**User’s Guide**

CIS 4911 - Senior Project (U01)

Event Driven Cloud Computing

Cory McAn

April 24th, 2015

Jason Dettbarn

# Copyright and Trademark Notices

All Addigy specific logos and slogan material is copyright Addigy, Inc. All other materials developed for this project are available open source under MIT License:

<http://opensource.org/licenses/MIT>

# Abstract

This document will give an overall user guide to all the features of Event Driven Cloud Computing. I have included some images to assist the user and clarify the instructions. I have also included links to the necessary code and systems needed to run and test this project.

**Introduction**

Event Driven Cloud Computing is a backend audit processing software that is intended for IT Monitoring and Alerting. The Amazon Web Services Lambda and S3 are used to provide a serverless run environment (Lambda) and an easy way to maintain data (S3). The software analyzes audit data that is put into the bucket and sends alerts in real-time to an IT Admin that has access to a PubNub Console where the alerts flow in.

**Hardware and Software Requirements**

Hardware:

This software does not require any particular hardware as the main computation is done by Lambda in the cloud.

Software:

Cyberduck – for uploading batches of test files to test alerting

IntelliJ IDEA – IDE with node.js compatibility and excellent plugins for use with AWS

AWS Manager – plugin for IDEA that gives secure access to AWS S3

Apache server – for running demo locally

Node.js Interpreter – used to test code IDE side

Web Browser – Chrome, Firefox, Internet Explorer

**Installation and Setup**

Step 1:

First we need to get the source files for the project into a directory where we can access them.

Open a shell program, navigate to the desired folder to store the project files, and execute the command to copy the project files into the folder:

Git clone <https://github.com/FIU-SCIS-Senior-Project-2015-Spring/Event-Driven-Cloud-Computing.git>

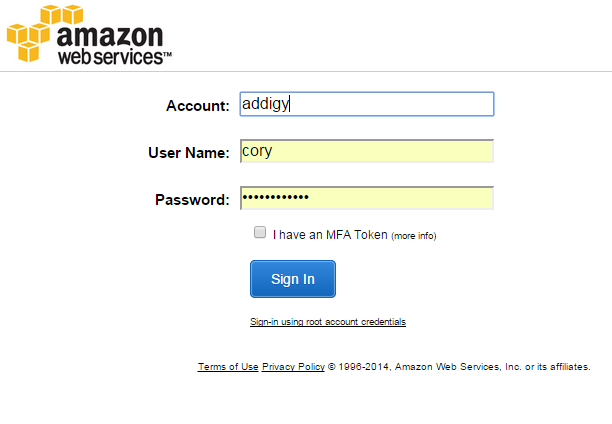
Now that we have the source files, we can view, edit, and upload them to Amazon Web Services where we will host our Lambda functions to be run on event triggers.

Step 2:

Navigate to <https://addigy.signin.aws.amazon.com/console>.

Ensure that the account is “addigy”.

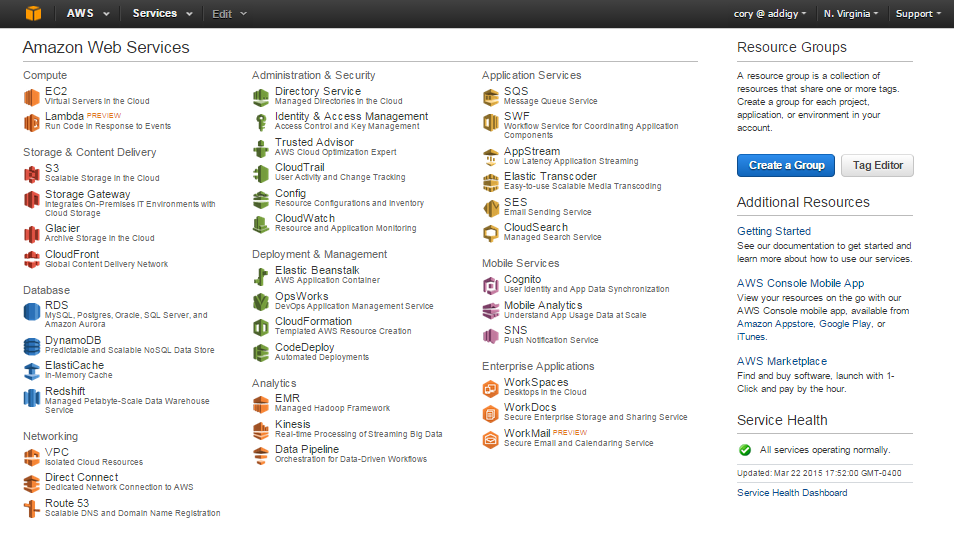
Log into Amazon Web Services with the credentials given to you by your AWS admin.



