with emojis (processFaces)
    - Select which emoji to use based on sentiment analysis (getEmojiBasedOnSentiment)
  - Save the new image to filesystem (saveNewImageToSystem)
  - Upload the new image to a different S3 bucket (uploadToBucket)
- All of this is done in less than 150 lines code. It can be cut down further but we leave it as an exercise to you.

### 3. EMOJI

We cannot do this workshop without an array of fun emojis! Check out the github source code for this lesson and grab the **emoji** folder. It should have 8 emojis. Copy this folder to your own working folder (**lesson4**) where **serverless.yml** resides.



#### 4. AWS REKOGNITION

In the previous step, you used AWS Rekognition to perform image analysis. This service scans any image and attempts to detect faces, landmarks, labels and signs. It can also detect features like eyes and mouth, and try to guess how a person in the photo is feeling. We need to grant our **faceswap** Lambda function access to AWS Rekognition.

- Open **serverless.yml** and add the following new **iamRoleStatement**:

```
iamRoleStatements:
  - Effect: Allow
    Action:
      - rekognition:DetectFaces
    Resource: '*'
```

- The old S3 role statement should still be there. **Do not remove it.**

#### 5. ENVIRONMENT VARIABLES

If you look closely at the code in step 1, you will see that it refers to environment variables such as TEMP\_FOLDER, OUTPUT\_PREFIX, UPLOAD\_BUCKET and TRANSFORM\_BUCKET. The TRANSFORM\_BUCKET variable is new and important. It refers to the second bucket serverless framework will create for us; it will contain modified images. Update the **faceswap** function definition in your **serverless.yml** as follows.

```
faceswap:
  handler: faceswap.execute
  environment:
    TEMP_FOLDER: '/tmp/'
    OUTPUT_PREFIX: 'output-'
    UPLOAD_BUCKET: ${self:custom.uploadBucket}
    TRANSFORM_BUCKET: ${self:custom.transformBucket}
  events:
    - s3:
        bucket: ${self:custom.uploadBucket}
        event: s3:ObjectCreated:*
```

#### 6. TRANSFORM BUCKET

The other thing we need to add is the second bucket, which will store the transformed images.

- Update **serverless.yml** to make the **custom:** section look like this:

```
custom:
  uploadBucket: ${self:service}-${self:provider.stage}-uploads-YOURINITIALS
  transformBucket: ${self:service}-${self:provider.stage}-transformed-YOURINITIALS
```

We must not forget about security, so update your **iamRoleStatements** to look like this:

```
iamRoleStatements:
  - Effect: Allow
    Action:
      - rekognition:DetectFaces
    Resource: '*'
  - Effect: "Allow"
    Action:
      - s3:GetObject
      - s3:PutObject
    Resource: arn:aws:s3:::${self:custom.uploadBucket}/*
  - Effect: "Allow"
    Action:
      - s3:GetObject
      - s3:PutObject
    Resource: arn:aws:s3:::${self:custom.transformBucket}/*
```

## 7. UPDATE SLACK FUNCTION

Last -- but not least -- lets create a **slackupdate** function. We are not going to implement it now, but it will be good to have it ready for the next lesson.

- Open **serverless.yml** and add the following to the end of the **functions:** section.

```
slackupdate:
  handler: slackupdate.execute
  events:
    - s3:
        bucket: ${self:custom.transformBucket}
        event: s3:ObjectCreated:*
```

- In the same folder as **serverless.yml**, create a new file called **slackupdate.js** and copy the following code to it.

```
'use strict';

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

  callback(null);
};
```

## 8. DEPLOY AWAY!

Let's deploy your functions to AWS. From the terminal (in your **lesson4** folder), run **serverless deploy**.

```
Peters-MacBook-Pro-2:Lesson 4 petersbarski$ serverless deploy
Serverless: Packaging service...
Serverless: Uploading CloudFormation file to S3...
Serverless: Uploading service .zip file to S3 (151.63 KB)...
Serverless: Updating Stack...
Serverless: Checking Stack update progress...
.....
Serverless: Stack update finished...
Serverless: Removing old service versions...
Service Information
service: serverless-chatbot
stage: dev
region: us-east-1
api keys:
None
endpoints:
POST - https://729bcwptkj.execute-api.us-east-1.amazonaws.com/dev/echo
functions:
hello: serverless-chatbot-dev-hello
faceswap: serverless-chatbot-dev-faceswap
slackupdate: serverless-chatbot-dev-slackupdate
```

## 9. TEST

Let's test our progress so far. Go to your Slack channel and upload a picture with faces in it. If you look at the source code folder you got from GitHub, you'll find a directory with sample pictures you can use for testing.



After uploading the image, you'll see the "I am working on <filename>... should be done soon." message from *botty*. Next, move to the AWS S3 Console, and open your S3 transform bucket: you should see your image. Open the image to see it with emojis.



Wow! You're smashing it! And now, it's time to take a little break before we move on to the next lesson.