# **Kura Labs:**

# Deployment 1

Overview of Deployment:

* Initial commands (Jenkins Installation)

1. sudo apt update:

Used to retrieve and update the information of packages stored within

/etc/apt/sources.

1. sudo apt install default-jre

This command uses the package manager apt to install the Java Runtime Environment which is a dependency for Jenkins. It is able to compile and launch Java based programs

1. wget -q -O – <http://pkg.jenkins.io/debian-stable/jenkins.io.key> | sudo gpg --dearmor -o /usr/share/keyrings/jenkins.gpg

This one liner is used to retrieve the necessary authentication key to be able to download the Jenkins package repository onto your machine.

1. sudo sh -c ‘echo deb [signed-by=/usr/share/keyrings/Jenkins.gpg] [http://pkg.jenkins.io/debian-stable binary/](http://pkg.jenkins.io/debian-stable%20binary/) > /etc/apt/sources.list.d/jenkins.list’

Before we can install Jenkins, this next command is responsible for adding the Jenkins package repository to the list of available sources within the package manager, apt. Once added we first retrieve the updated source list then finally we can install Jenkins.

Text

Description automatically generated

* Initiating Jenkins and running jenkins

Jenkins is to be initiated with the following command:

systemctl start jenkins

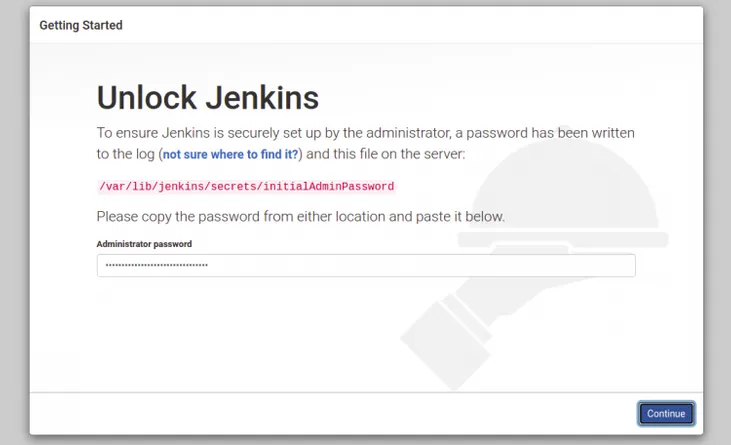
Whereas; systemctl status is meant to confirm that jenkins is running on your EC2

Text

Description automatically generated

Once the command line displays this, we’ve confirmed that Jenkins is running without incident. By default, it appears once jenkins is open it will be hosted on port 8080.

To access Jenkins, I used the following url format: <EC2 public ip address>:8080

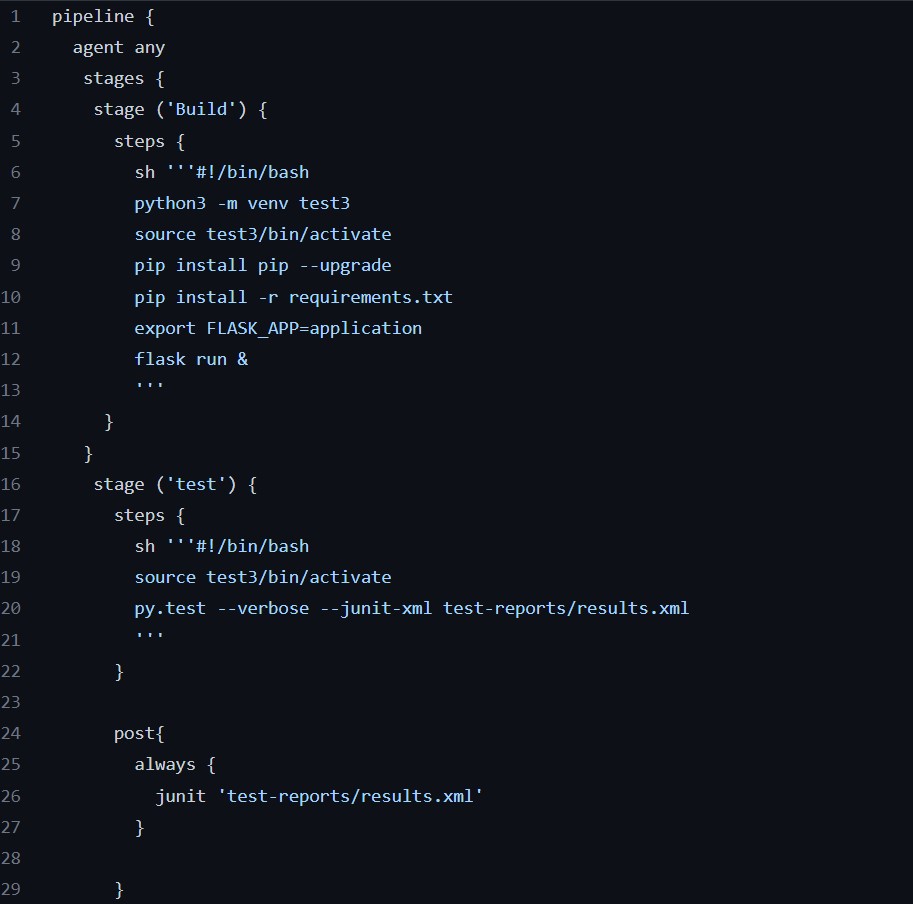
The very first time I installed Jenkins and went to that url, I was greeted with this screen

