HARVARD
Extension School

E91 Cloud DevOps Final Project

Group 1

Names & assigned task(s):

Item	Task(s)	Contributor Name
1	**AWS setup, Dev & Stage, Logging/Monitoring Developer** • Set up the AWS infrastructure and AWS client instances, create CFT • Set up Dev and Staging container shell scripts and CFTs for jenkins to use • work with adjacent team members to integrate CFT creation/destruction of containers into Jenkins for dev and stage and push to Prod • Work on Logging/monitoring for AWS • Documentation Editor	Christine Liu
2	**Jenkins Developer** • Define AWS Jenkins setup, and set up pipelines. • Work with adjacent team members to set up webhooks.	Charles Mateer
3	**Testing Developer** • Create test scripts and tools potential on jenkins • implement Python HTML Parser	Jingjing Zheng

4	**Production Instance (GCP Instance setup, Docker container, webhook server)**
	• **Prod Instance**
	  ○ Instance setup
	  ○ Container & Site setup
	  ○ Github webhook server setup
		Stephen Akaeze

5	**GCP setup, Ansible Server, Docker file, Logging/Monitoring Developer** • Set up GCP environment and templates • create and manage ansible server on GCP with ansible playbooks • Create Dockerfile to be used for Dev, stage containers. • Work on logging/monitoring for GCP • work with adjacent team members to integrate CFT creation/destruction of containers for dev and stage into Jenkins and push to Prod	Philip Cunningham
6	**Front end Developer** • Create full scale website (HTML, CSS, JS) for public GIT repo to demonstrate real life CI/CD workloads.	Elias Falconi

Repos URLs:

- Public Repo: https://github.com/cwmat/csci-91-final-phase1-public
- Private Repo: https://code.harvard.edu/cwm850/csci-91-final-phase1-private

Submission Date: 12/19/18

HARVARD
Extension School

Table of Content

PROJECT OBJECTIVE **3**

PROJECT APPROACH **4**

PLANNING 4

Roles 4

Architecture 5

Project Set Up Plan 6

Meeting frequency and goals 6

IMPLEMENTATION 6

Dockerfile - Philip Cunningham 7

AWS Infrastructure Setup - Christine Liu 7

AWS Base Environment CFT template 7

Dev EC2 CFT template 9

Staging EC2 CFT template 9

GCP Infrastructure Setup - Philip Cunningham and Stephen Akaeze 10

Ansible Server instance - Philip Cunningham 10

Prod Instance - Stephen Akaeze 14

Prod Server Instance, Container Setup and Webhook Configuration - Stephen Akaeze 15

Purpose 15

Implementation 15

Jenkins Pipeline and WebHooks - Charles Mateer 17

Initial Setup 17

Install Plugins and Configure Git Webhooks 19

Setup Pipeline 21

Freestyle Job Configurations 23

Test Cases - Jingjing Zheng 26

The site is up 26

The content is what you expected 26

Website - Elias Falconi 26

Background 27

Initial setup 27

Responsiveness 30

JavaScript 32

Miscellaneous info 33

Logging/Monitoring - Christine Liu & Philip Cunningham 34

Purpose 34
Logging/Monitoring in AWS 34
Logging/Monitoring in GCP 35
Checking Logs 35
FOLDER DESCRIPTIONS 37
Private Github 37
Public Github 37

FINAL PROJECT FUNCTIONALITY 38
INITIAL WEBSITE 38
Link to the website 38
Image of the initial website 38
OUTLINE OF CHANGES 39
Changing Files 39
Jenkins CI/CD process 41
RESULTS 43
Change results reflected in Github 43
Change results reflected on site 44

DOCUMENTATION FOR USER 45
E91 CLOUD DEVOPS FINAL PROJECT PHASE 1 45
MOTIVATION 45
HOW IT WORKS 45
CONTRIBUTING GUIDELINES 45
AUTHORS 46

PROJECT OBJECTIVE

Using DevOps techniques learned in class, group 1 will develop and maintain an open source website project that accounts for minimal/no funding and security in a single container. We will use CI/CD techniques to build a pipeline for a website that needs a seamless, end-to-end continuous delivery and deployment workflow. A continuous integration pipeline will support test, configuration, and deployment of a website and a continuous delivery pipeline will support automatic building, testing, and release of code changes to production environment.

HARVARD
Extension School

PROJECT APPROACH

In the project approach section, group 1 will explain how we collaborated. We document our planning and implementation processes and results.

PLANNING

To ensure that DevOps was used properly and efficiently, we had several planning sessions to discuss the CI/CD requirements, architecture, roles, infrastructure build-out plan, and how to approach development, code reviews, and version control.

Roles

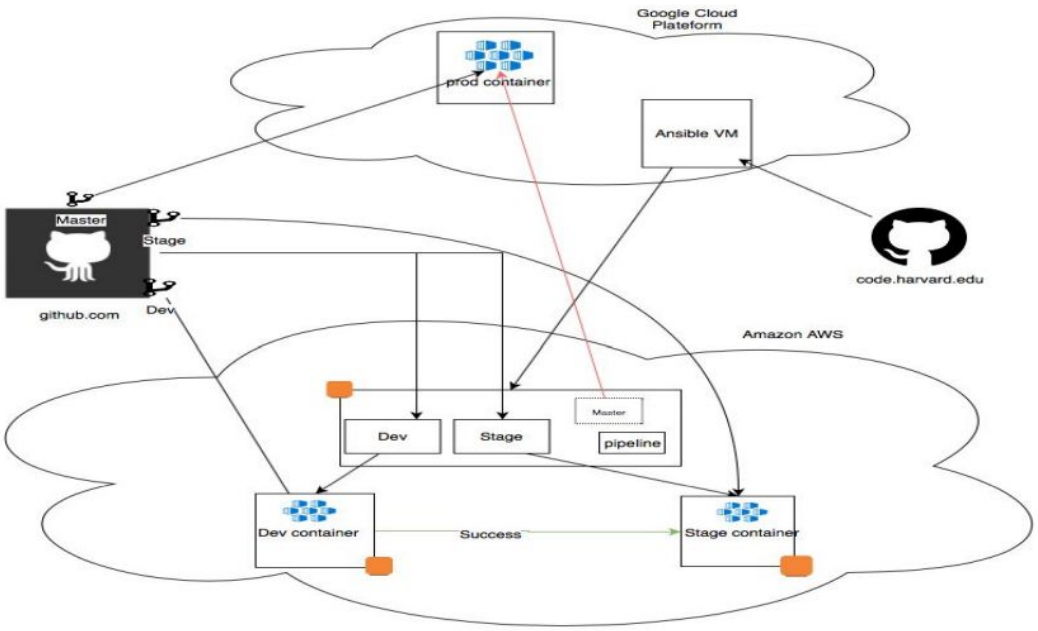
As a group, we brainstormed and decided the best way to organize and divide the roles to ensure efficient project buildout. Although we had roles, the group was in constant communication to ensure timely project development. All group members agreed and contributed to overall solution and troubleshooting promptly.

- **Testing Developer (Jingjing Zheng):**
  - Create test scripts and tools potential on jenkins
  - implement Python HTML Parser
- **Front end Developer (Elias Falconi):**
  - Create full scale website (HTML, CSS, JS) for public GIT repo that demonstrates a real CI/CD workload
- **GCP setup, Ansible Server, Docker file, Logging/Monitoring Developer (Philip Cunningham):**
  - Set up GCP environment
  - create and manage ansible server on GCP with ansible playbook
  - Create Dockerfile to be used for Dev, stage containers.
  - Work on logging/monitoring for GCP
  - work with adjacent team members to integrate Jenkins triggered/destroyed CFT templates for Dev and Staging and push to Prod
- **Production Instance (Stephen Akaeze):**
  - **Prod Instance**
    - Instance setup
    - Public repo & Container & Site setup
    - Github webhook server setup
- **Jenkins Developer (Charles Mateer)**:
  - Define AWS Jenkins setup, and set up pipelines.
  - Work with adjacent team members to set up webhooks.
- **AWS setup, Dev & Stage, Logging/Monitoring Developer (Christine Liu)**:
  - Set up the AWS infrastructure for AWS client instances, create CFT

- Set up Dev and Staging containers with CFT templates and shell scripts
- work with adjacent team members to integrate Jenkins triggered/destroyed CFT templates for Dev and Staging and push to Prod
- Work on Logging/monitoring for AWS
- Documentation Editor

Architecture

This was the architecture explained in Week 13 lecture to be used for the final project. We will be using CI/CD techniques to deploy software rapidly, repeatedly, and reliably. Testing and Staging will be hosted in AWS, while production will be hosted in GCP. The Ansible VM will set up the Jenkins. There will be a private repo to store our infrastructure templates (cloud formation and GCP templates) and any other sensitive data files, keeping confidential user data private. There will also be a public repo to store website code, test cases, and code that do not pose a security concern that other developers can use and improve. When a developer commits a change to the public repo dev branch, Jenkins will trigger the creation of a dev container and run through tests. If the tests pass, Jenkins will spin up a stage container and destroy the dev container. After running through staging tests, and if the tests pass, Jenkins will destroy the stage container and the changes will be committed to staging/master public repo and reflected in production. This architecture ensures automated testing and deployment of code that addresses the funding concern for dev and stage EC2 containers by only creating these test VMs when a change has been committed and tearing them down after.



Project Set Up Plan

To keep the project organized and efficient, we decided whose accounts we would use and who would set up the code repository and SSH key for us to use.

Accounts: As a team, we decided whose accounts to use in order to complete final project

- AWS account and roles - Charles Mateer
- GCP account and roles - Philip Cunningham

Code repositories and SSH key: Charles Mateer was responsible for setting up the code repos that the team will be using and generating the final project SSH key.

- Public Repo: https://github.com/cwmat/csci-91-final-phase1-public
- Private Repo: https://code.harvard.edu/cwm850/csci-91-final-phase1-private

*Infrastructure Development planning and implementation documented in Implementation Section

Meeting frequency and goals

In order to complete this project, we performed as a DevOps team.

- **Group Chat:** A group chat was set up so that constant, immediate communication was possible.
- **Scrum Calls:** Scrum calls were set up every other day to go through progress, raise concerns and dependencies, solve issues, and troubleshoot.
- **Code reviews/Version Control:** As a group, during our scrum calls, we would discuss our updates and ask for verbal approval of code changes. In addition, we learned how to implement git pull requests.
- **Pair programming:** We also practiced pair programming, where we would work with another person, checking before committing code.
- **Live Working sessions:** In addition to pair programming, we also held a lot of teamwide working sessions for holistic system development, integration, and testing via video conferencing.

IMPLEMENTATION

In this section, we detail how we implemented the architecture for this open source project. Each group member documented their assigned part of the project throughout this section. As described in the architecture section, our infrastructure spanned across AWS and GCP. Jenkins, Stage, and Dev are in AWS and prod and Ansible server are in GCP. When a developer pushes a commit to the dev branch of the public repo, the change will automatically trigger Jenkins to create a dev environment and run through some tests. If the tests pass dev, the changes will move onto staging and Jenkins will trigger creation of a staging environment, initiate some tests, and the destruction of the dev environment. If the tests pass staging, the

staging environment will terminate and the changes will be reflected in public repo staging branch, the master branch of public repo, and prod.

Dockerfile - Philip Cunningham

The Dockerfile is used in the shell scripts used to install docker containers on the VMs. The Dockerfile was tailored and configured to work with the container setup script to be used in all infrastructure setup. There are two docker files for the project, one based on the file from class and a second docker file for ubuntu based OS that creates an apache2 server.

```
# 1) For ubuntu GCP prod
FROM    ubuntu:latest


# 2)
RUN     apt-get update && apt-get install -y sudo && rm -rf /var/lib/apt/lists/* && sudo apt-get update -y && sudo apt-get install apache2


# 3)
EXPOSE  80


# 4)
CMD     ["/usr/sbin/apache2ctl","-D","FOREGROUND"]


# 5) may need to copy multiple files and dir
COPY    index.html /var/www/html/
COPY csci-91-final-phase1-public/* /var/www/html/
```

The docker file installs apache2, exposes port 80, runs on the container and copies the open source public project and copies the website files and folder structure to the container.

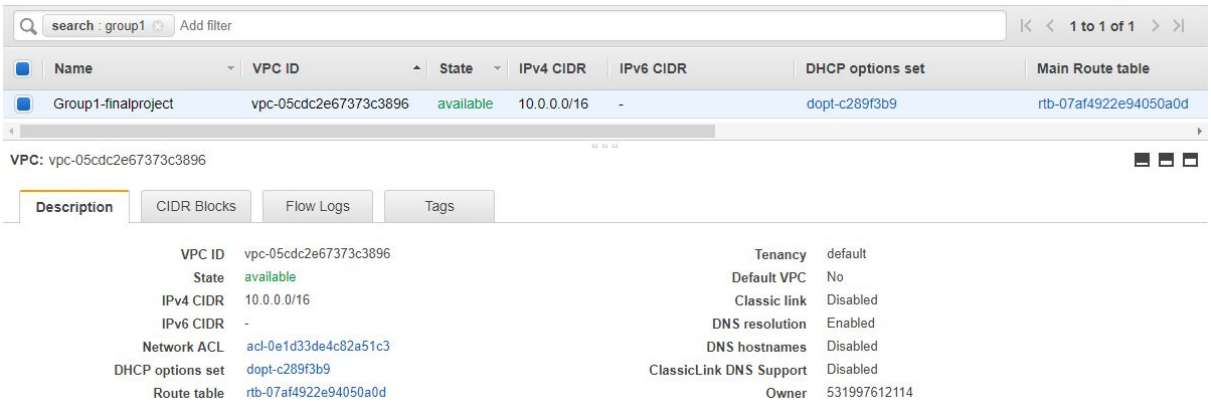
AWS Infrastructure Setup - Christine Liu

As discussed and agreed within group 1, we decided to have a total of 3 cloud formation templates in the private repo for the AWS infrastructure. I contributed to mostly the private repo.

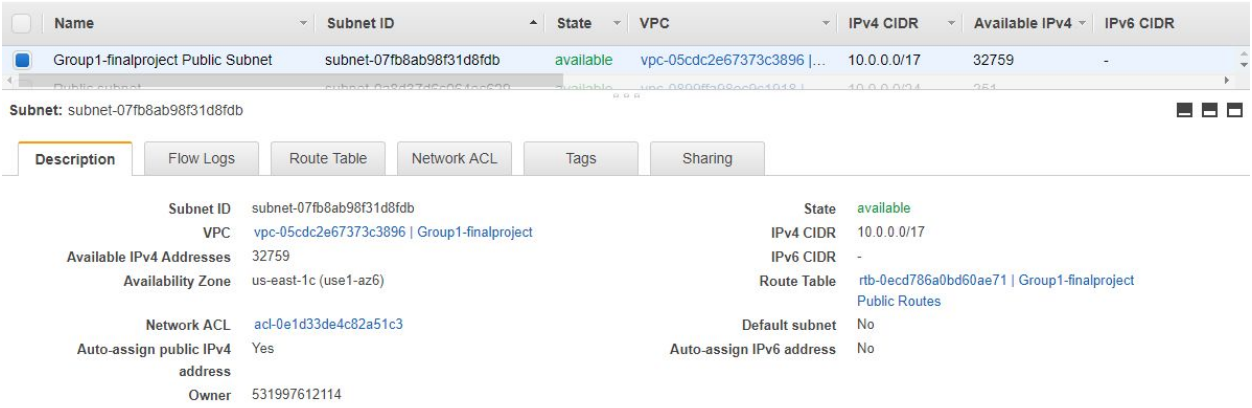
AWS Base Environment CFT template

The first CFT called 0-deploy-vpc-subnet-sg-jenkins.yaml contains the code to setup the VPC, Subnet, security group, and Jenkins EC2 (https://code.harvard.edu/cwm850/csci-91-final-phase1-private/blob/master/cloud-formation/0-deploy-vpc-subnet-sg-jenkins.yaml). It is located in the private repo.

