Writing a cron job microservice with Serverless and AWS Lambda

30 JANUARY 2017 on Engineering

We recently had a situation where we needed to create a new cron job to fetch all users from our database who are coming to the end of their trial and insert them into our customer.io database. Cron jobs are easy to write, but difficult to setup. You can edit `/etc/crontab` on the server; if you're using heroku you can use their Scheduler; or you can use some implementation of cron in your programming language of choice (e.g. Node.js).

The cron job that we needed to write was unrelated to our application code, so whilst we could have put the functionality in there it seemed like the wrong place. Alternatively we could have put the code onto a new server. This would mean provisioning a new box for something that is only going to run once a day for 10 seconds. This seems very wasteful and expensive.

Enter AWS Lambda/Serverless

AWS Lambda is a cloud computing platform which allows you to upload a single block of code and run it in a cloud environment. The beauty of it is that you only pay for the time when your code is running (in increments of 100ms). This combined with AWS ScheduledEvents sounds like the perfect solution!

Serverless is a framework and command line tool which simplifies the creation of AWS Lambda functions through a nicer API than the official aws-sdk node module. It has good docs for setup as well as how to give it access to your AWS account: https://serverless.com/framework/docs/providers/aws/guide/installation/.

A microservice can be created with the following commands (you will need to setup AWS too https://serverless.com/framework/docs/providers/aws/guide/credentials/):

```
npm i serverless -g
serverless create --template aws-nodejs --path users-ending-trial
cd users-ending-trial
```

Then update the `serverless.yml` file and add the scheduled events option with the time you want to run it:

```
service: users-ending-trial
provider:
  name: aws
  runtime: nodejs4.3
functions:
  fetchTrialUsers:
    handler: handler.fetchTrialUsers
    events:
      # 10am every morning
      - schedule: cron(0 10 * * ? *)
```

Update the `handler.js` with your code to run:

```
'use strict';

const mongo = require('mongodb').MongoClient;

const CUSTOMER_IO_KEY = process.env.CUSTOMER_IO_KEY;
const CUSTOMER_IO_SECRET = process.env.CUSTOMER_IO_SECRET;
const MONGO_READ_URL = process.env.MONGO_READ_URL;

const CustomerIo = require('customerio-node');

const cio = new CustomerIo(CUSTOMER_IO_KEY, CUSTOMER_IO_SECRET);

function getRelativeDate(offset) {
  return new Date(new Date().setDate(new Date().getDate() + offset));
}

module.exports.fetchTrialUsers = (event, context, callback) => {
  mongo.connect(MONGO_READ_URL, (err, db) => {
    db.collection('projects').find({ 'trial.trialEndsAt': { $gt: getRelativeDate(3), $lt: g
etRelativeDate(4) }, plan: 'free' }).toArray((err, projects) => {
      if (err) {
        console.error('Error reading projects from database');
        return callback(err);
      }

      Promise.all(projects.map((project) => {
        return cio.track(project.owner, { name: 'threeDaysLeft', data: { project } });
      })).then(() => {
        callback(null, projects.length);
      }, (e) => {
        console.error('Error tracking customers in customer.io', e);
        return callback(e);
      });

      db.close();
    });
  });
};
```

To test this locally, Serverless gives you the `invoke local` command which cleverly mocks out Lambda internals:

```
serverless invoke local -f fetchTrialUsers
```

To deploy it to production:

```
serverless deploy -f fetchTrialUsers
```

To run it, just take the command from before without the `local`:

```
serverless invoke -f fetchTrialUsers
```

Et voila! This should now run your lambda function every day at your designated time. This is a really powerful way of working which allows you to focus on the code and not on the infrastructure which is required to run your code. I can see us using this for many more things in future.

Marc Cuva

Read more posts by this author.