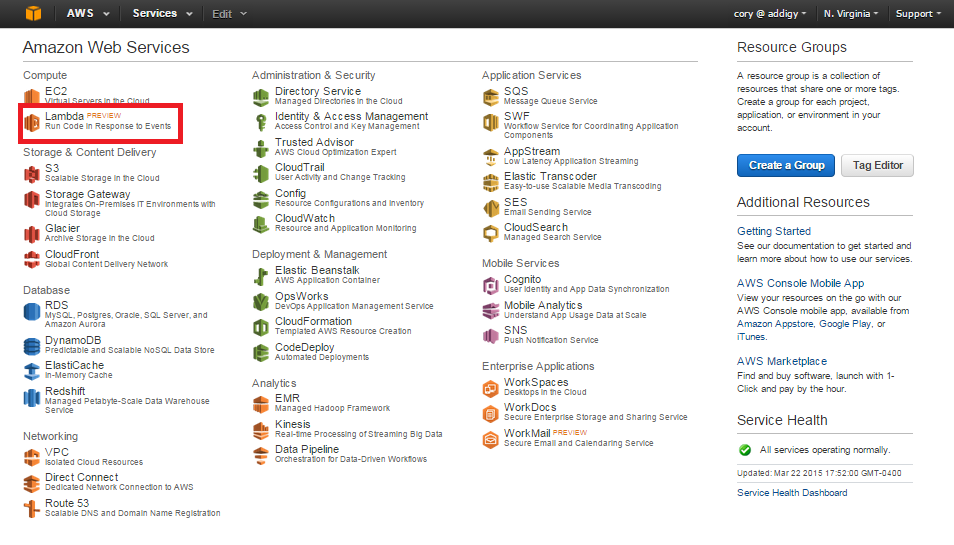
Step 3:

You will now be at the Amazon Web Services home page.

The only necessary services here are Lambda and S3.



Let’s start by clicking on Lambda to understand how to create and invoke a Lambda Function.

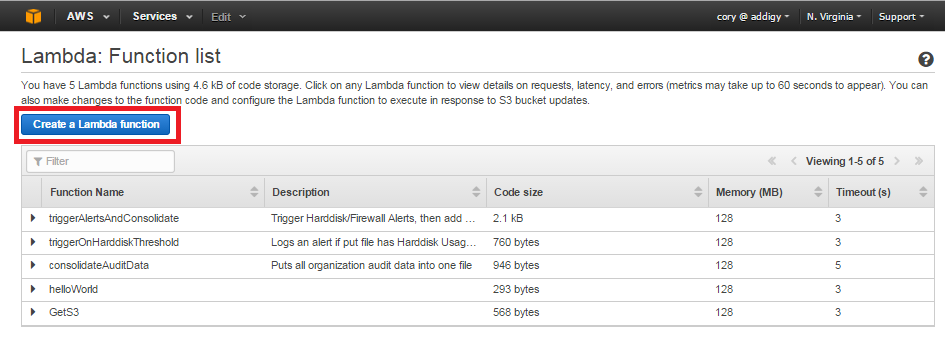


Step 4:

By clicking Lambda, we arrive at the Lambda Function List.

Here you can manage your Lambda functions. There are many features such as creating a function, editing a function, configuring the event source, testing the function in the Lambda console, and checking the logs related to any Lambda function.

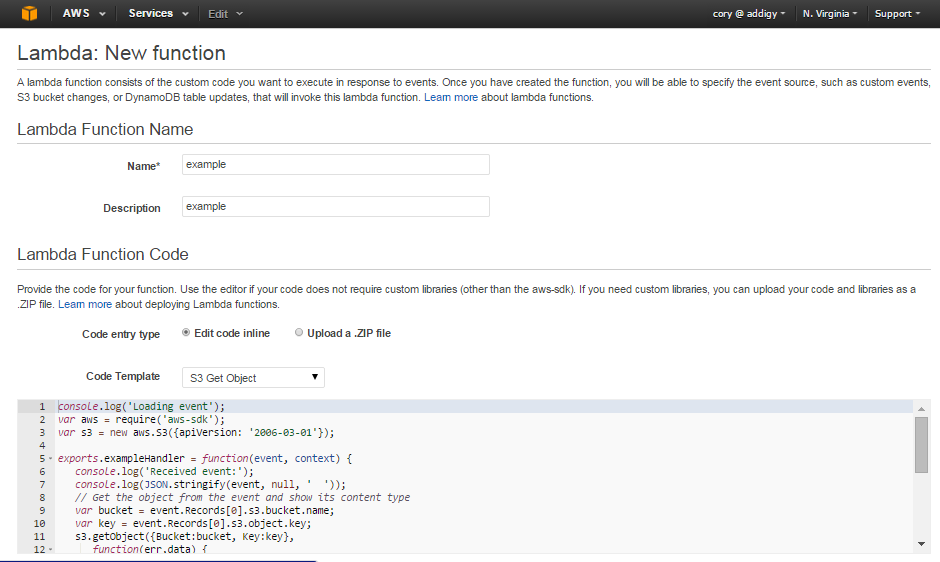
Let’s try creating a Lambda function by clicking “Create a Lambda function”.



Step 5:

This is the page for creating a Lambda function. AWS provides you with function templates for testing and learning purposes, which would be helpful to look through.