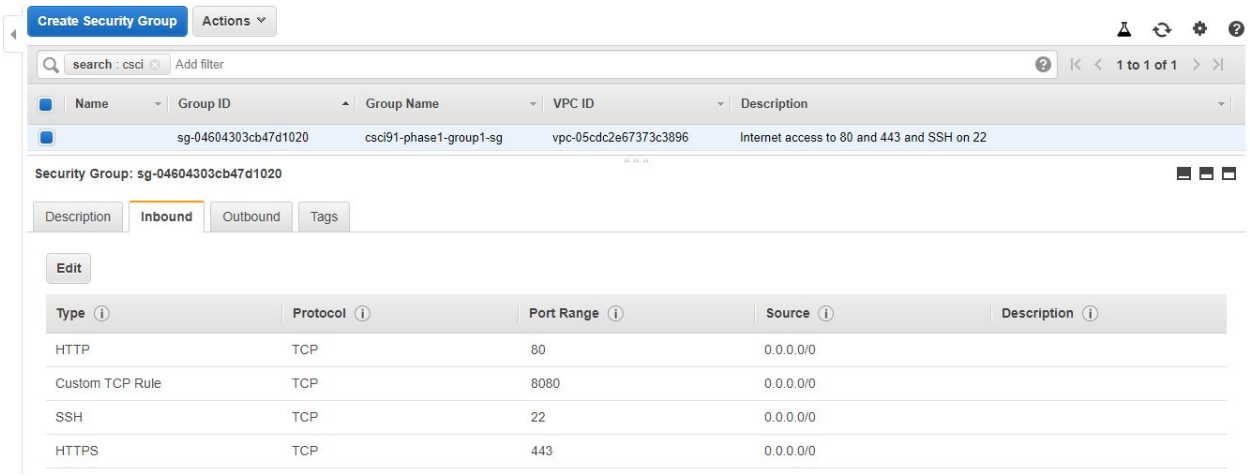
- **VPC**: Group1-finalproject



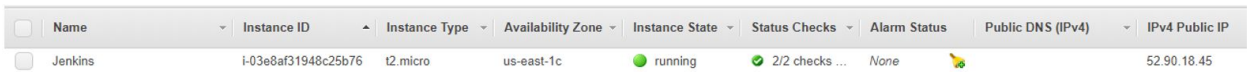
- **Subnet**: Group1-finalproject Public Subnet



- **Security Group:** csci91-phase1-group1-sg



- **Jenkins EC2:** Public IP of 52.90.18.45



Dev EC2 CFT template

The second template called 1-deploy-dev.yaml is a CFT that Jenkins will trigger to create the dev container EC2 when a change is made to the public repo dev branch. (https://code.harvard.edu/cwm850/csci-91-final-phase1-private/blob/master/cloud-formation/1-deploy-dev.yaml). It is located in the private repo. The template will:

- Create a CentOS dev EC2 in the base VPC, subnet, with same security group as the base AWS infrastructure described in the first template
- Installs AWS CLI and uses AWS commands to push its IP address to an S3 bucket so that Jenkins and testing scripts can use it as an endpoint.
- Install Docker Container on EC2 by:
    - Installing, starting, and enabling docker
        - yum install docker -y
        - systemctl enable docker
        - systemctl start docker
    - Creating a directory and download the Dockerfile by curling Phil's Dockerfile and cloning the dev branch of the public repo to get the index.html file
        - mkdir /tmp/build-docker-image && cd /tmp/build-docker-image
        - git clone -b dev https://github.com/cwmat/csci-91-final-phase1-public.git
        - curl -O https://s3.amazon.com/final-e91/Dockerfilecentos
        - mv /tmp/build-docker-image/Dockerfilecentos /tmp/build-docker-image/Dockerfile
    - Building apache httpd image
        - docker build -t centosapache .
    - Running a container from the image and naming it dev
        - docker run --name dev -d -p 80:80 centosapache

Staging EC2 CFT template

The third template called 2-deploy-staging.yaml is a CFT that Jenkins will trigger to create the staging container EC2 when change has passed dev tests and moves to staging (https://code.harvard.edu/cwm850/csci-91-final-phase1-private/blob/master/cloud-formation/2-deploy-staging.yaml). It is located in the private repo. The template will:

- Create a CentOS dev EC2 in the base VPC, subnet, with same security group as the base AWS infrastructure described in the first template
- Installs AWS CLI and uses AWS commands to push its IP address to an S3 bucket so that Jenkins and testing scripts can use it as an endpoint.
- Install Docker Container on EC2 by:
    - Installing, starting, and enabling docker
        - yum install docker -y
        - systemctl enable docker
        - systemctl start docker

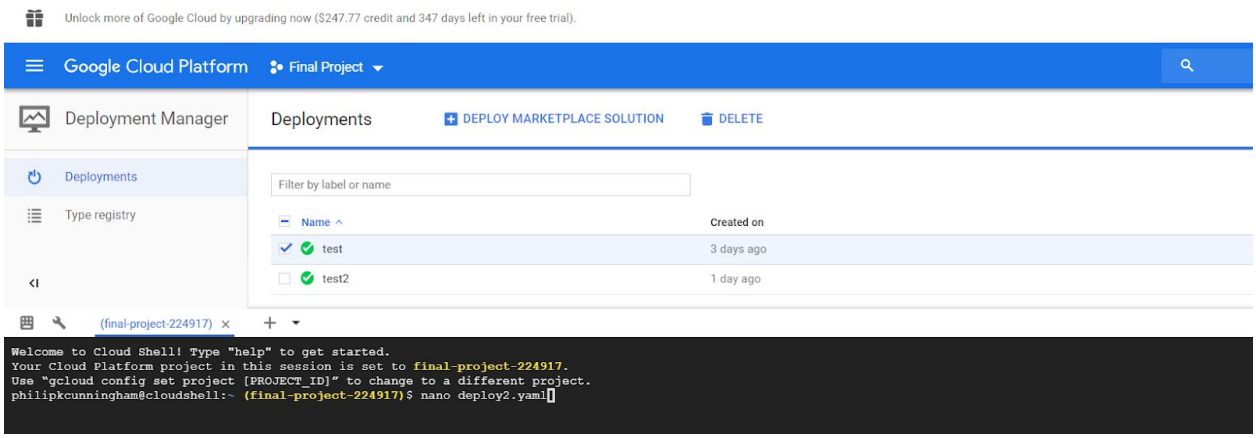
- Creating a directory and download the Dockerfile by curling Phil's Dockerfile and cloning the dev branch of the public repo to get the index.html file
  - mkdir /tmp/build-docker-image && cd /tmp/build-docker-image
  - git clone -b staging https://github.com/cwmat/csci-91-final-phase1-public.git
  - curl -O https://s3.amazon.com/final-e91/Dockerfilecentos
  - mv /tmp/build-docker-image/Dockerfilecentos /tmp/build-docker-image/Dockerfile
- Building apache httpd image
  - docker build -t centosapache .
- Running a container from the image and naming it dev
  - docker run --name stage -d -p 80:80 centosapache

GCP Infrastructure Setup - Philip Cunningham and Stephen Akaeze

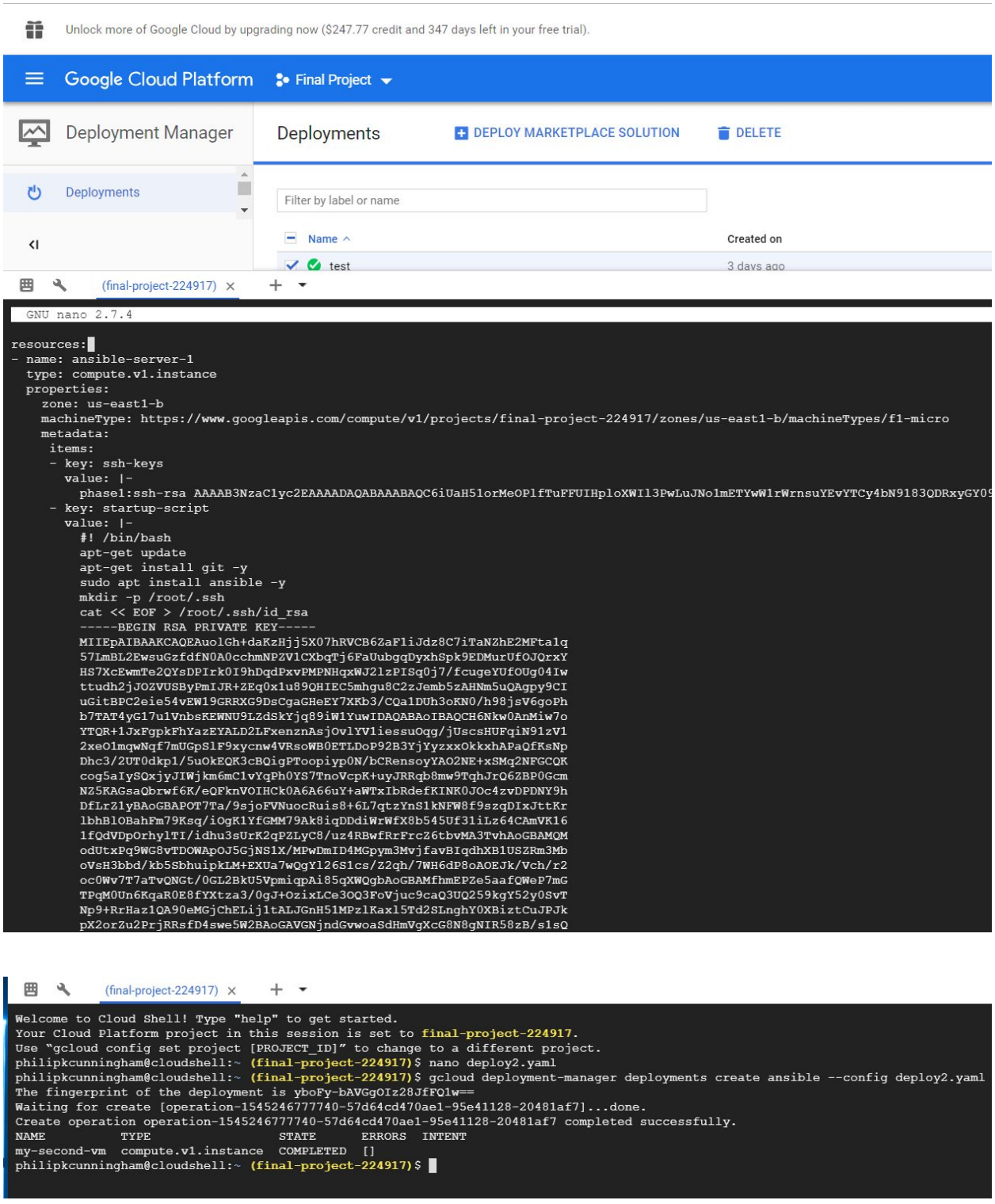
To set up the GCP infrastructure, two templates were made that sets up the prod instance and ansible server instance

Ansible Server instance - Philip Cunningham

GCP deployment manager to deploy ansible server. The yaml files are in the private repository under GCP templates.



The ansible server was configured to have the SSH keys public and private to perform GIT pulls, SSH into client instances and execute playbooks. The template will create the instance in the final project subnet with ubuntu 18.10 on the AZ us-east1-b and configured for a f1-micro machine type.



The ansible server can be launched in the GCP console by using deployment manager commands, naming the server, then referencing the deployment yaml.

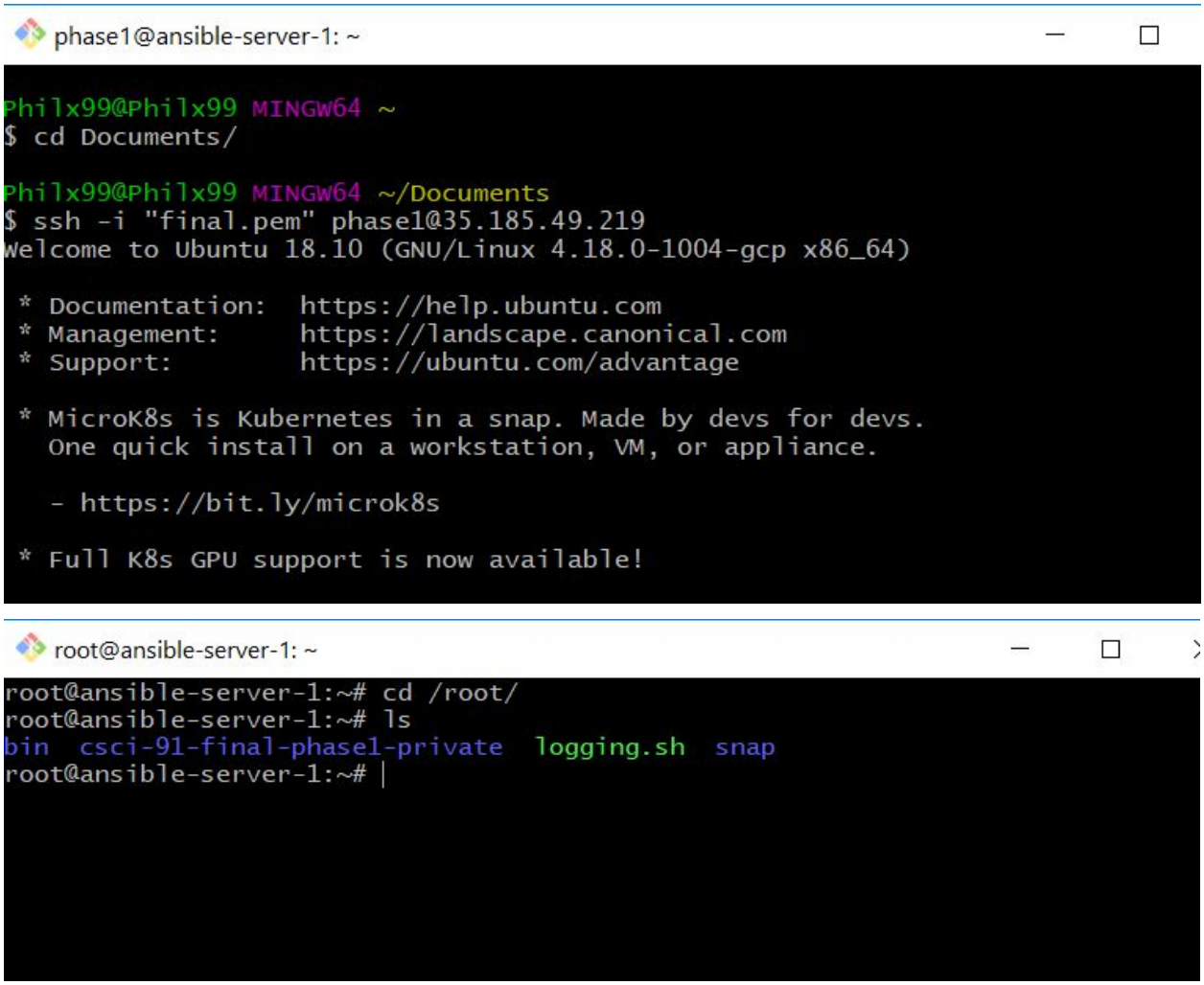
gcloud deployment-manager deployments create ansible --config deploy2.yaml

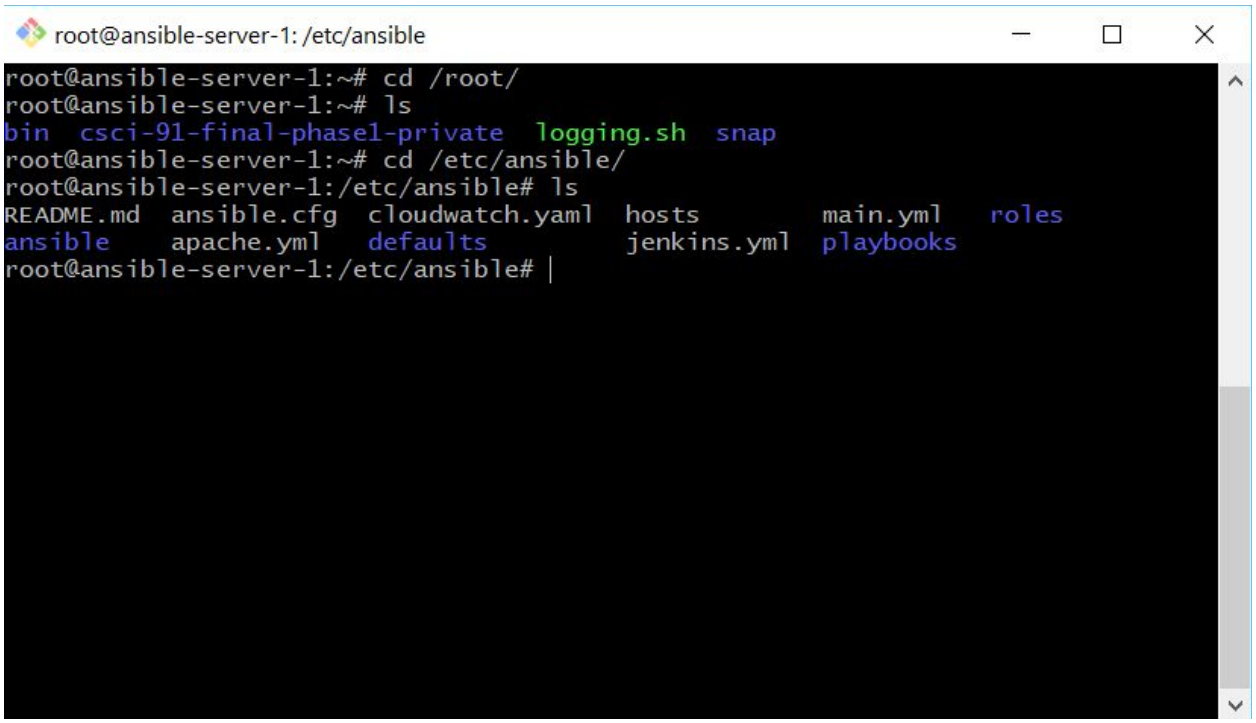
GCP deployment manager also makes it extremely easy to modify changes to the configuration yaml and can be updated while the deployment is running.