A simple matter of following the instructions and using the command:

sudo cat /var/lib/jenkins/secrets/initialAdminPassword. Once entered, the next page instructs us to create an admin account, however for this deployment I skipped that option and went straight to the Jenkins dashboard.

* Application dependencies

Within our application repository there is the Jenkinsfile, this is responsible for configuring our pipeline. Within the pipeline there are various stages in which certain commands are ran.



Most notably within build we create a python virtual environment inorder to test and run our application. However, we have yet to install the necessary packages for this virtual environment and the package manager pip. Thus we run the following commands:

1. sudo apt install python3-pip
2. sudo apt install python3.10-venv

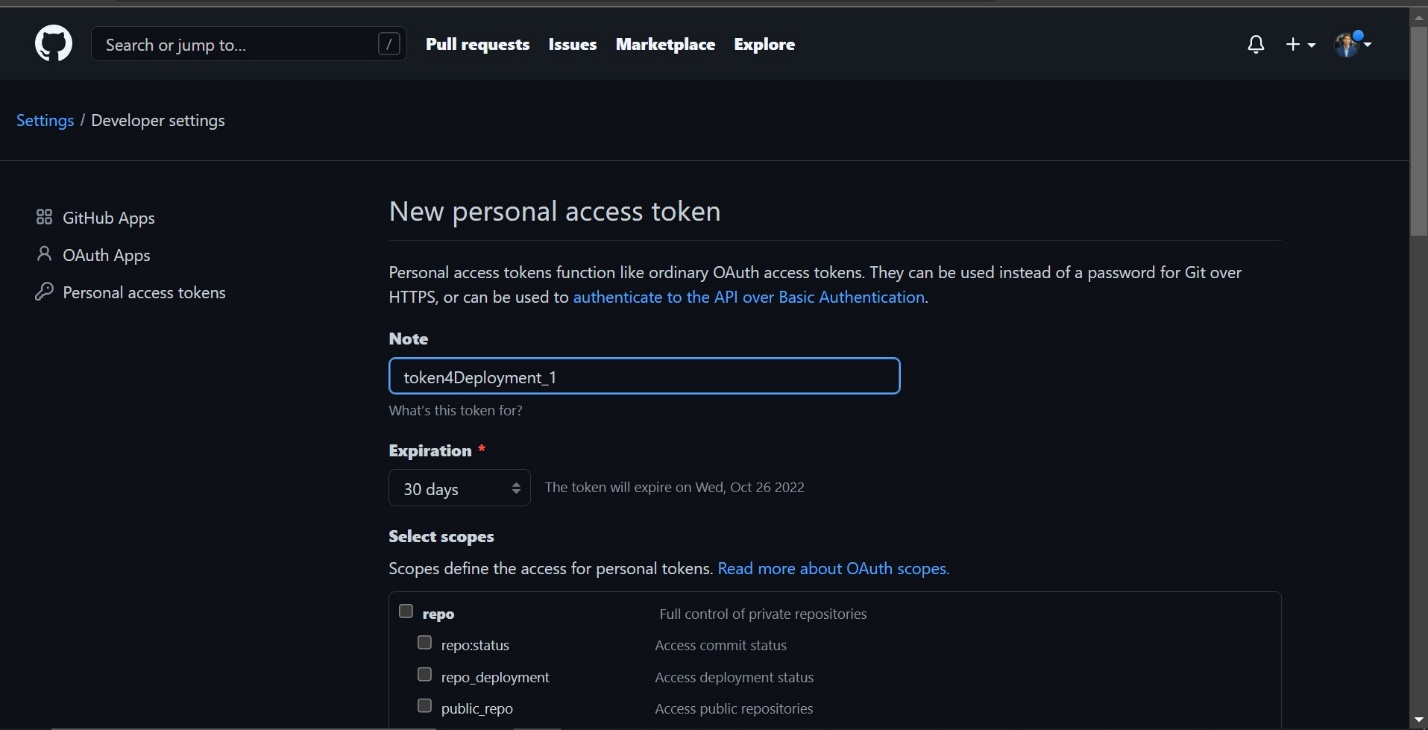
With both those two installed once we connect our repo to jenkins there should be no issues within our build stage and beyond.