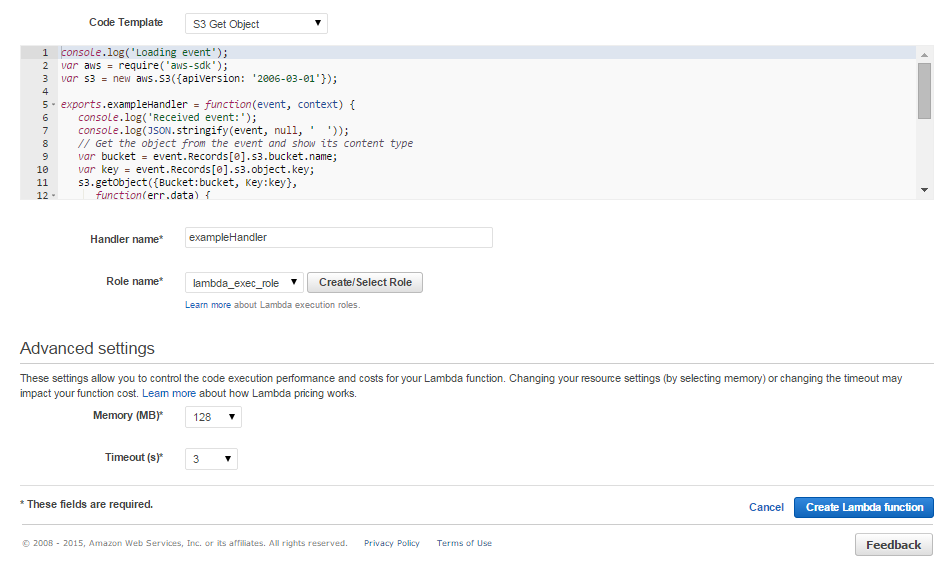
On this page we need to specify the name of our Lambda function and optionally describe the function. After that we can either upload a .zip file with our Node.js code (This is necessary if the code requires any packages not present in the core libraries of Node.js) or we can paste the Node.js code into the provided box if there are no additional libraries needed.



Next we need to name the handler, which is the same name that follows “exports.” In this case the handler is exampleHandler.

After we have a handler, we need to specify the execution role of the lambda function. Your Amazon Web Services admin must give you access to a lambda execution role as normal users cannot create them. Once your admin creates the role and gives you access you can select it from the drop down menu.

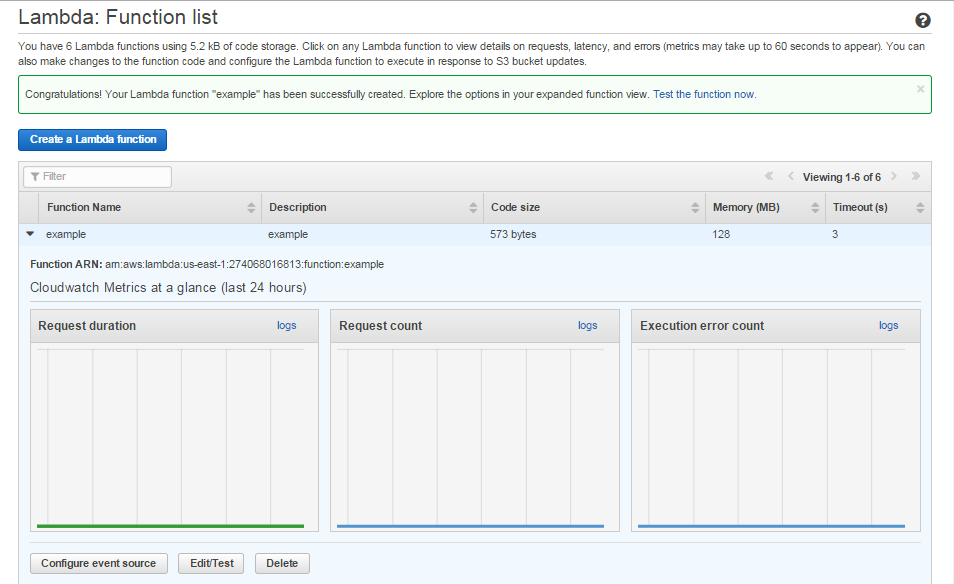
There are also advanced settings related to how long the Lambda function will run until it times out and the maximum amount of memory a Lambda function can use. As stated on the page, changing these values will have an impact on the cost of running the Lambda functions, so be sure about any changes you make to these values.



Now we simply need to click “Create Lambda function” and it will show up in our list!

Click the down arrow next to the function name and you will see the options below:

(next page)



Step 6:

Now we have created a Lambda function. Each of your functions can be assigned to an S3 Bucket for the purpose of being triggered by a certain event on the Bucket.

Step 7:

Once you have set up Amazon Web Services, uploaded the Lambda functions and configured the functions to a bucket or buckets, you are ready to begin testing the functionalities.

The Lambda functions are written in Node.js, so it is necessary to find an IDE which can compile the code for IDE-side testing before doing a real run in AWS.

The IDE I used for this project is IntelliJ IDEA as it has strong debugging tools for node.js and plugins that integrate with AWS.

Navigate to this link and download the version that is relevant to your system: <https://www.jetbrains.com/idea/download/>

Step 8:

Next we need to install the Node.js interpreter, which you can get at the node website.

Navigate to: <https://nodejs.org/#download> and download the node framework onto your machine.