gcloud deployment-manager deployments update ansible --config deploy2.yaml

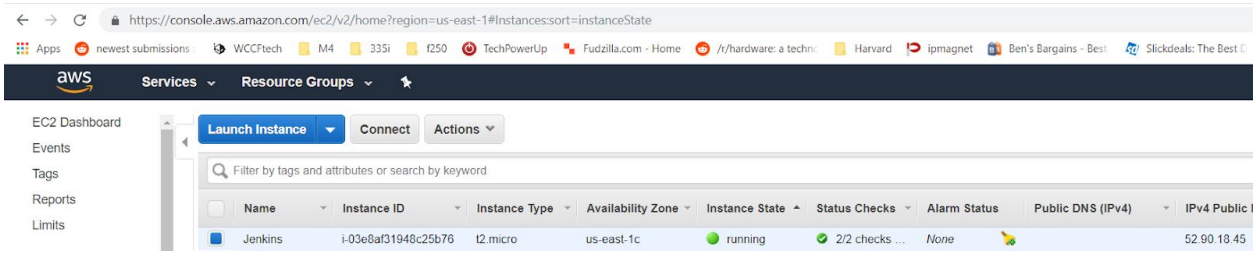
After the deployment manager run successfully a new instance is created in the GCP compute engine

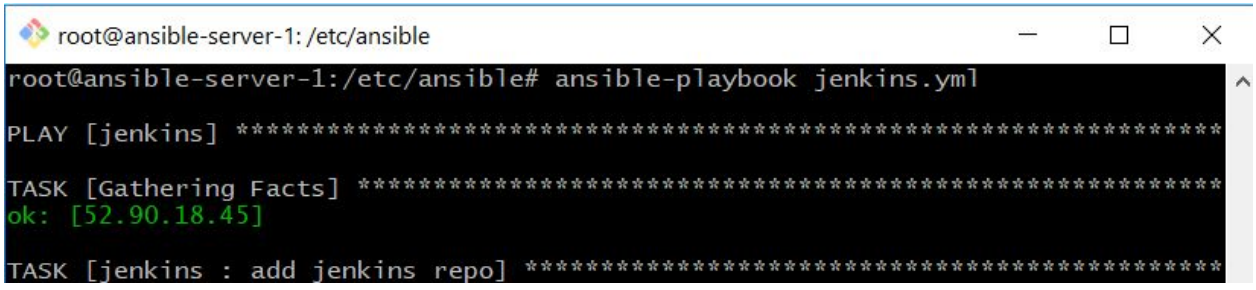
Ssh into the ansible instance and we can see that the Deployment manager kicked off the setup of ansible server, pulling the private git repo, moving the folder into /etc/ansible and ran the playbook to configure jenkins on the AWS instance based on the host file ip addresses in the private git repo.

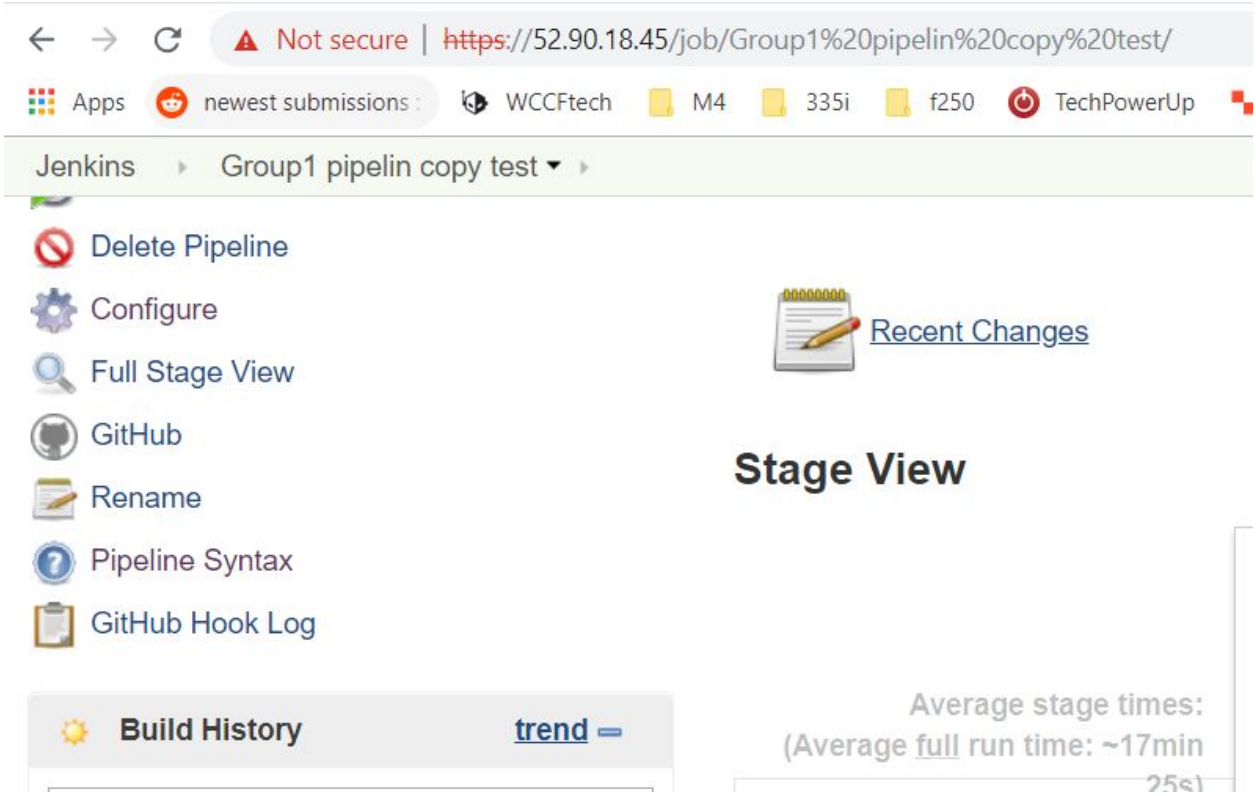


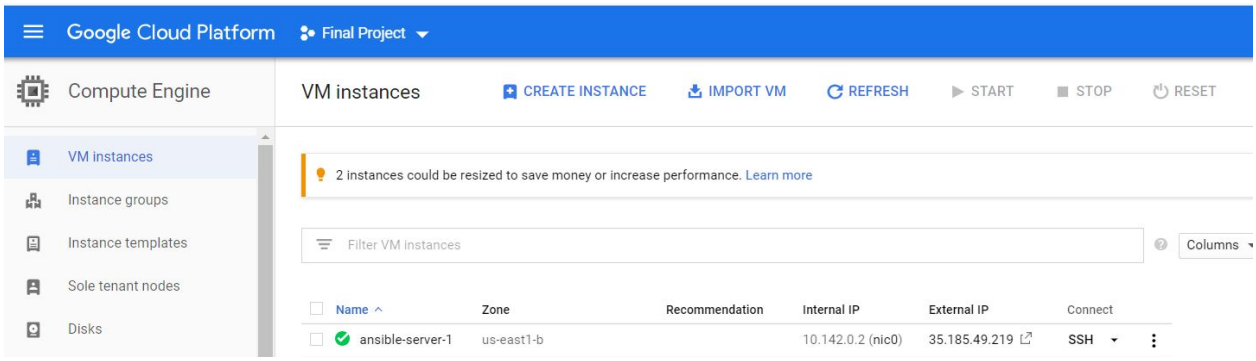


Jenkins playbook can be re run at any time provided that the updated IP address is updated in the host file.









Prod Instance - Stephen Akaeze

Described in next section

Image of GCP infrastructure setup results

Name	Zone	Recommendation	Internal IP	External IP	Connect
ansible-server-1	us-east1-b		10.142.0.2 (nic0)	35.185.49.219	SSH
final-prod-server	us-west2-a		10.168.0.3 (nic0)	35.236.24.202	SSH

HARVARD
Extension School

Prod Server Instance, Container Setup and Webhook Configuration - Stephen Akaeze

Purpose

As specified in the project requirements, the Prod server should meet the following requirements,

- Be located in the GCP infrastructure
- Setup a container running apache and sharing the website from the master branch of the public repo
- Set up an update mechanism which triggers the container site to be rebuilt whenever there is an update to public repo.

Implementation

Prod Instance Setup: The prod instance was setup using the GCP console as an n1-standard-1 (1 vCPU, 3.75GB memory) instance in us-west2-a availability zone. The selected operating system is Centos 07. SSH login was enabled using the group 1 shared public and private key pairs. The Prod public IP address is 35.236.24.202.

Prod Container Setup: The Prod container setup was done using a script (ProdConSetup.sh) which does the following,

- Updates the operating system
- Installs git and docker cli tools
- Stops the current docker container
- Deletes the current docker container and image
- Restarts Docker daemon
- Deletes the old public repo and clones a new public repo
- Creates the a directory for the docker container and moves the necessary apache files to the docker directory
- Creates a new docker container running apache with the Dockerfile

Based on the above steps, the script can repeatedly executed to successfully build new containers running apache.

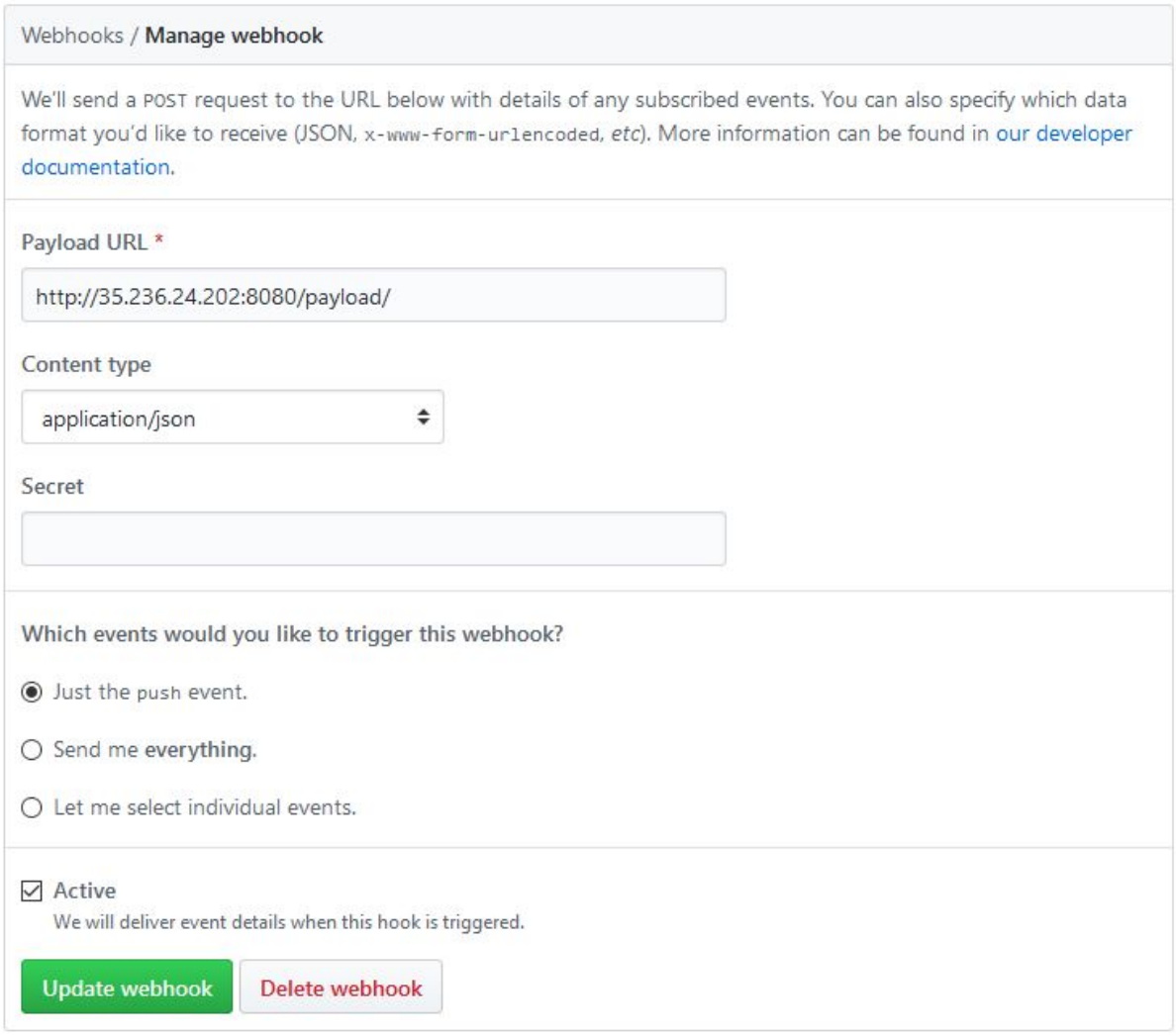
Container Update Mechanism: This essentially refers the implementation approach used in updating the Prod container/site after Jenkins has merged stage branch into the master branch on the public repo. The easy approach would be to use a cron job but cron jobs are not efficient as it wastes cpu resources. In a real world environment, the docker container might demand high processing or memory resources which could be compromised by a cron job. Cron jobs are also not every resilient. So we decided to use the github webhooks for the Prod server.

Webhooks are efficient as they are remote notifications informing the prod machine about a new push in the public repo. The webhooks are sent as HTTP POST from github to any registered server in the repo settings. The webhook implementation was handled in two steps

1) Step1 - Webhook server: Step 1 is accomplished using a NodeJS server (hookapp) running on the Prod instance. It listens utilizes Express.js as a middleware to listen for HTTP POST requests on port 8080 with path "/payload" (as shown hookapp/bin/ww and hookapp/app.js files). Whenever a HTTP POST request/webhook payload arrives on port 8080 with path "/payload" (essentially 35.236.24.202:8080/payload), the NodeJS server runs the ProdConSetup.sh which handles the repo update and docker container setup. The forever tool was used in running the server in the background as well as logging the ProdConSetup.sh output for reference as show below,

```
[[root@final-prod-server ~]# forever list
info:    Forever processes running
data:        uid  command       script          forever pid  id logfile                 uptime
data:    [0] z9b7 /usr/bin/node hookapp/bin/www 2891    2897    /root/.forever/z9b7.log 1:7:50:46.224
[[root@final-prod-server ~]# exit
```

2) Step 2 - Webhook server registration on github: Step 2 involves registering the NodeJS server as a webhook recipient for github. This accomplished by selecting add webhook from the public repo → settings ---> webhook page. The "Content Type" was set as Json and "Payload URL" set as "35.236.24.202:8080/payload", pointing the NodeJS server in the Prod server. The Event was set to as "push" as shown below



With both steps completed, NodeJS server successfully started receiving webhooks from github and rebuilds the prod container with every webhook payload received. By default, NodeJS is a single threaded and event-driven service so it is certainly more efficient that using a cron job.