* Linking Jenkins to Github

In this deployment Jenkins will be handling our build and testing phase, thus inorder to do so we must first link our repository to our Jenkins instance.

2 credentials are necessary for this to occur:

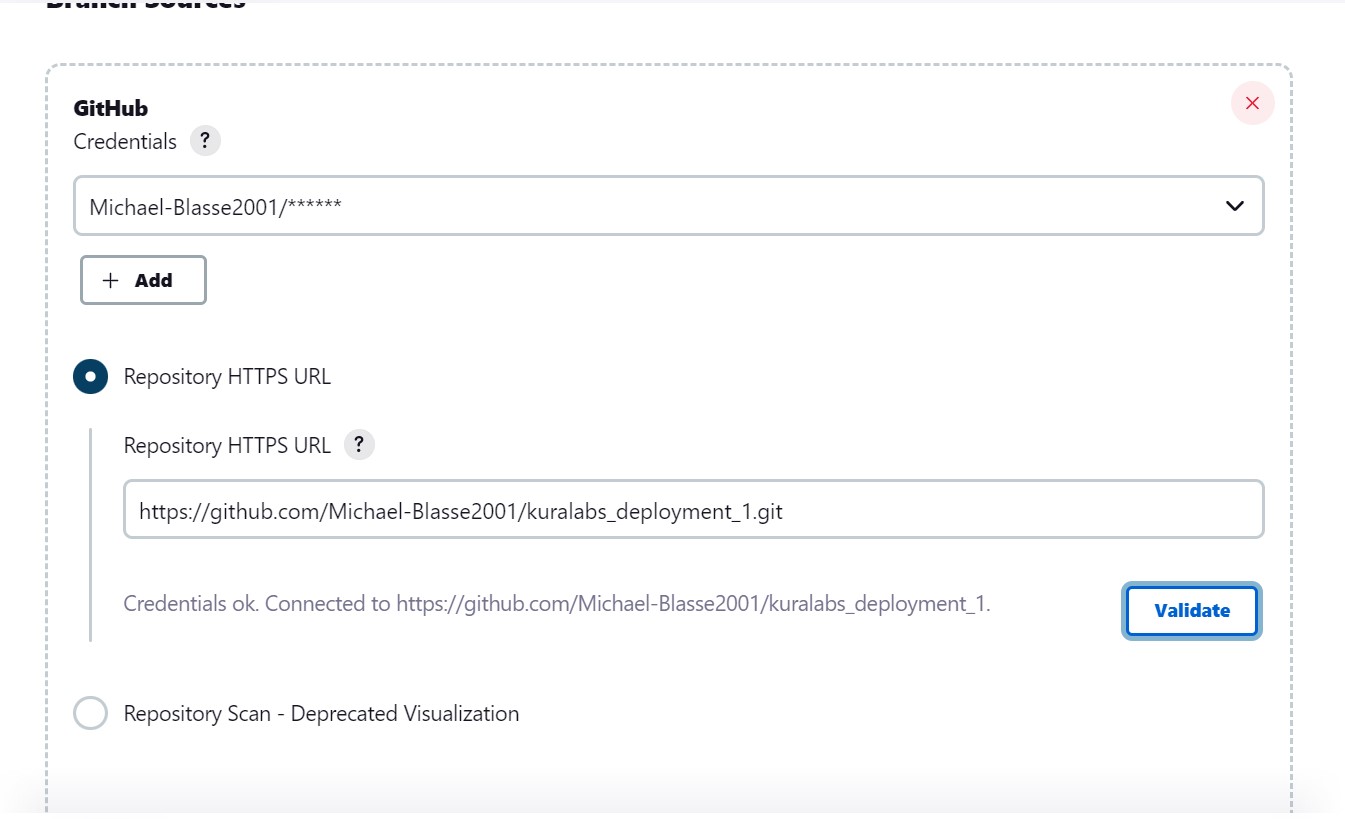
A personal access token- The personal access token is a created by going into Github developer settings and selecting personal access token from the list. For this token only 2 permissions were needed, repo and admin:repo\_hook.