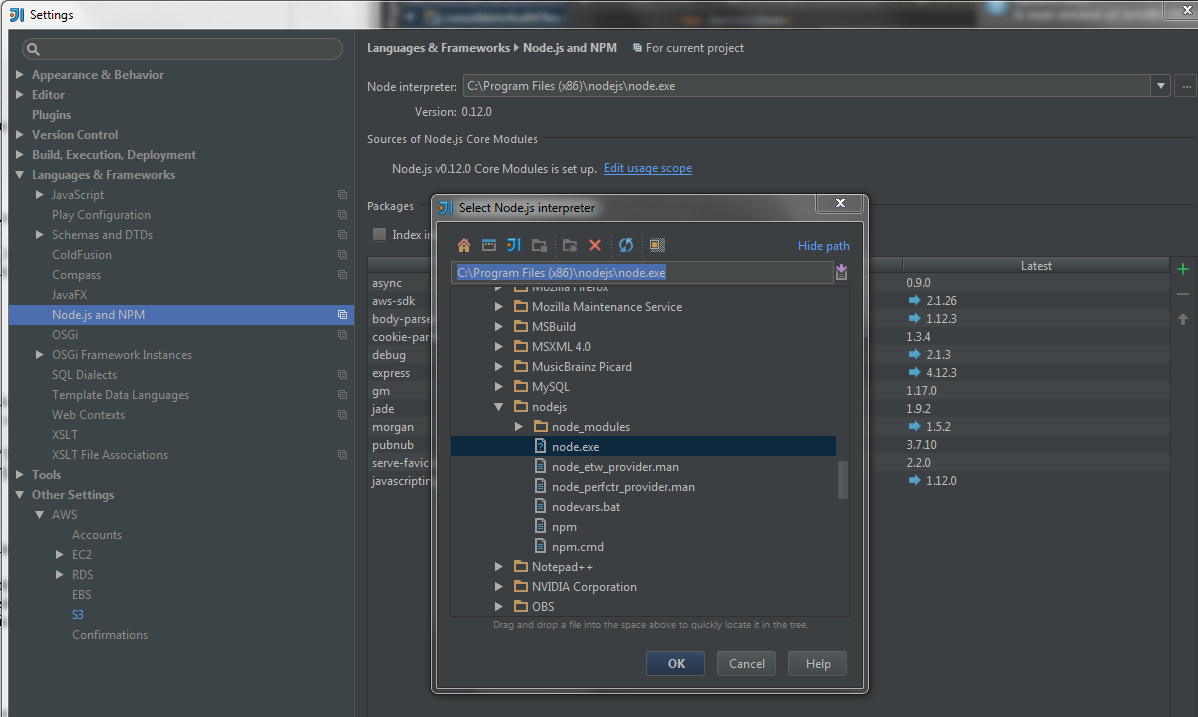
Step 9:

Now we will install the IDE and the Node Interpreter so that we can do any small tests using the IDE without needing to push new code to AWS.

With the IDE installed and open, File > Settings.

Under settings, click the arrow to the left of “Languages & Frameworks” > Node.js and NPM

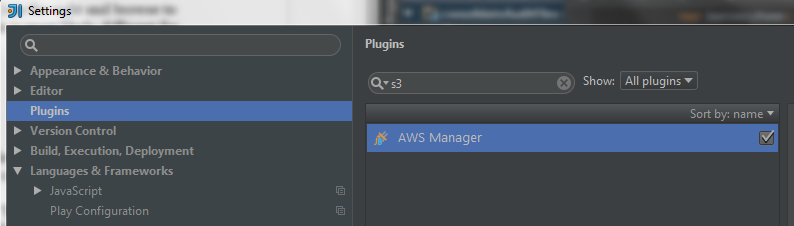
There is a box at the top for the Node Interpreter. Click the box (…) on the right and browse to node.exe (For Windows machines) and use it as the interpreter. (File is most likely different for other OS’s)



Step 10:

Now we can install the AWS S3 plugin for IntelliJ IDEA, which allows us to pass the AWS credentials through the plugin giving us a secure way of testing the code on the IDE-side.

In the same “Settings” window as in the previous step, navigate to “Plugins”. There is a search box at the top, simply search “s3” and the first result should be AWS Manager.

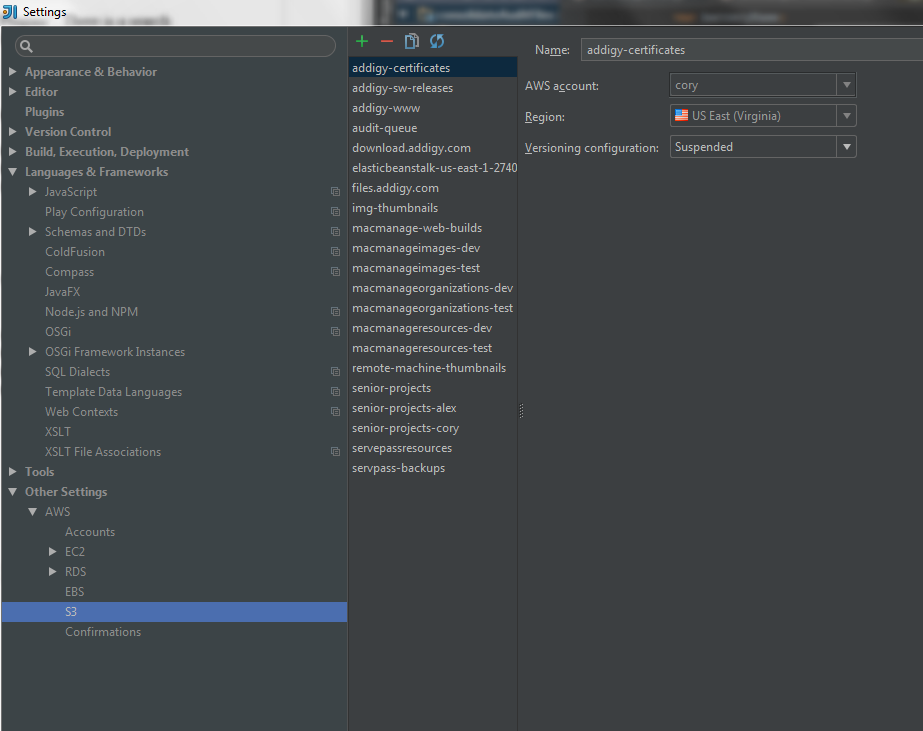


Step 11:

Click “Install JetBrains Plugin…”. Once installed, navigate to Settings > “Other Settings”.