Jenkins Pipeline and WebHooks - Charles Mateer

Initial Setup

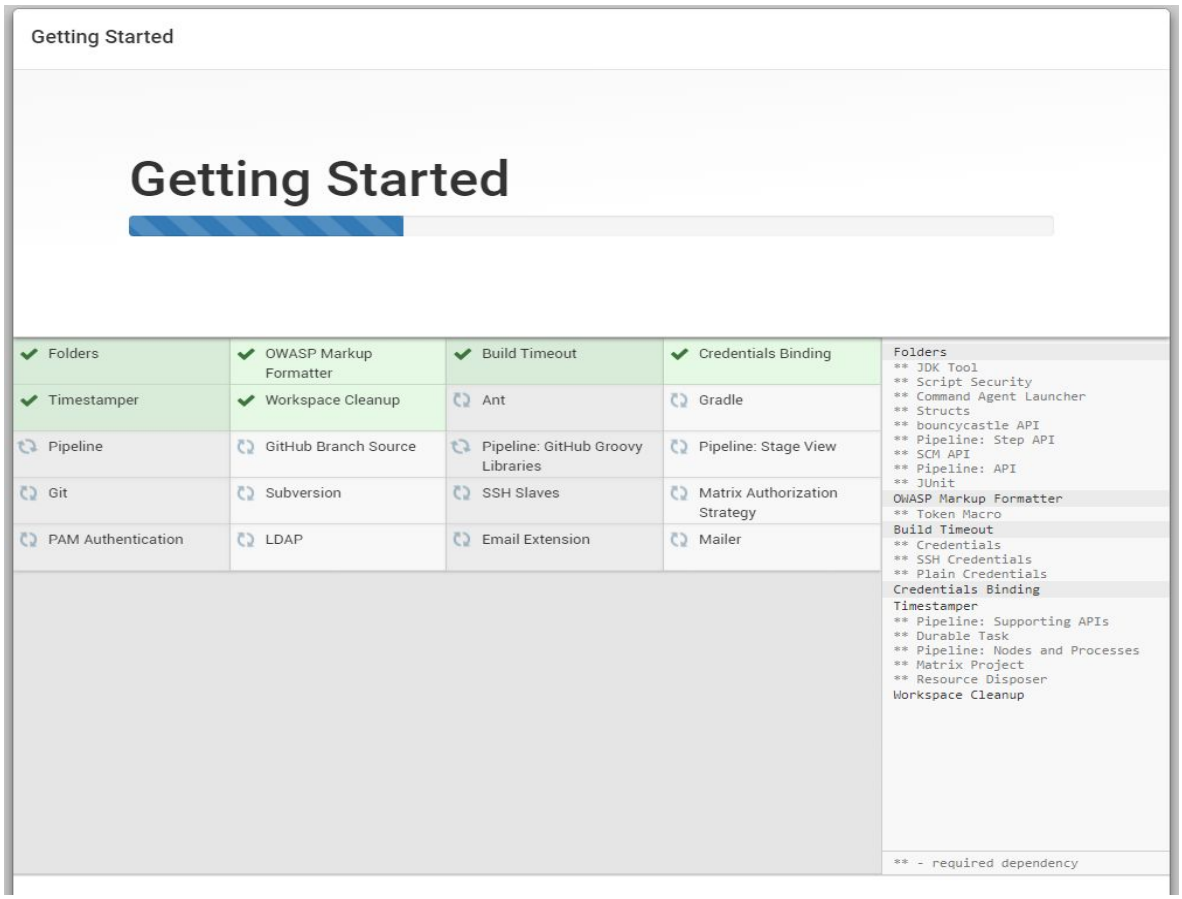
Once Ansible has installed Jenkins, go to the public IP for the Jenkins instance on HTTPS (port 443).

Next you will need to SSH to the Jenkins instance and:

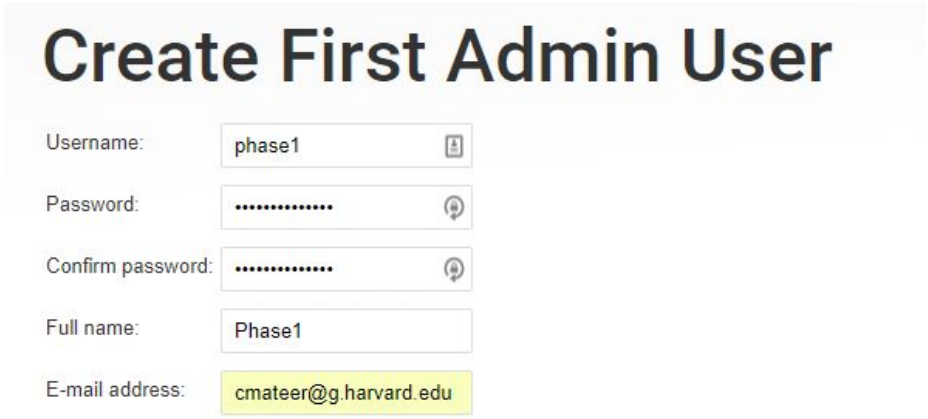
```
sudo su -
cat  /opt/jenkins/secrets/initialAdminPassword
```

and use the resulting password to unlock the Jenkins admin.

Choose to install suggested plugins:



Then setup an admin user:



Use the default Instance Configuration.

Install Plugins and Configure Git Webhooks

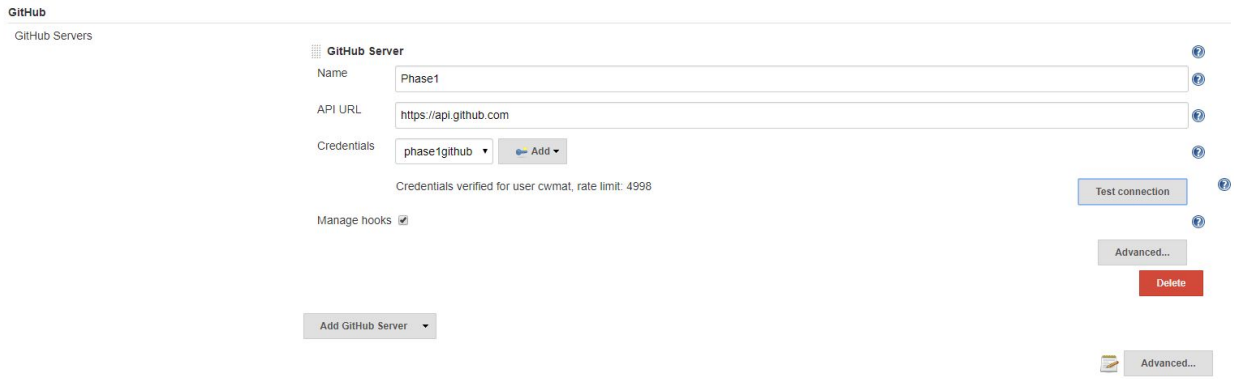
The Git plugin was installed as part of the suggested plugins.

Also install, Deploy over SSH.

Go to Credentials > System > Global and add a Secret Text entry with your GitHub Access Key.

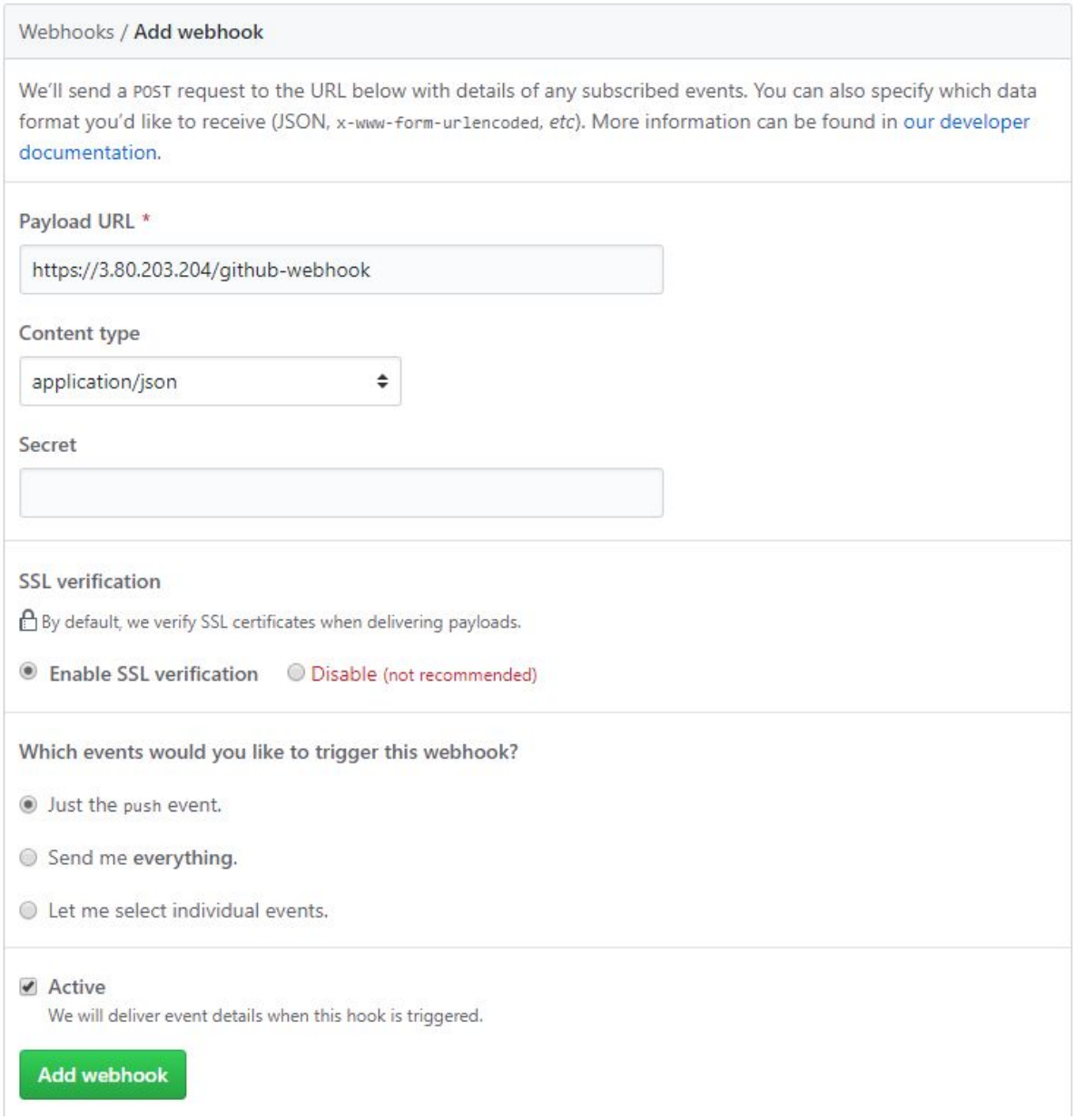
Go to Manage Jenkins > Configure System.

Scroll down to the GitHub Section and choose to Add a server:

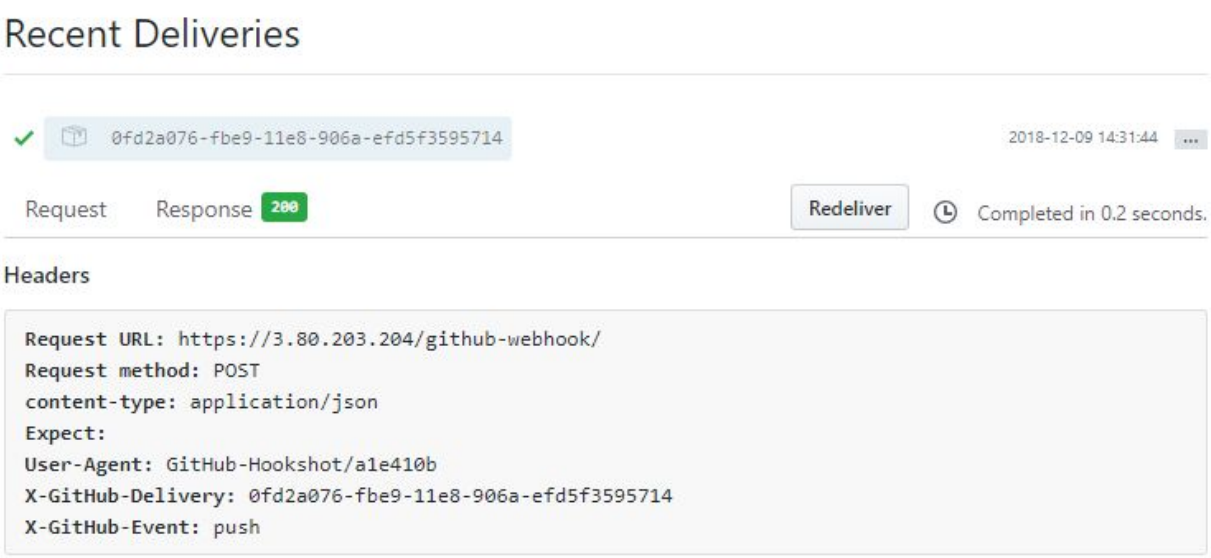


Test the connection to make sure it works.

Go to the public repo > Settings > Web Hooks and add a webhook.
Set the payload URL to the Jenkins server plus “/github-webhook” and the content-type to application/json.
Check Active and then update.

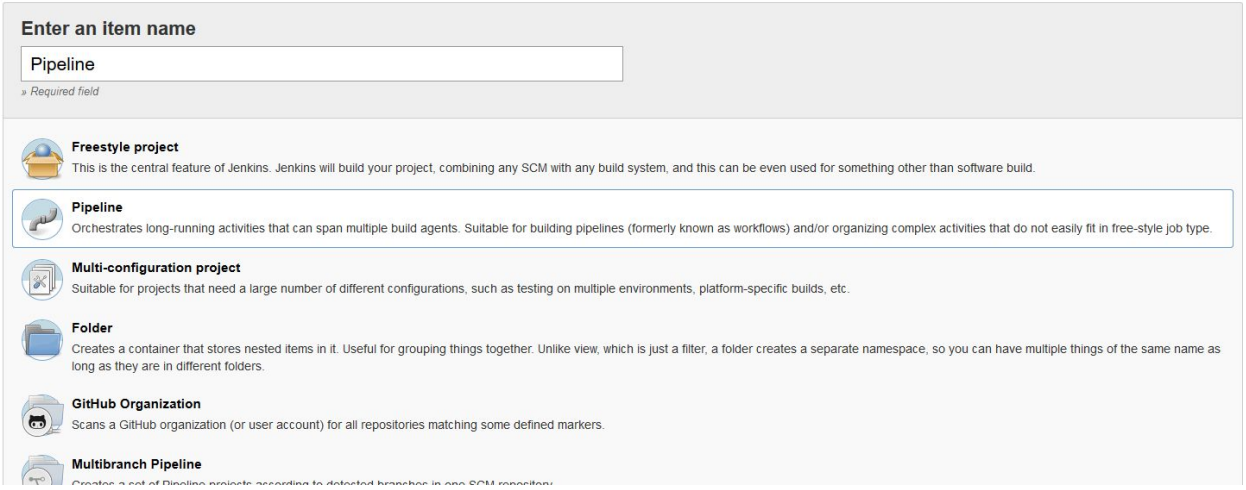


You should see a success down below:

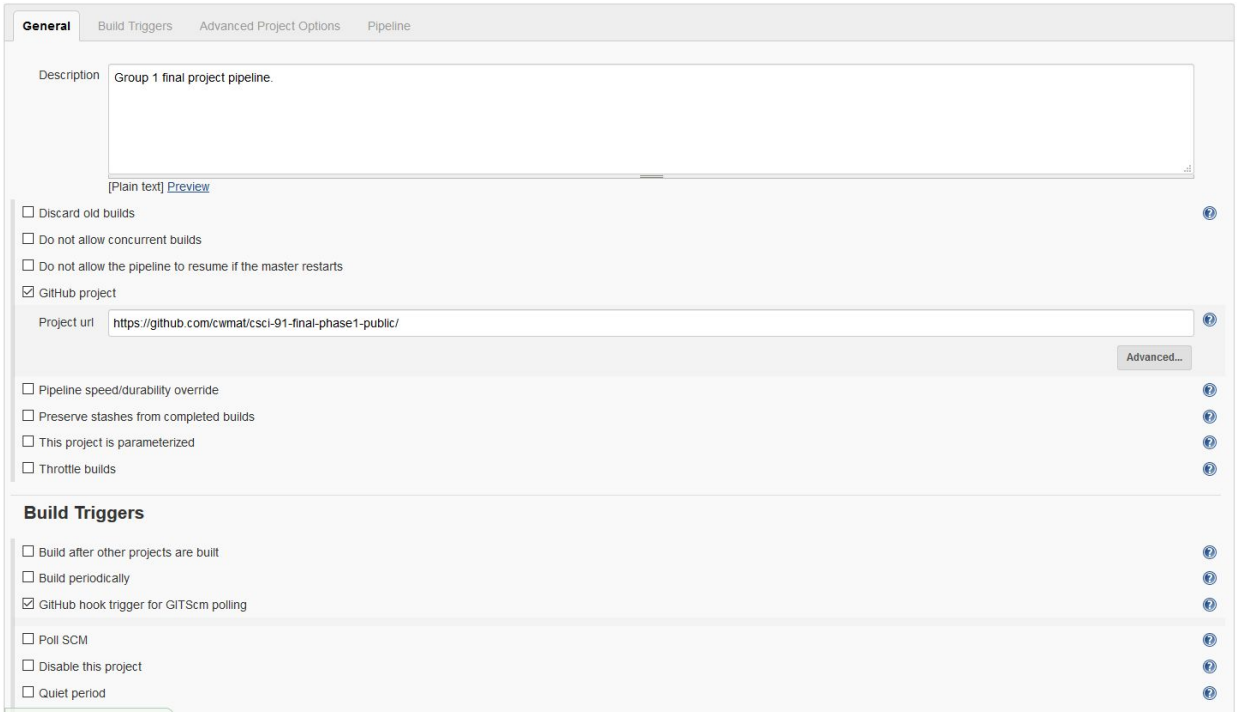


Setup Pipeline

Go to Jenkins > New Item and create a Pipeline Project:



Configure the pipeline to use the project's public repo and to be triggered by Git webhooks:



Under "Pipeline" you can now add the pipeline script from this repo `phase1-pipeline`.