Github account name

With these two credentials we click new item on our Jenkins dashboard and create a multi-branch pipeline. In our case, named “Build Flask”. Once a title and description were given within branch sources, we select Github then the add button and finally enter our credentials. The token goes in the password field and our username in the username field.

Once this is done, I simply copied the repository url within the browser and pasted it within the Repository HTTPS URL field. After validating, I simply clicked save.



As Github and Jenkins are linked the pipeline began automaticallyGraphical user interface, text, application

Description automatically generated

Within a few seconds the build successfully completed. The application passing all stages.

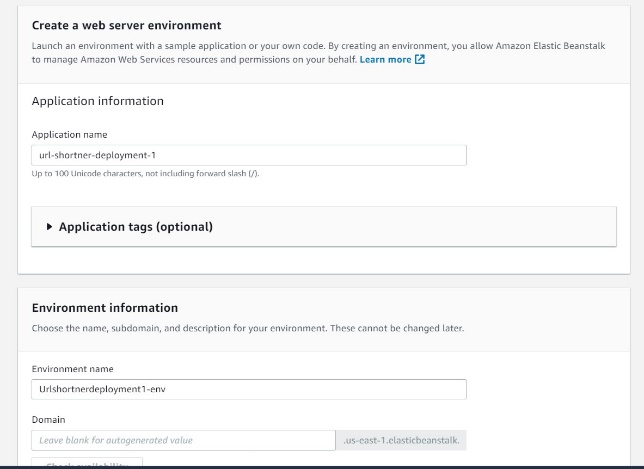
Graphical user interface, application

Description automatically generated

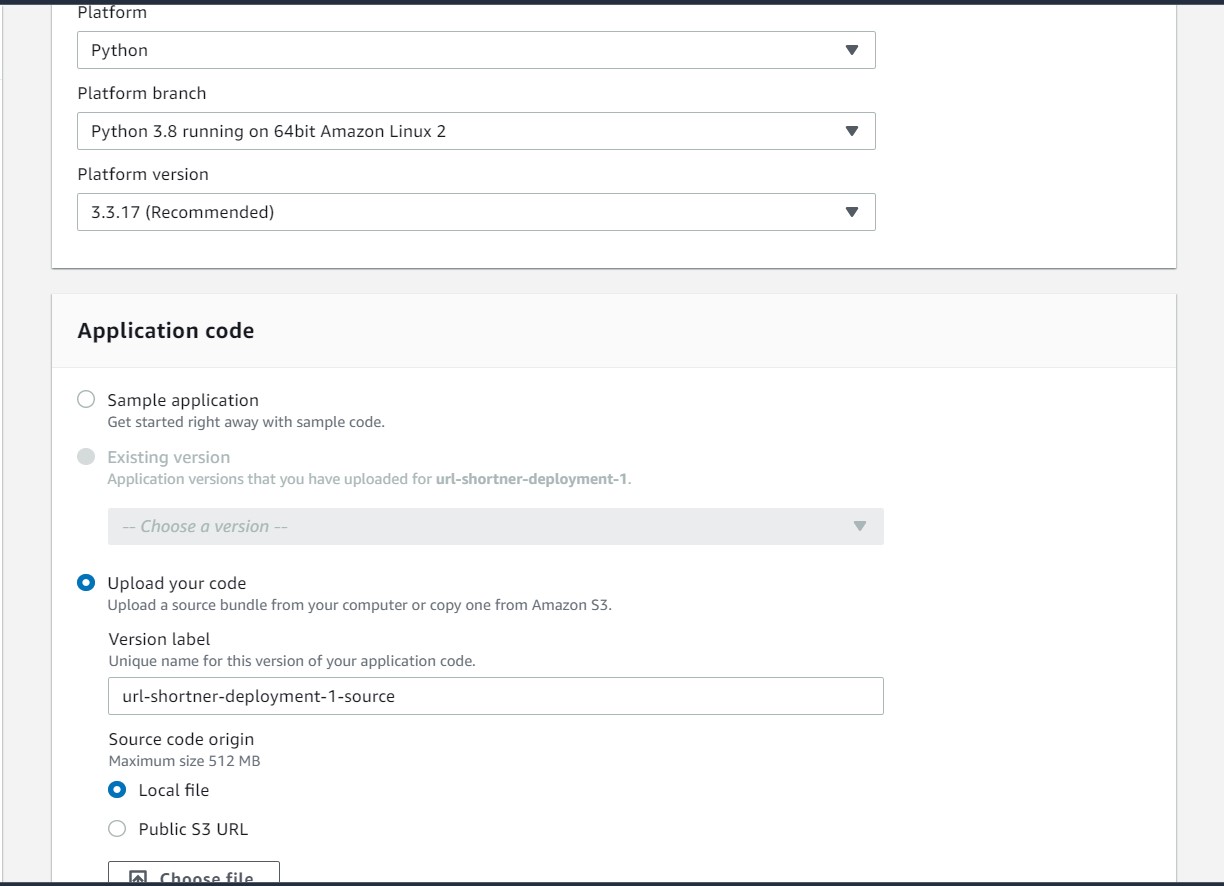
* Deploying to Elastic Beanstalk

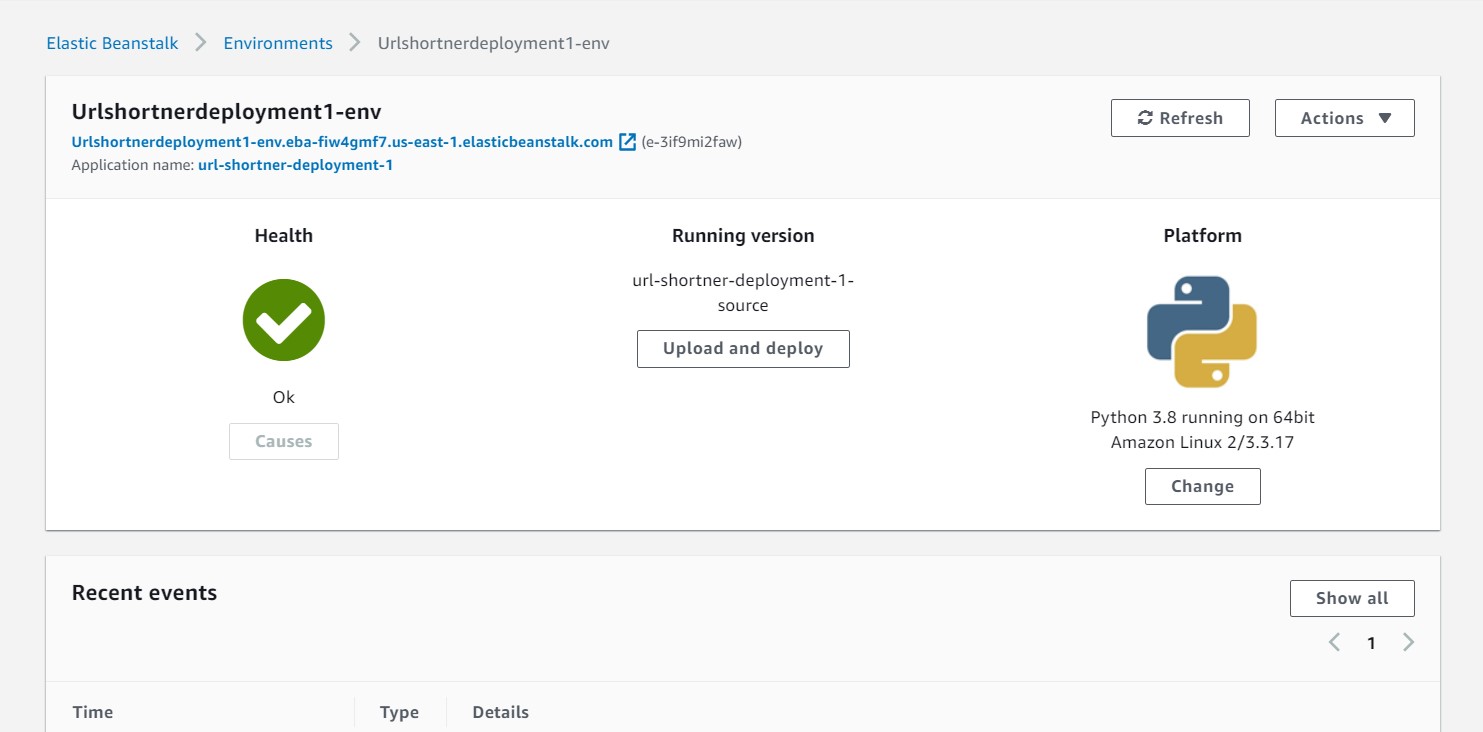
Finally entering the deployment stage of our pipeline. Due to conflicts I was unable to simply clone locally to my vm and zip it to deploy into Elasticbeanstalk. A combination of my limited hardware resources within my vm and being unable to send it over to my local machine. To bypass this I downloaded the initial URL-shortner zip from github then unzipped it to a folder, and finally rezipped only the contents of the folder. Making sure to leave out the folder itself.