Now AWS is available, and we can configure our login to AWS in the IDE.

Click Accounts and type in your credentials. Once you have the credentials click Test Connection. If your credentials are correct you should be able to click “S3” and see all the buckets available to your AWS account.



Step 12:

Now that we’ve fully set up our IDE with access to AWS, any of the code in the Code folder can be tested without interacting directly with the S3 web portal.

IMPORTANT: The code naming convention used for this project is simple: Either the code is the IDE version or the Lambda version for uploading to S3 and use in buckets. If the code has the suffix “Lambda” then it is configured for being used in a bucket, otherwise it is for testing in the IDE. Statements such as “context.” must be used in the bucket context, but are not recognized by the node framework in the IDE.

Step 13:

The user manual also has details for getting connected to PubNub for the purpose to testing the “multipleRunTester” which uploads many files to the bucket and gets alerts in PubNub console.

Here I will describe a method for accomplishing the multipleUpload to S3.

For this project I used CyberDuck, which has support for AWS and S3.

Navigate to <https://cyberduck.io/> and download CyberDuck for your OS.

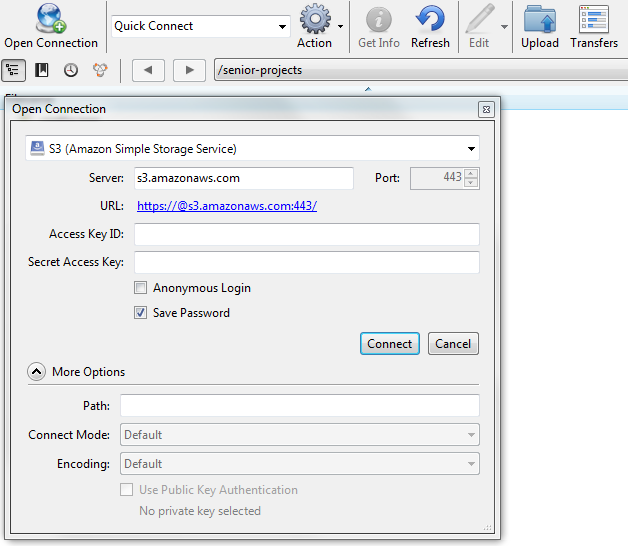
Once installed, all we need to do is configure our AWS login to be able to use CyberDuck for multipart uploads.

Click File > Open Connection…

In the drop down menu there are many different upload options. We want S3 (Amazon Simple Storage Service).

The server will be automatically changed for you. Now just put in your AWS credentials and click “Connect”.

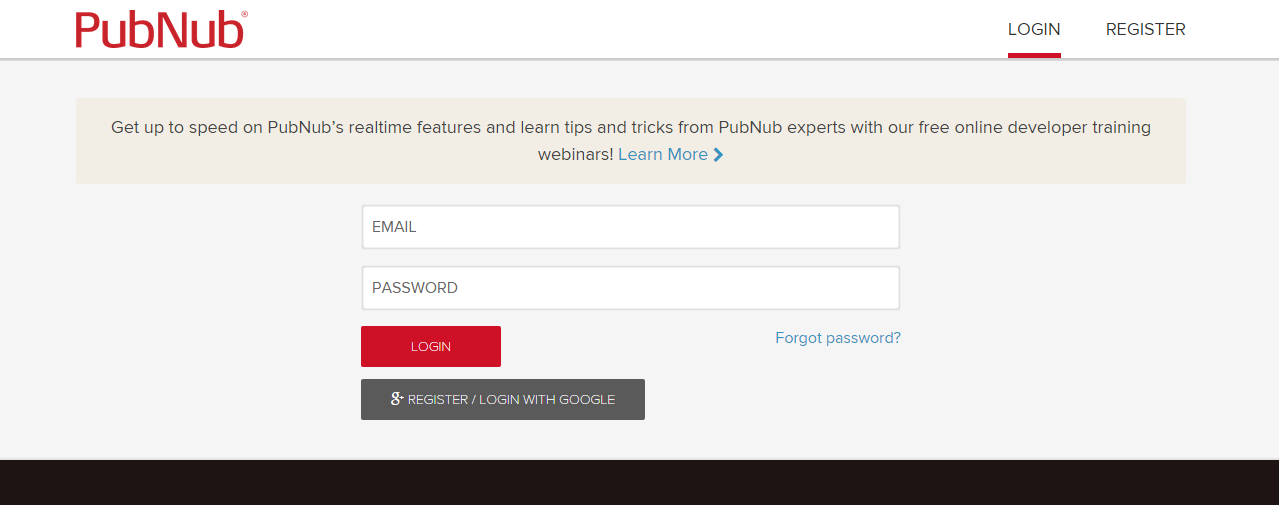
Once you are connected, you will have a view with all the buckets linked to your AWS account.