The pipeline will now run whenever a commit is made to the dev branch of the public repo.

The pipeline uses two stand alone jobs which utilize the Git Publisher plugin (which is not supported in pipelines) to merge changes to the staging and master branches of the public GH repo.

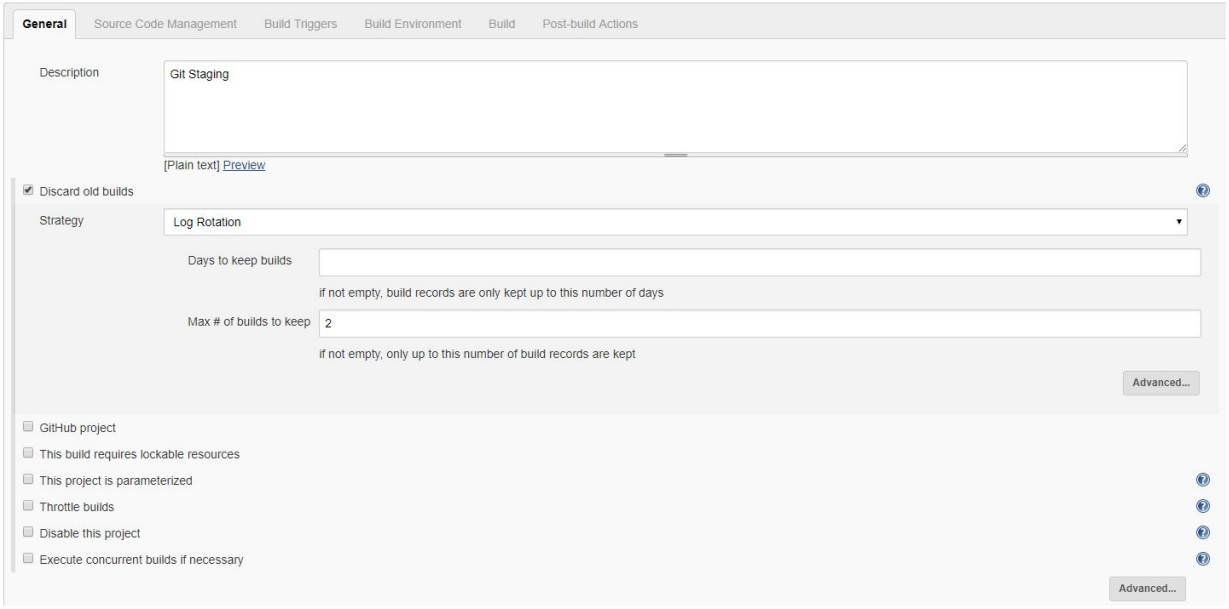
Additionally, the pipeline uses freestyle jobs using the **AWS Cloud Formation plugin** to create a CF stack from a CF template in the private repo.

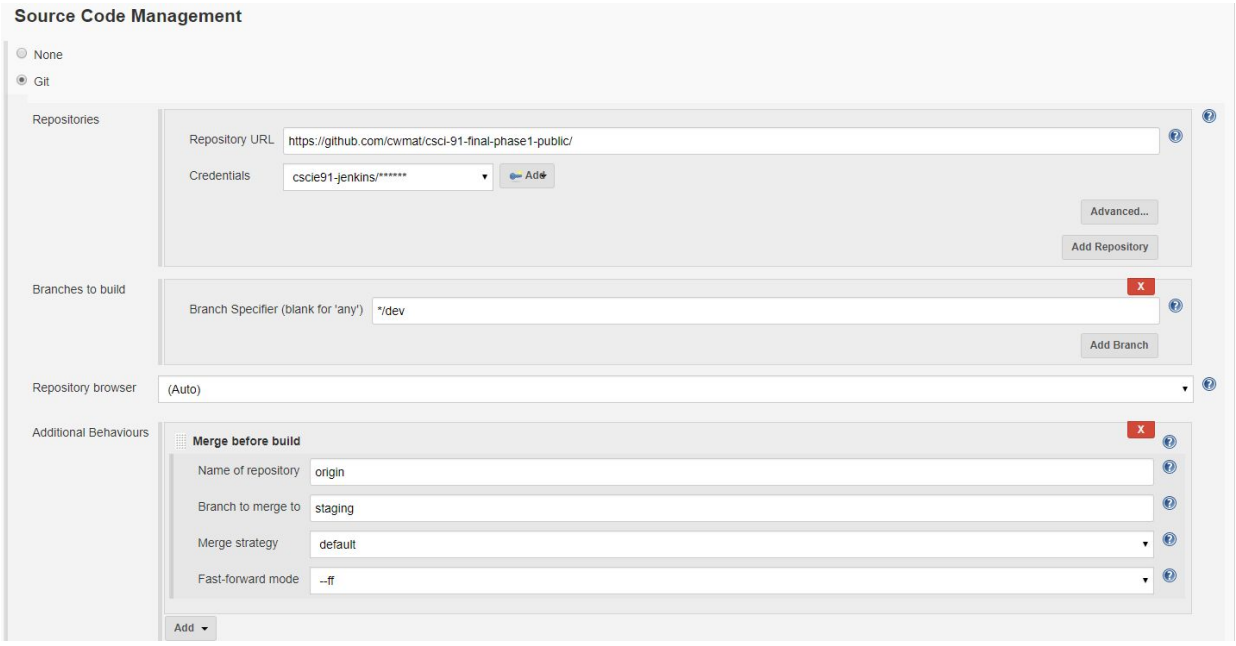
Once the CF stack is stood up, a docker script is invoked to build the project.

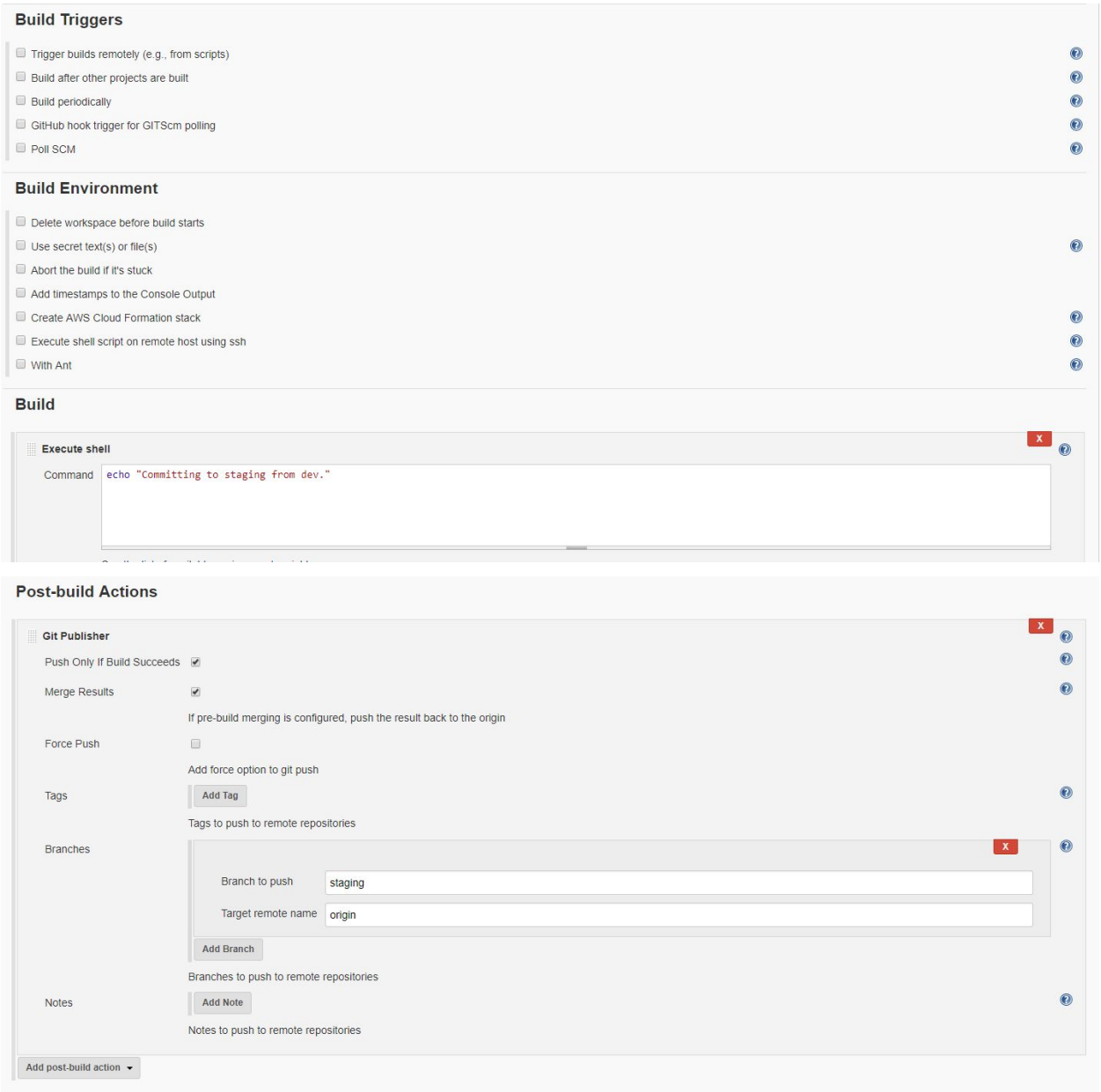
Once the project is built, it is tested. The IPs for the EC2 instance that Jenkins stood up are pushed to an S3 bucket which the Python test scripts CURL from to perform the tests. Upon success, the CF stack is torn down and the code is merged and committed to the next branch (staging or master).

Freestyle Job Configurations

Commit to Staging



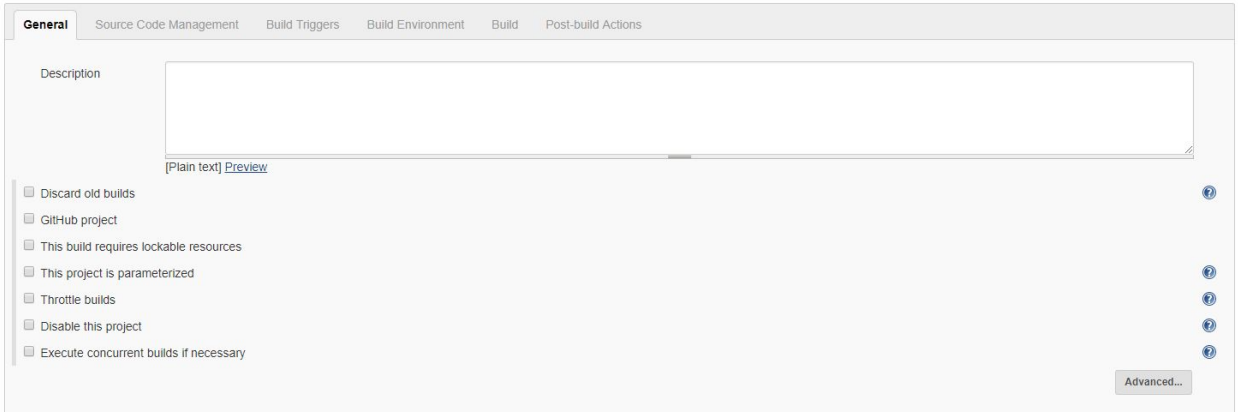


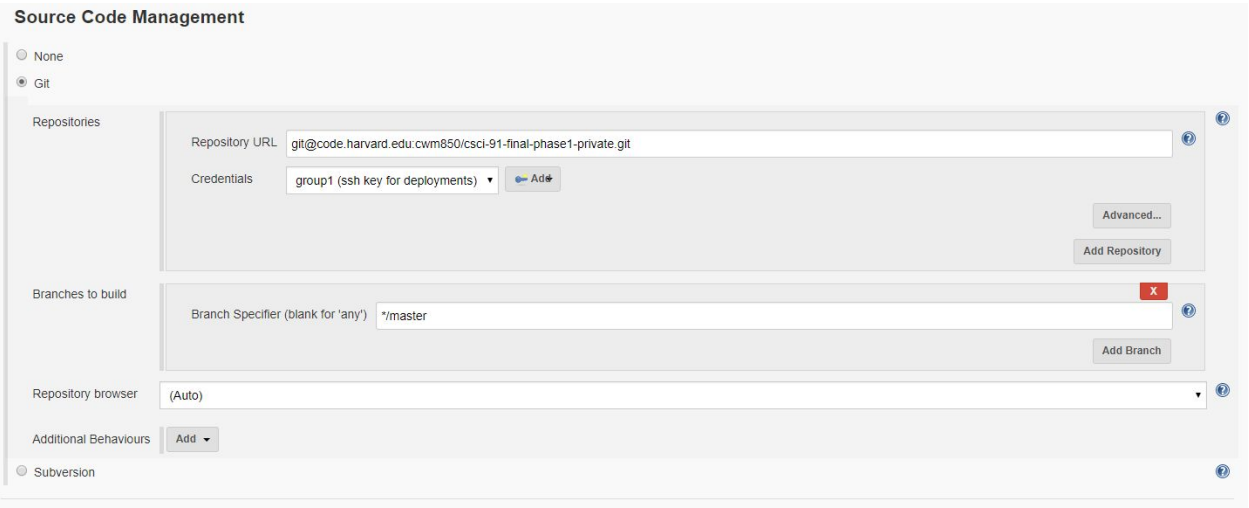


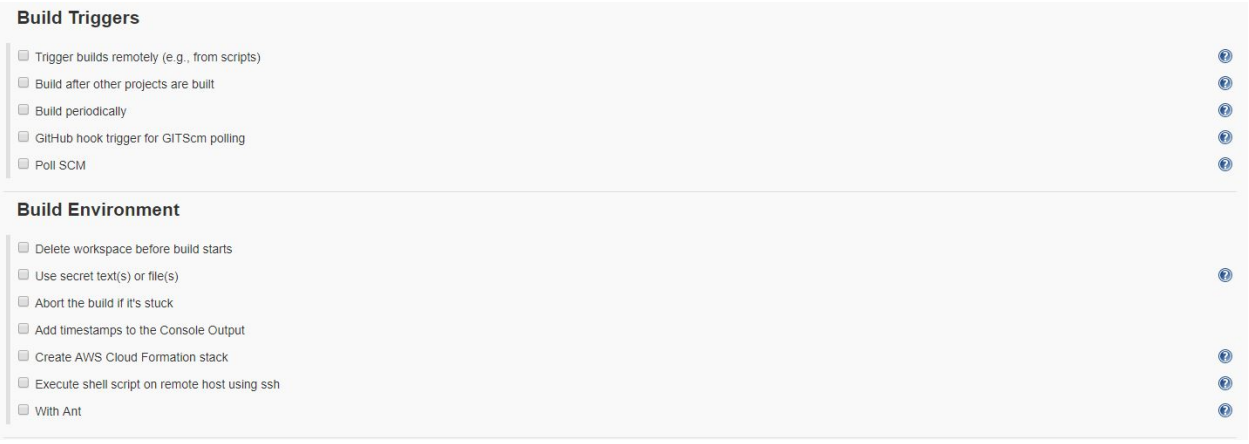
Commit to Master

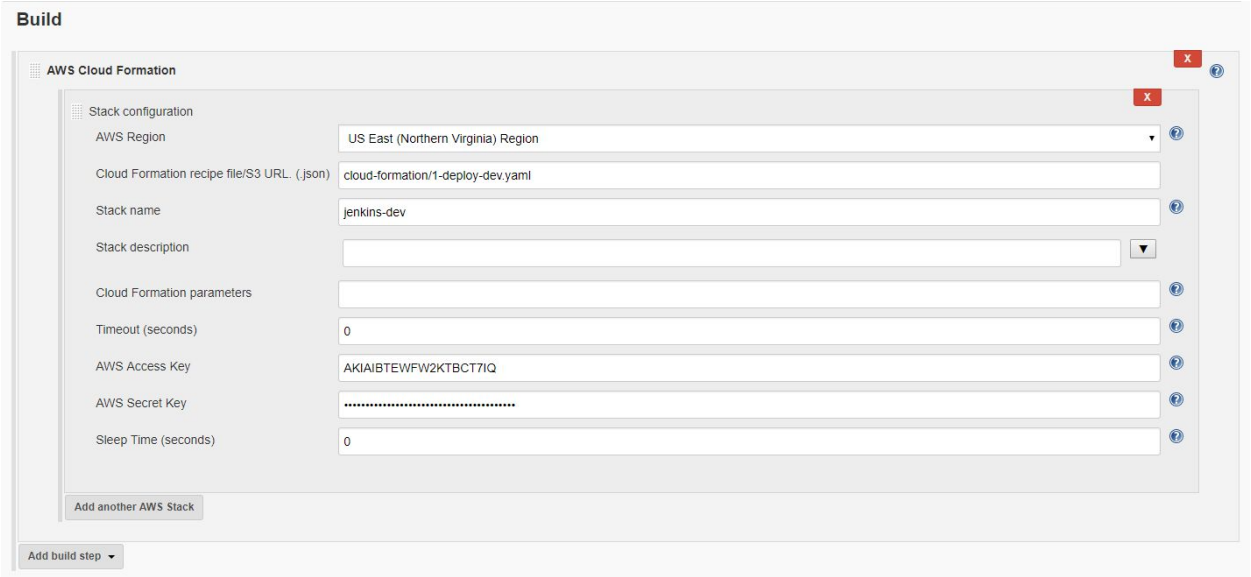
Same as above but the “Branch to Push” under Post-Build Actions > Git Publisher is “master”.