From then on I created a web server environment



Then after selecting python as my platform, I uploaded the URL-shortner.zip



From then on it was a matter of waiting for the environment to be constructed. My result was a success

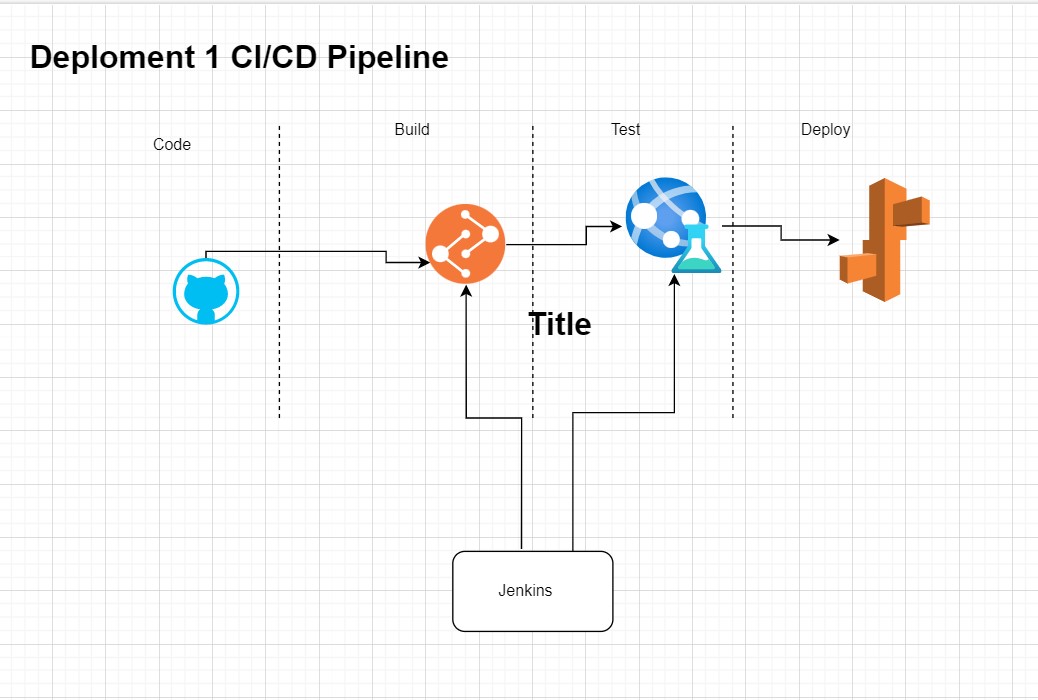
To confirm the environment loaded the code correct I launched the elastic beanstalk url in my browser

Graphical user interface, application

Description automatically generated

Confirmation the app was successful deployed.

Deployment Pipeline:



Improvements

* Add monitoring, currently there almost no way to actually monitor how our program is doing once it passes the initial testing phase. Thus we’re in the dark when it comes to the performance of our application. Using a service like Cloudwatch would be great to give us live and up to date information based on our environment’s performance.
* Testing, currently there is only 1 test to confirm whether our program is viable. However, it may prove insufficient should another aspect of the program be coded incorrectly.
* Finally, lack of automation. Currently the pipeline is manual once it passes the testing phase. Jenkins must also manually be triggered to create a new build once there are updates to the repository. There is manual delivery to the elastic beanstalk environment.