Navigate to a bucket and now we can attempt a multi part upload!

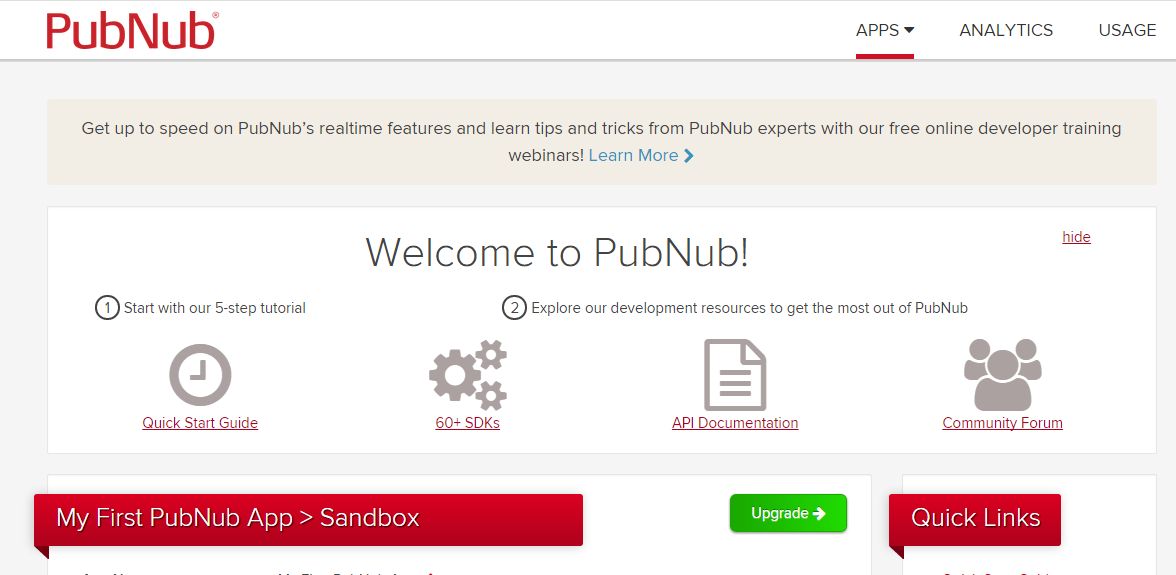
Step 14:

Now that we have AWS and S3 configured, let’s get PubNub set up so that we can test our functions!

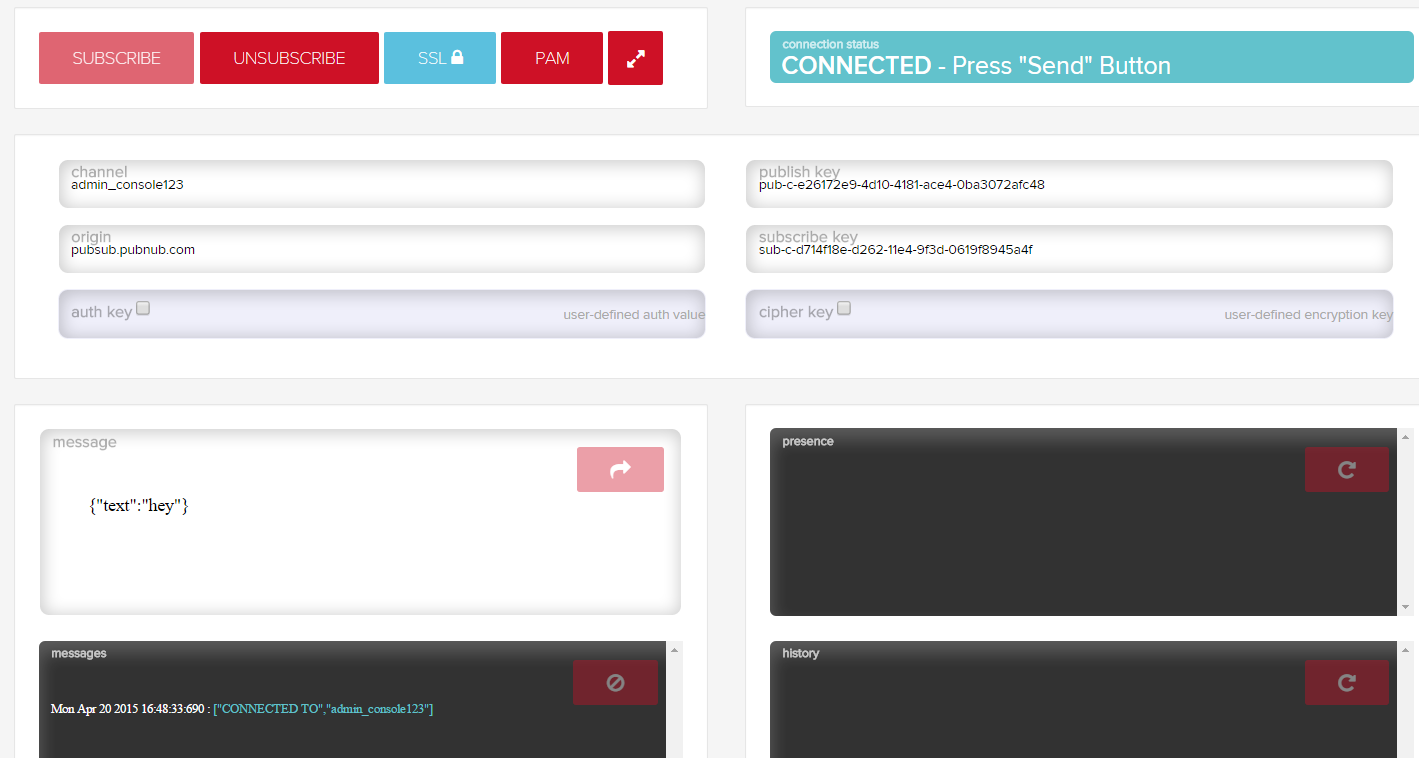
Navigate to <https://admin.pubnub.com/#/login>.