Create Dev CF


General
Source Code Management
Build Triggers
Build Environment
Build
Post-build Actions
Description
[Plain text] Preview
Discard old builds
GitHub project
This build requires lockable resources
This project is parameterized
Throttle builds
Disable this project
Execute concurrent builds if necessary
Advanced...


Source Code Management
None
Git
Repositories
Repository URL
git@code.harvard.edu:cwm850/csci-91-final-phase1-private.git
Credentials
group1 (ssh key for deployments)
Add
Advanced...
Add Repository
Branches to build
Branch Specifier (blank for 'any')
*/master
Add Branch
Repository browser
(Auto)
Additional Behaviours
Add
Subversion


Build Triggers
Trigger builds remotely (e.g., from scripts)
Build after other projects are built
Build periodically
GitHub hook trigger for GITScm polling
Poll SCM
Build Environment
Delete workspace before build starts
Use secret text(s) or file(s)
Abort the build if it's stuck
Add timestamps to the Console Output
Create AWS Cloud Formation stack
Execute shell script on remote host using ssh
With Ant



Create Staging CF
Same as above but invokes the staging CF script.

Test Cases - Jingjing Zheng

Two tests are created to deliver actionable feedback for each stage, make sure any changes/release are safe to proceed through the delivery pipeline.

The site is up

In this test, the script sends a GET request to the site, then the unittest will check the response status from the header. If the response status code is 200, the site is up, and it is reachable.

The content is what you expected

In this test, the script sends a GET request to the site hosting on dev / staging server. Then save the response data, which basically is a HTML formatted content. HTMLParser lib is used to extract content from a specific div tag. We use unittest verify whether the content is what we expected. Tests are ran in our project as below:

1. Testcase #1, content from title.
2. Testcase #2, content from section header

Website - Elias Falconi

The website developed is a fully functional site that sells ADT security (http://35.236.24.202/).
In the public repo, Elias is both Usernamelias and Second Admin.

Background

This was a site I created for a client while briefly working as a web developer. The client wanted the site in WordPress. However, in this project, I recreated it as a static website. Since I no longer had the code for the site, everything was created from scratch. The only thing I had was the psd file, which showed exactly how the client wanted the site, with all the images, text, etc.

For this group project, we decided to create a full-blown website so that we can work under realistic CI/CD workloads and conditions. Having a website with far more content and code gives us more robust testing of the infrastructure, jenkins pipeline, and test cases. The project team wanted to go above and beyond the requirements outlined in class that required an additional role for web development. This additional role required a lot of collaboration with other team members and ensured that our infrastructure solutions were as flexible, scalable, and robust.

Initial setup

Usually when creating a website, I first start with the HTML and CSS simultaneously, and then later add functionality (either JS or server side code) which is how I approached this project. Also, I usually always use Bootstrap for styling, and jQuery for the JavaScript. So one of the first things I did was include the link and script tags for them.

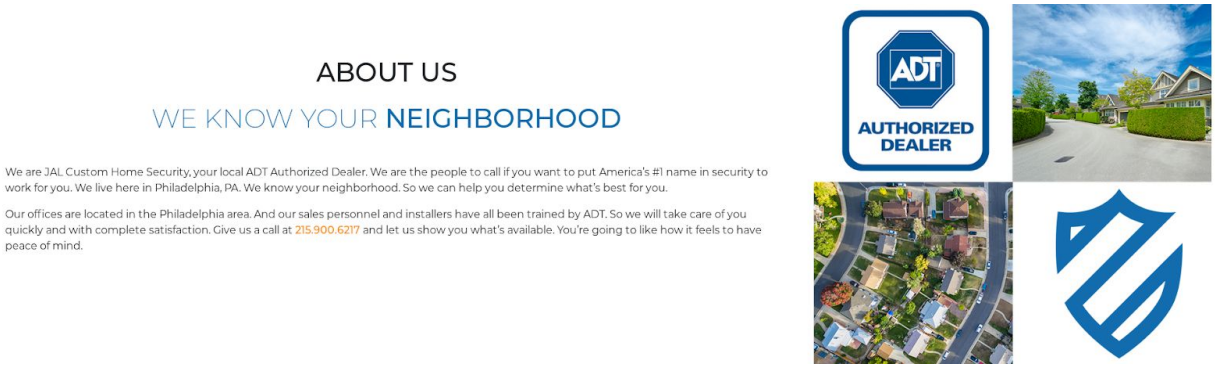
```
<head>
  <meta charset="utf-8">
  <meta http-equiv="x-ua-compatible" content="ie=edge">
  <title>CSCI-91 Public: Phase 1</title>
  <meta name="description" content="">
  <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
  <link rel="shortcut icon" href="/img/favicon.ico">
  <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.1.3/css/bootstrap.min.css" integrity="sha384-MCw98/SFnGE8fJT3GXwEOngsV7Zt27NXFoaoApmYm81iuXoPkFOJwJ8ERdknLPMO" c
  <link rel="stylesheet" href="css/normalize.css">
  <link rel="stylesheet" href="css/main.css">
  <link href="https://fonts.googleapis.com/css?family=Montserrat:thin,regular,medium,bold,100,200,300,400,500,600"
    rel="stylesheet">
  <link rel="stylesheet" href="https://use.fontawesome.com/releases/v5.5.0/css/all.css"
    integrity="sha384-B4dIYHKNBt8Bc12p+WXckhzcICo0wtJAoU8YZTY5qE0Id1GSseTk6S+L3BlXeVIU"
    crossorigin="anonymous">
</head>
```

```
<script src="https://code.jquery.com/jquery-3.3.1.min.js" integrity="sha256-FgpCb/KJQlLNfOu91ta32o/NMZxltwRo8QtmkMRdAu8="
  crossorigin="anonymous"></script>
<script src="https://stackpath.bootstrapcdn.com/bootstrap/4.1.3/js/bootstrap.min.js"
  integrity="sha384-ChfqqxuZUCnJSK3+MXmPNIyE6ZbWh2IMqE241rYiqJxyMiZ6OW/JmZQ5stwEULTy"
  crossorigin="anonymous"></script>
  <script type = "text/javascript"
  src = "https://ajax.googleapis.com/ajax/libs/jqueryui/1.11.3/jquery-ui.min.js">
</script>
/body>
```

After including the tags for Bootstrap and jQuery, I worked on the HTML and CSS for each section of the site, one by one. That is, first the top section:

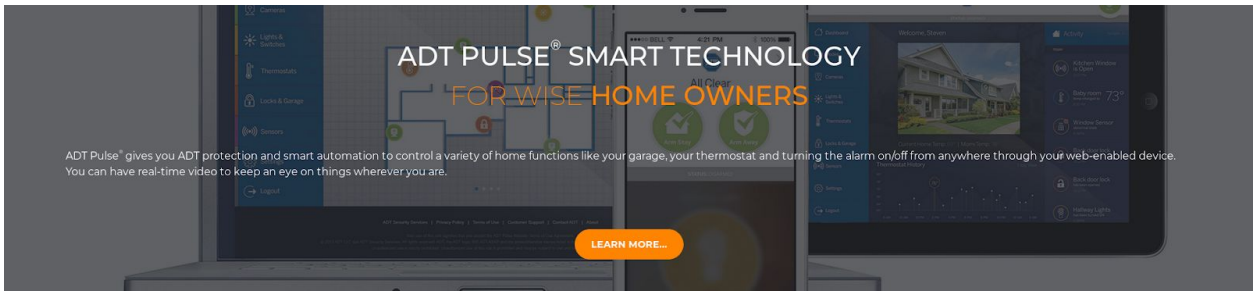
The top section includes everything in the screenshot above, and wrapped in a container (using Bootstrap's "container-fluid" class), with three rows. The first row contains the logo in the upper left corner and the contact info in the upper right corner. The second row is just the navbar. And the third row is everything else in the top section, divided into two columns. The first column has a grid width of eight, and contains the text "GET ADT MONITORED..." and the two buttons beneath it. The second column has a grid width of four, and contains the contact form. Lastly, I gave the whole top section a background image using CSS.

The next section I worked on was the "About" section:



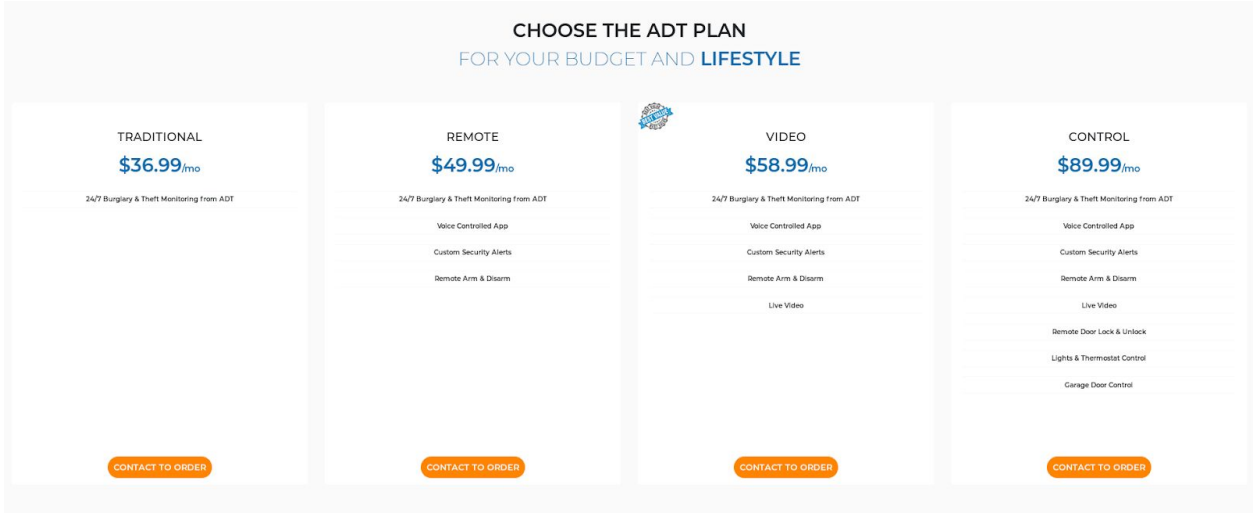
Again I used Bootstrap, and wrapped the content in the "container-fluid" class, followed by a "row." Within the row, I created two columns: one with a grid width of eight (the "ABOUT US..." text) and the other has a grid width of four (the image).

Then I worked on the "Smart Technology" (A.K.A. "Services") section:



This whole section is wrapped in the “container-fluid” class as well, followed by a “row.” Within the row there is just one column, which spans the whole grid (width of 12). Like the top section, the background image was included via CSS.

Next, I worked on the “Packages” section:



First, I wrapped everything up in the container-fluid class. Then I included the section headings (“CHOOSE THE ADT PLAN…). Next, right underneath the headings I have a row, which is divided into four columns of equal grid width (width of 3). Then in each column, I included a Bootstrap card for each of the packages (i.e. Traditional, Remote, Video, and Control), and then styled accordingly.

Then after the mid-sections were taken care of, I started on the footer:

The footer contains three rows. The first row is split into four columns of equal grid width, and contains the contact info (address, phone number, email address, and social media links). The next row is also split into four columns of equal grid width, and contains the JAL logo and disclaimers, menu links, news, and a newsletter sign-up form. The last row contains one column, which spans the whole grid, and just contains additional disclaimers. Again, like the other sections, HTML was added first, followed by custom CSS.

Responsiveness

Bootstrap helps tremendously with creating a responsive website, but simply adding some Bootstrap classes isn't enough. So, I went through section by section again, adding additional custom CSS that will make the site more responsive, such as media queries.

```
@media only screen and (max-width: 767px){
    .contact-info{
      display: none;
    }
    .top-section-h1{
        font-size: 2em;
    }
    .top-section-h2{
        font-size: 3.5em;
    }
```

```
@media (max-width:1556px){
    .top-heading .card{
        margin-bottom: 0;
        margin-right: 0;
    }
}
@media (max-width:991px){
    .top-heading .card{
        margin-bottom: 40px;
    }
    .mid-section-3 .card{
        height: 600px;
        width: 80%;
        margin-left: auto;
        margin-right: auto;
        margin-bottom: 20px;
    }
```

```
@media (max-width:1500px) and (min-width:992px){
    .mid-section-1 p{
        margin-left: 10px;
        margin-right: 10px;
    }
    .mid-section-1-left{
        margin-top: 10px;
    }
    .mid-section-1-heading{
        margin-bottom: 5px;
    }
    #mid-section-1-h2{
        margin-bottom: 10px;
    }
}
@media (max-width:1296px) and (min-width:992px){
    .mid-section-1 p{
        font-size: 0.8em;
        margin-left: 0;
        margin-right: 0;
    }
    .mid-section-1-left{
        margin-top: 5px;
    }
```

After adding the media queries, I moved on to adding some JavaScript.

JavaScript

First thing I did was create new div tags on the top of each section that will serve as a destination for the menu links. For example the “about” div below:

```
<div id="about"></div>
<div class="container-fluid mid-section-1">
```

This div with id=”about” will serve as destination for the About menu item.

```
<li class="nav-item">
  <a id="abt" class="nav-link abt" href="#about">About</a>
</li>
```

Then, finally, I added the JavaScript for each menu item:

```
$(".abt").click(function(e) {

    $('html, body').animate({
        scrollTop: $($.attr(this, 'href')).offset().top
    }, 800);
});
$(".svcs").click(function(e) {

    $('html, body').animate({
        scrollTop: $($.attr(this, 'href')).offset().top
    }, 800);
    });

$(".pkgs").click(function(e) {

    $('html, body').animate({
        scrollTop: $($.attr(this, 'href')).offset().top
    }, 800);
});
```

What this code does is gracefully move the view to the part of the page specified in the href of the menu item. By gracefully I mean you see it moving. Without the JavaScript, it would just jump to that part of the page, without any noticeable motion.

The same was done for the “Contact” link, except that after it’s clicked, the header of the contact form pulsates to place emphasis on it.

```
$(".contact").click(function(e) {

    $('html, body').animate({
        scrollTop: $($.attr(this, 'href')).offset().top
    }, 800);
    $(".top-heading .card-header").effect( "pulsate", {times:5}, 3000 );
});
```

After the menu items were taken care of, I included a jQuery sticky navbar plugin, which will make the navbar follow the user while he/she scrolls the page. I included the JS for this in the js directory, and then changed some HTML and initiated it with the following code:

```
<script type="text/javascript">
  $(document).ready(function() {
    $('.stickyMenu').smoothMenu({
      stickyMenu: true,
      slidingLine: true
    });
  });
</script>
```

I then turned my attention to all the buttons on the site. I gave them functionality by having them link to the appropriate places on the page (e.g. the “Contact to Order” button in the Packages section, which links to the contact form).

Miscellaneous info

Throughout developing the site, and again after everything was done, I changed any HTML, CSS, or JS that I felt would make the site better. For example, changing the font-size, font-weight, colors, the opacity of the navbar, adding HTML form validation, etc.

The font I used is from Google Fonts and it’s called Montserrat, which was included in the HTML with this link tag:

```
<link
href="https://fonts.googleapis.com/css?family=Montserrat:thin,regular,medium,bold,100,200,300,400,500,600"rel="stylesheet">
```

While committing to the public repository, I always pushed to the dev branch, with the following commands:

```
git add –all
git commit -m “example comment”
git push origin dev
```

The hope would then be that it’ll pass the Jenkins test, and get merged to staging. Then while in staging, it should pass its test and get merged to master.

Logging/Monitoring - Christine Liu & Philip Cunningham

Purpose

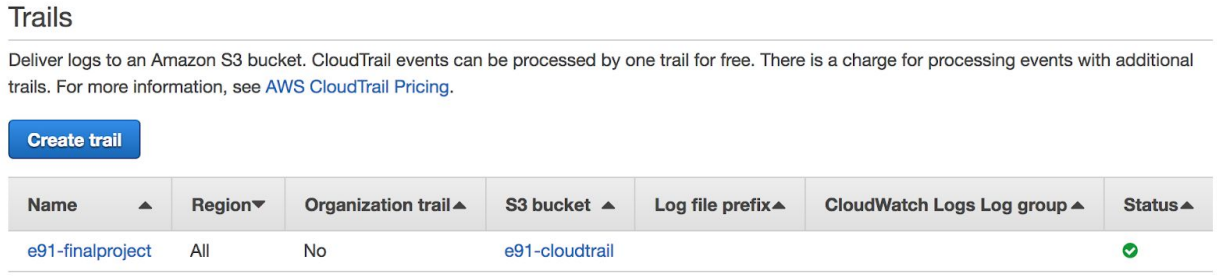
For production environments, security is a top priority. The team decided to set up logging and monitoring as an above and beyond, implementing DevSecOps. Logs are important for security to see who is accessing the system and required for audit purposes. It gives us the ability to track what is going on in all the instances and gives the team the ability to use logs to troubleshoot while retaining transparency of what is happening in the infrastructure.

Logging/Monitoring in AWS

To implement logging, we wanted to advantage of some of the great AWS resources that we learned about in class and decided to use cloudtrail and a S3 bucket due to its low cost and low overhead. Cloudtrail was set up to monitor all the AWS instances and port logs into an S3 bucket. These logs show who logs in, what packages are installed, what packages are running, and general system data.

Steps to create a cloudtrail in AWS
AWS -> Trails -> Create Trail -> Trail name (e91-finalproject) -> Create a new S3 bucket (e91-cloudtrail)

Resulting image of Cloudtrail log created



Logging/Monitoring in GCP

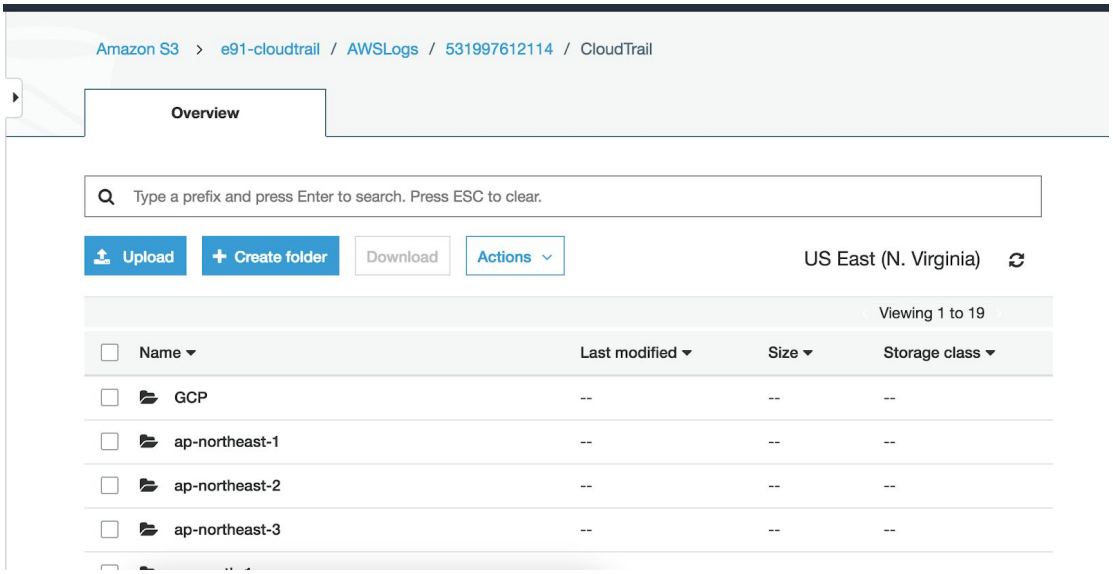
Then, to integrate the GCP infrastructure, research was done to figure out how to integrate the GCP instances into the S3 bucket. We used scripts to set up the prod and ansible instance in GCP to integrate with cloudtrail and port all the logs into the same S3 bucket.

Steps to create logging in GCP
AWS-integration.sh
The AWS integration script will install awscli so the to connect to a s3 bucket. First it will download and install packages unzip and python that will be used for installation of AWSCLI-bundle. Then the script will curl the installation, unzip, install and move the binaries to a specific folder for the second script to use.

GCP-ubuntu-logging.sh - sets up our ansible server to talk to S3 bucket
GCP-CentOs-loggingsetup.sh - sets up our ansible server to talk to S3 bucket
The GCP-ubuntu and centos logging scripts allow GCP instances can push their logs to their respective S3 buckets. The script also stores env variables for AWS configuration, AWS access key, AWS secret key, sets the endpoint and creates a cron job to log system activity every 5 minutes.

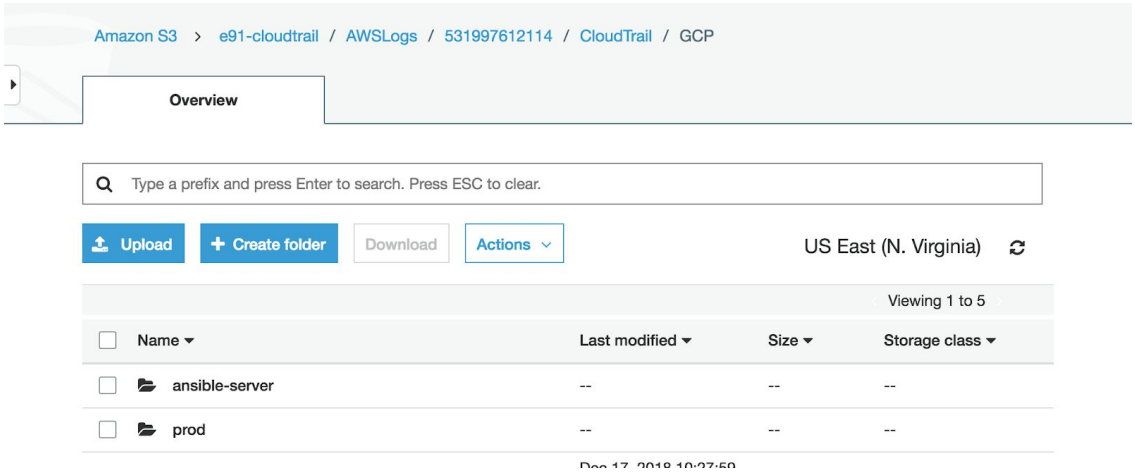
Checking Logs

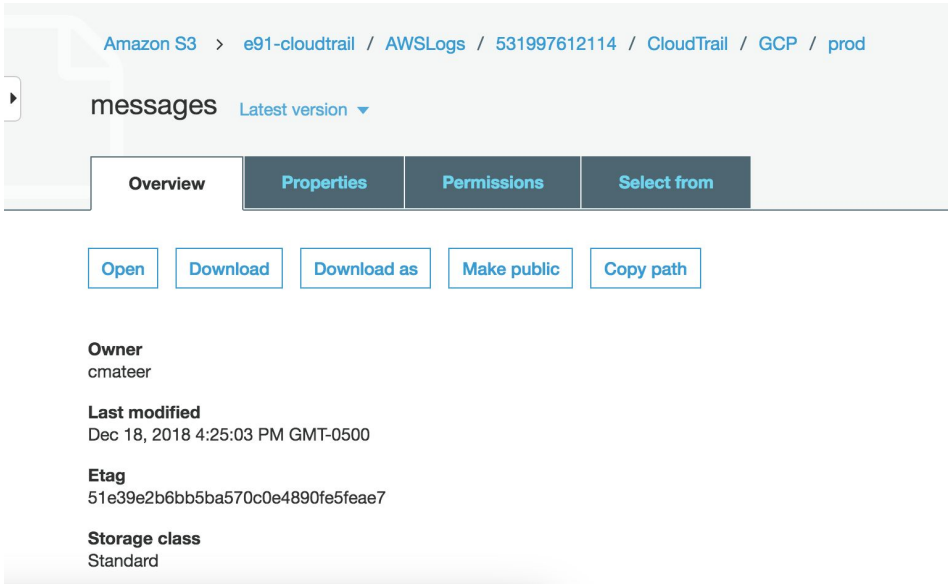
Steps:
Amazon S3 -> Buckets -> select e91-cloudtrail -> select AWSLogs -> select 531997612114 -> select CloudTrail
Image below shows S3 with all the logs. GCP infrastructure logs are in GCP folder. AWS EC2 instances are within folder named after availability zone (scroll list)



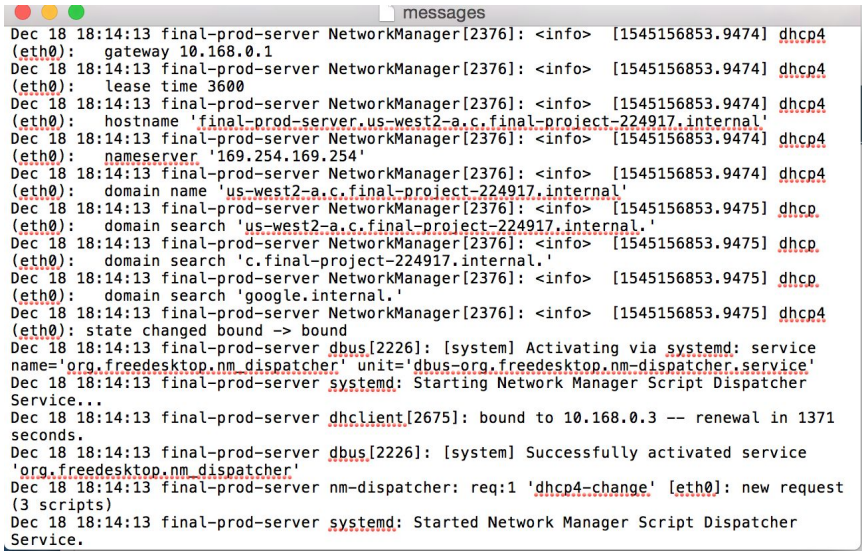
Example of checking logs

For example, if we wanted to check the GCP logs of the prod server, we would select the GCP folder -> select prod -> select messages -> select download





Example of a downloaded log



FOLDER DESCRIPTIONS

We have two Git repositories, public github and enterprise private code.harvard.edu:

Private Github

code.harvard.edu (enterprise github) (https://code.harvard.edu/cwm850/csci-91-final-phase1-private) with one branch to accommodate:

a. Branches
    i. Master

b. Folders
    i. ansible folder to hold Ansible configuration files
    ii. cloud-formation folder to hold CloudFormation files that will set up AWS infrastructure
    iii. GCP-templates folder that holds files to set up GCP infrastructure
    iv. docker folder that holds scripts to install docker on VMs
    v. jenkins folder to hold jenkins pipeline files
    vi. Security-logging folder that holds logging and monitoring setup for our AWS and GCP infrastructure

Public Github

Public Github (https://github.com/cwmat/csci-91-final-phase1-public) to perform version control for web site files and images:

a. Branches
    i. Dev
    ii. Staging

iii. Master

b. Folders

i. Index.html (file) - full scale web page
ii. css - folder with css files
iii. errors - folder that contains HTML files for server errors
iv. img - image folder that holds all of the images of the website, including the favicon
v. js - folder that contains js files for the JavaScript features
vi. tests - folder with test cases
vii. Prod_Server_Files - folder for the files used for prod
viii. docker - folder with docker files that developers can use to set up their dev and stage containers (does not contain sensitive information)

FINAL PROJECT FUNCTIONALITY

This shows an example of the end to end testing that we did together.

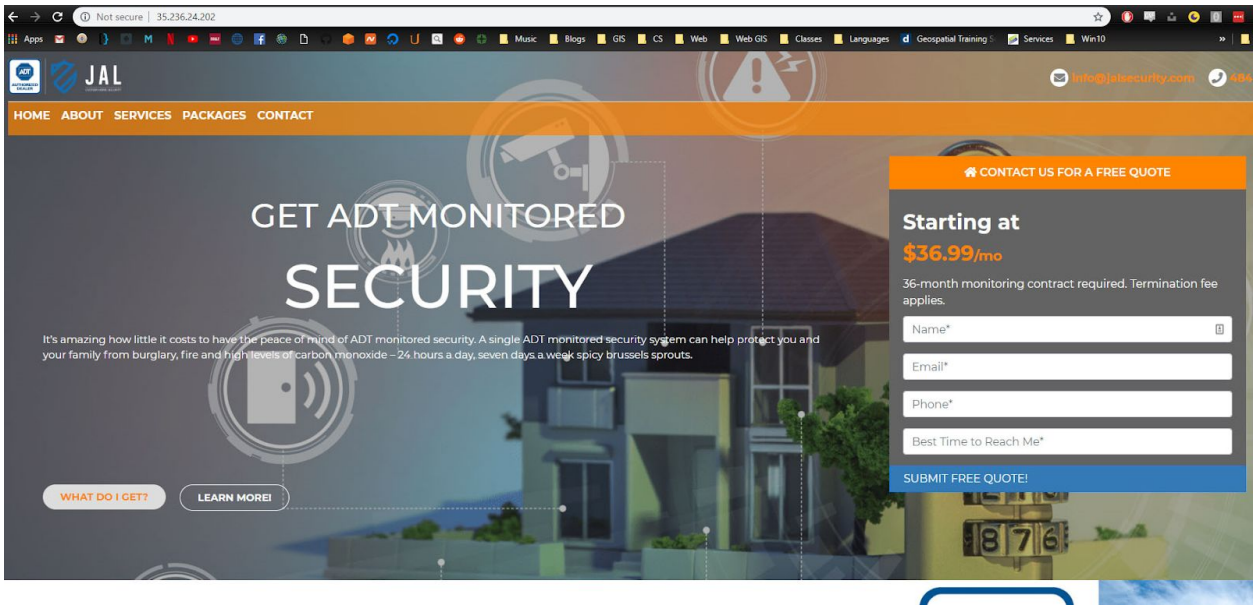
INITIAL WEBSITE

Link to the website

Link to Public Production Website: http://35.236.24.202/

Image of the initial website

Production



OUTLINE OF CHANGES

Changes made to the source code and pushed to the dev branch:
Removed text about spicy brussels sprouts and changed the navbar to green.