Input your login for PubNub and when you log in you’ll be at this screen:



Directly below “My First PubNub App” you will have your keys for publishing and subscribing to the PubNub channel. To the right under “Quick Links”, click the link titled “Debug Console”.



The debug console is all you need to test the lambda code in S3! Once you check the keys in the config file and make sure they match the publish and subscribe keys in the debug console, you can start testing a bucket by placing a file into the bucket and seeing the alert output in the console!

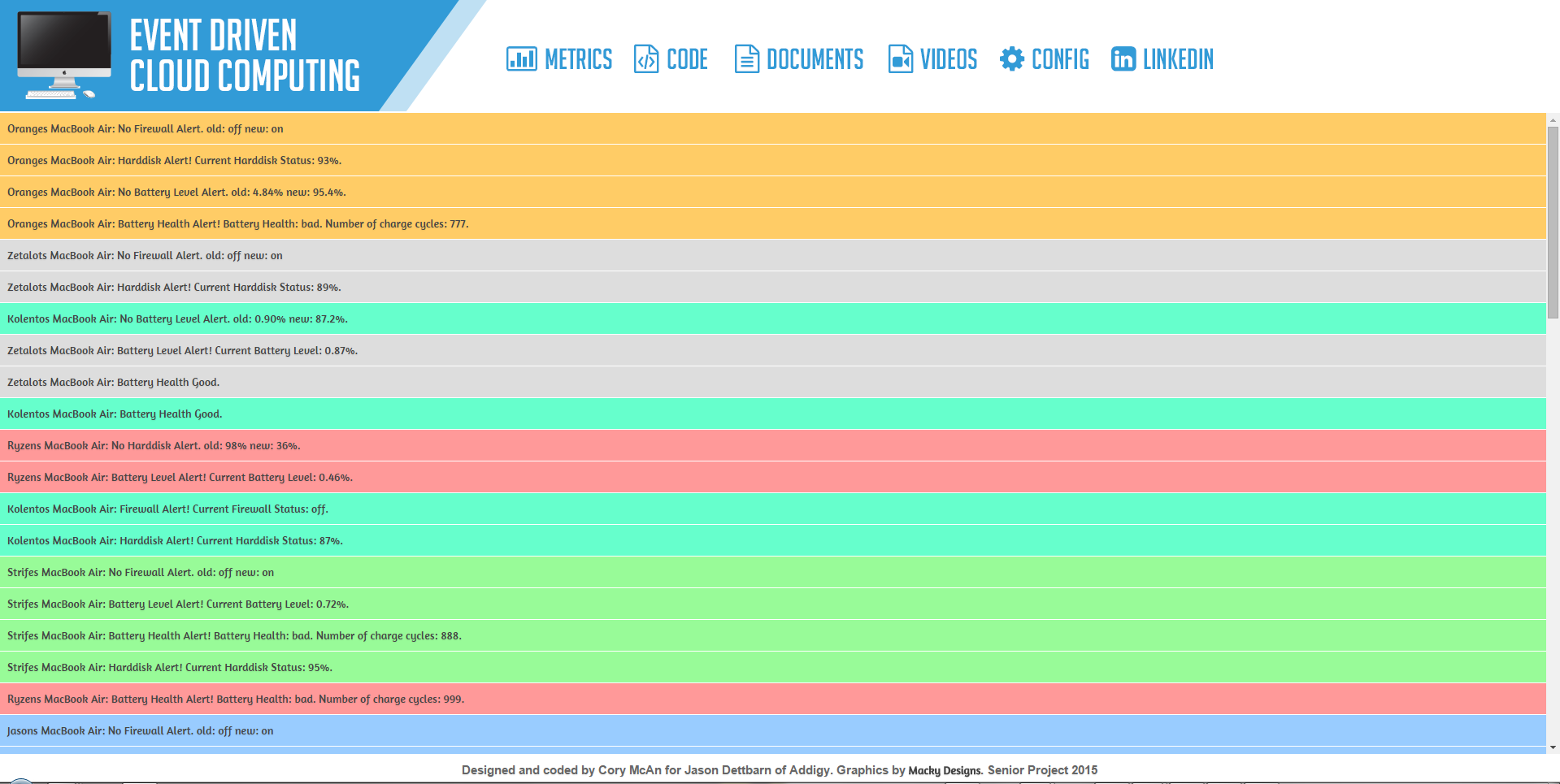
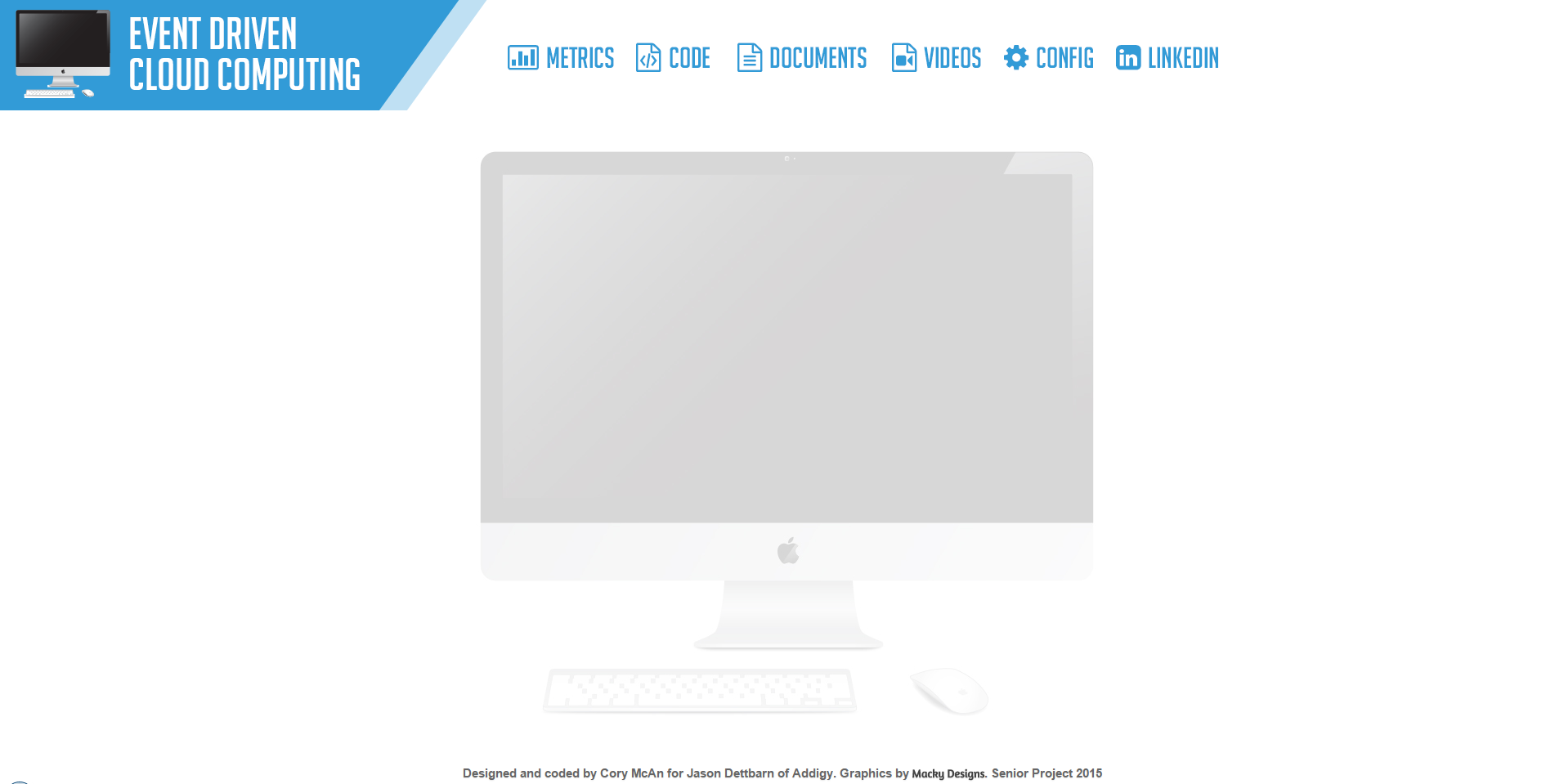
**Getting Started**

The next step up from the PubNub developer console is the Dedicated Console that I created for the demo. Inside of the project folder, navigate to Code/demo, and download the contents onto your machine.

To test it, run an HTML server on your machine (I used Apache) and navigate to 127.0.0.1 or <http://localhost>. I recommend testing it using the MultipleRunTester, Refer to the installation manual for the set up of the batch uploading software.

Once you upload the batch of audit files to the bucket of your choice and the PubNub publish keys in the configuration file in the bucket matches the channel of the subscribe keys of the demo.html page (you can modify the keys, located in the javascript near the bottom of the file), you will be able to see all the alerts related to the test files in the Dedicated Console window.

Below: Dedicated Console before and after uploading batch of audit files.



**Quick Reference**

Git Repo: <https://github.com/FIU-SCIS-Senior-Project-2015-Spring/Event-Driven-Cloud-Computing.git>

AWS signin: <https://addigy.signin.aws.amazon.com/console>

PubNub signin: <https://admin.pubnub.com/#/login>

IntelliJ IDEA: <https://www.jetbrains.com/idea/download/>

Node.js Interpreter: <https://nodejs.org/#download>

CyberDuck: <https://cyberduck.io/>

**References**

Tutorial pages for each service. Did not require other references.