Changing Files

main.css

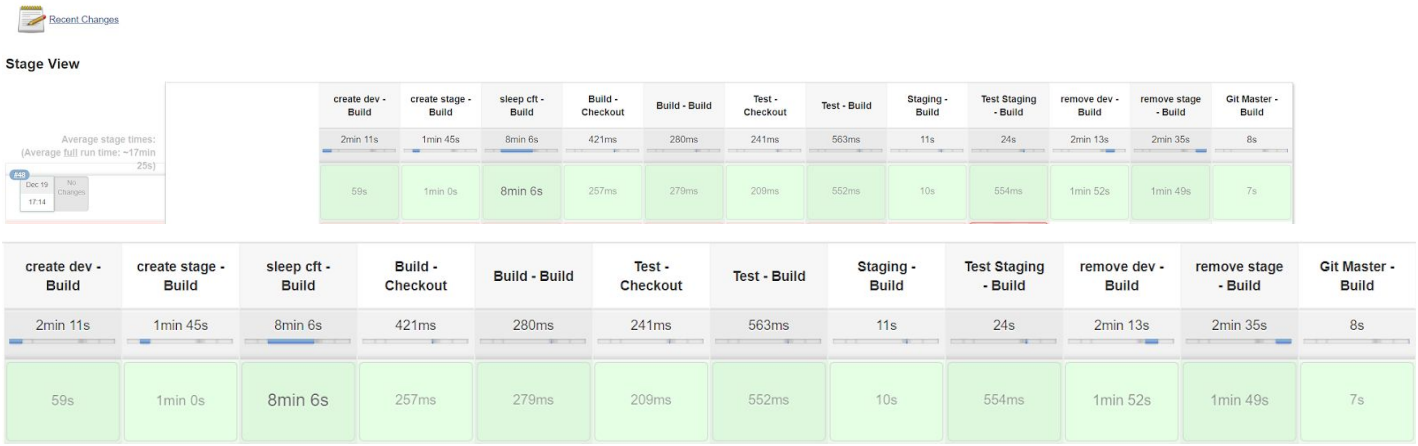
```
93      padding: 0.2em 0;
94  }
95
96  /* ================================================================================
97     Author's custom styles
98     ================================================================================ */
99  * {
100     font-family: 'Montserrat', sans-serif;
101 }
102
103 img {
104     max-width: 100%;
105 }
106
107 .navbar {
108     /* background-color: rgba(255, 132, 0, 0.7) !important; */
109     background-color: rgba(0, 255, 55, 0.7) !important;
110     border-top: 1px solid lightgray !important;
111     padding-left: 0;
112     padding-top: 0;
113     font-weight: bold;
114     margin-bottom: 30px;
115     z-index: 9999;
116     margin-left: 0 !important;
117     margin-right: 0 !important;
118 }
```

Committing changes to the public repo's dev branch:

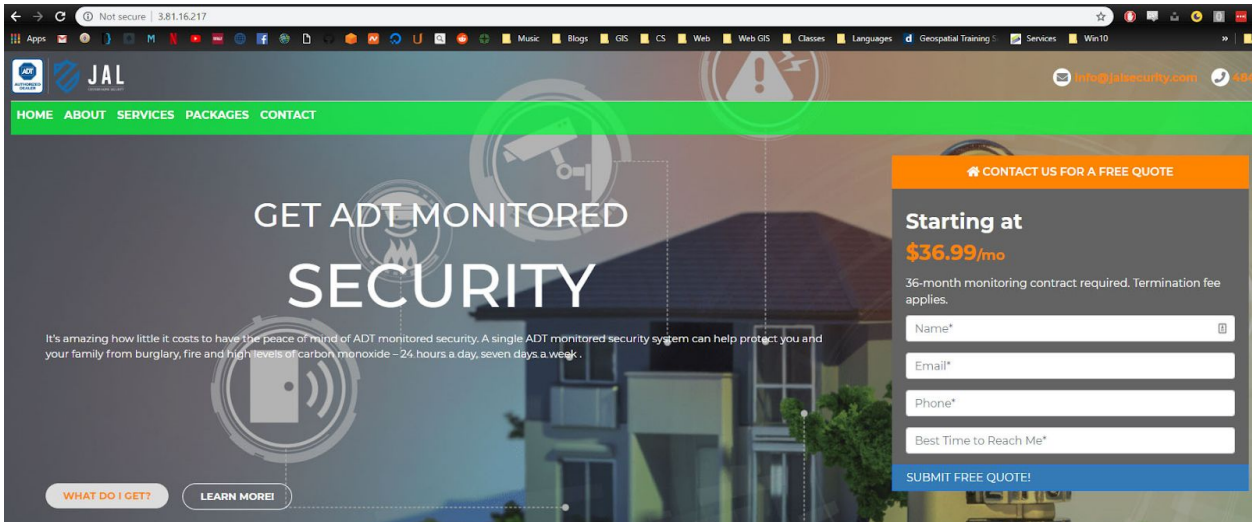
```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

PS C:\Users\chazm\OneDrive\Documents\Projects\CSCI91_Final\temp new\csci-91-final-phase1-public> git add .
PS C:\Users\chazm\OneDrive\Documents\Projects\CSCI91_Final\temp new\csci-91-final-phase1-public> git commit -m"Change nav bar color for test."
[dev 225cf1d] Change nav bar color for test.
 2 files changed, 312 insertions(+), 165 deletions(-)
PS C:\Users\chazm\OneDrive\Documents\Projects\CSCI91_Final\temp new\csci-91-final-phase1-public> git push origin dev
Enter passphrase for key '/c/Users/chazm/.ssh/id_rsa':
Counting objects: 5, done.
Delta compression using up to 12 threads.
Compressing objects: 100% (5/5), done.
Writing objects: 100% (5/5), 2.19 KiB | 561.00 KiB/s, done.
Total 5 (delta 3), reused 0 (delta 0)
remote: Resolving deltas: 100% (3/3), completed with 3 local objects.
To github.com:cwmat/csci-91-final-phase1-public.git
   0052a76..225cf1d  dev -> dev
PS C:\Users\chazm\OneDrive\Documents\Projects\CSCI91_Final\temp new\csci-91-final-phase1-public>
```

Jenkins CI/CD process

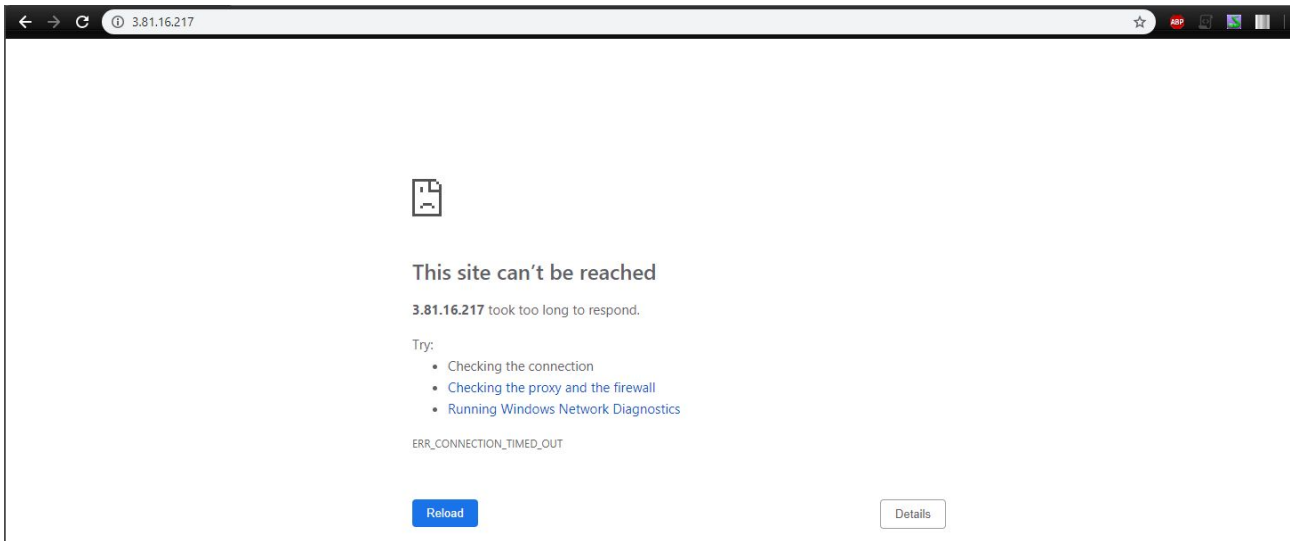
Jenkins Pipeline tests and builds in pushing code to GitHub staging and master branch after tests have passed. Jenkins also builds and tears down dev and stage environments when changes pass tests.



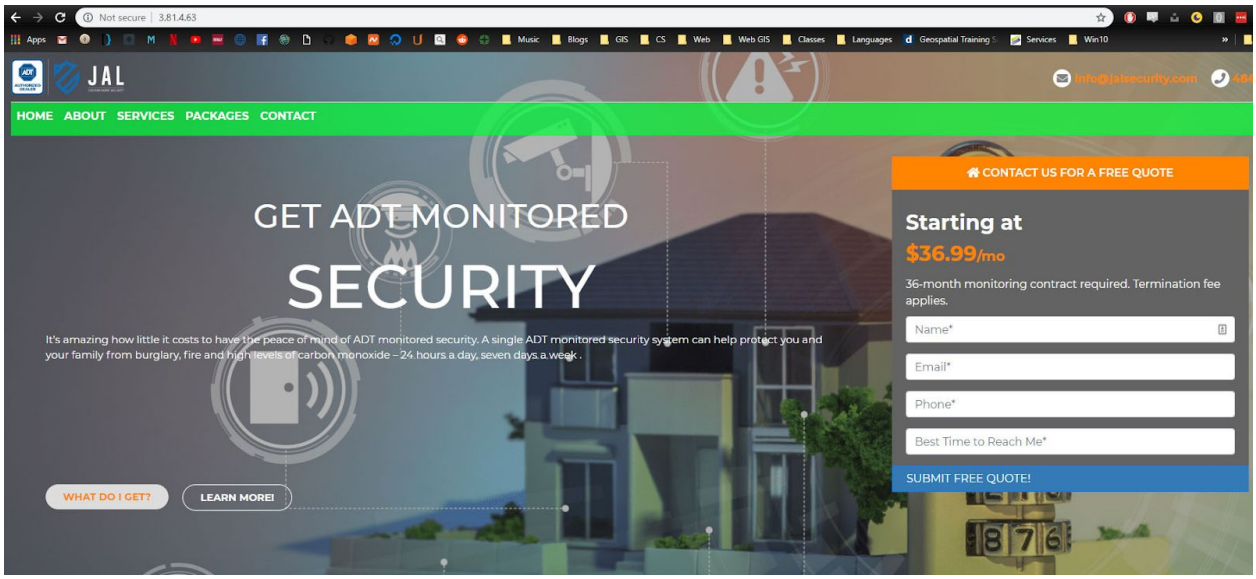
Jenkins created dev environment site:



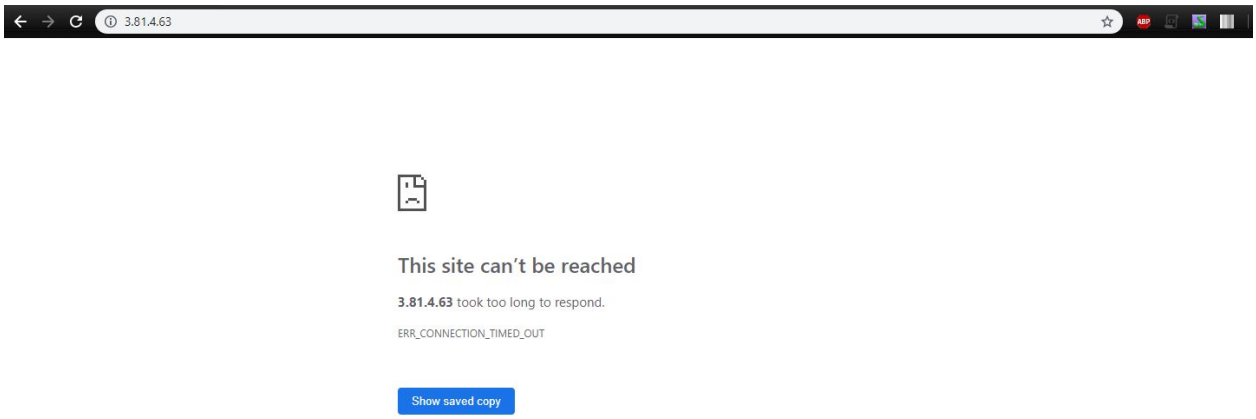
Proof that Jenkins destroyed testing environment after tests passed:



Jenkins created staging site:



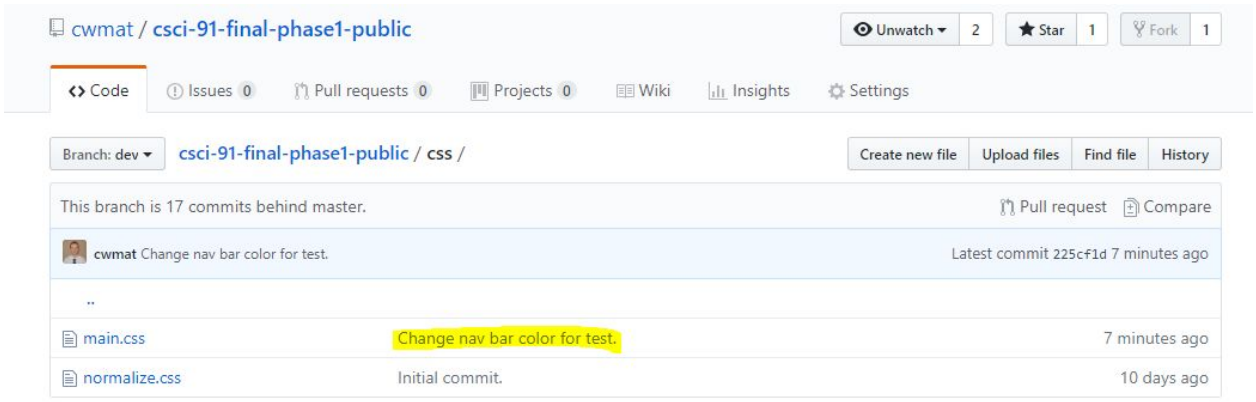
Proof that Jenkins destroyed staging environment after tests passed:

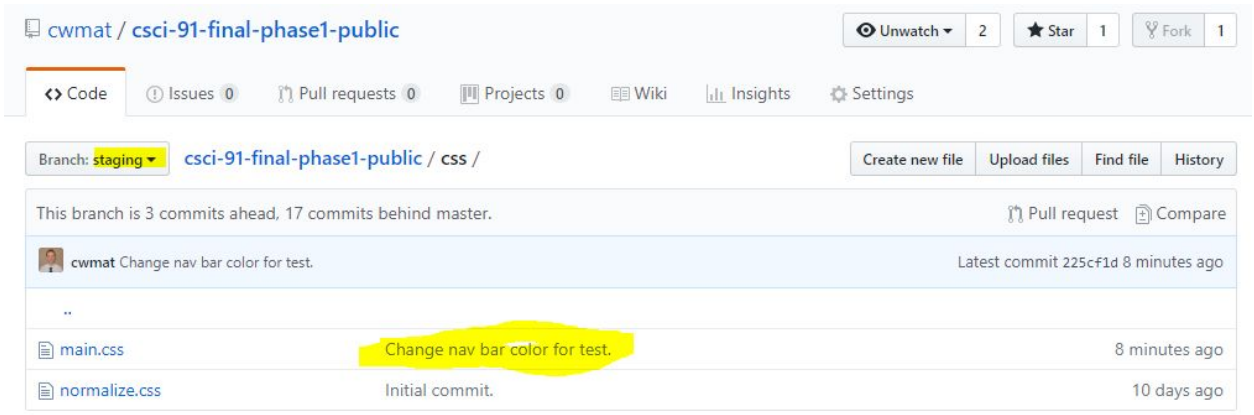


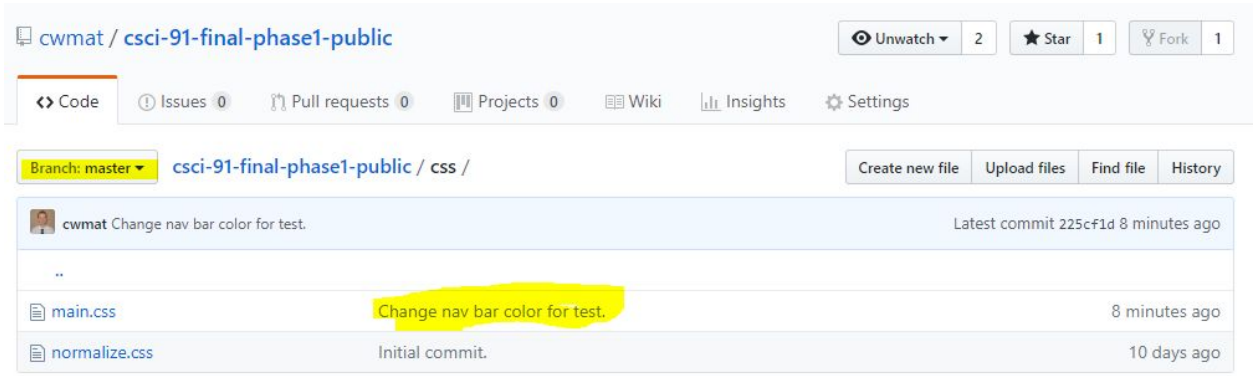
RESULTS

Change results reflected in Github

Images of GitHub repo branches. Dev was pushed to directly. Staging and master were updated by Jenkins after passing tests. Commits by Jenkins appear as cwmat as Jenkins is using cwmat's (Charles Mateer) access key.



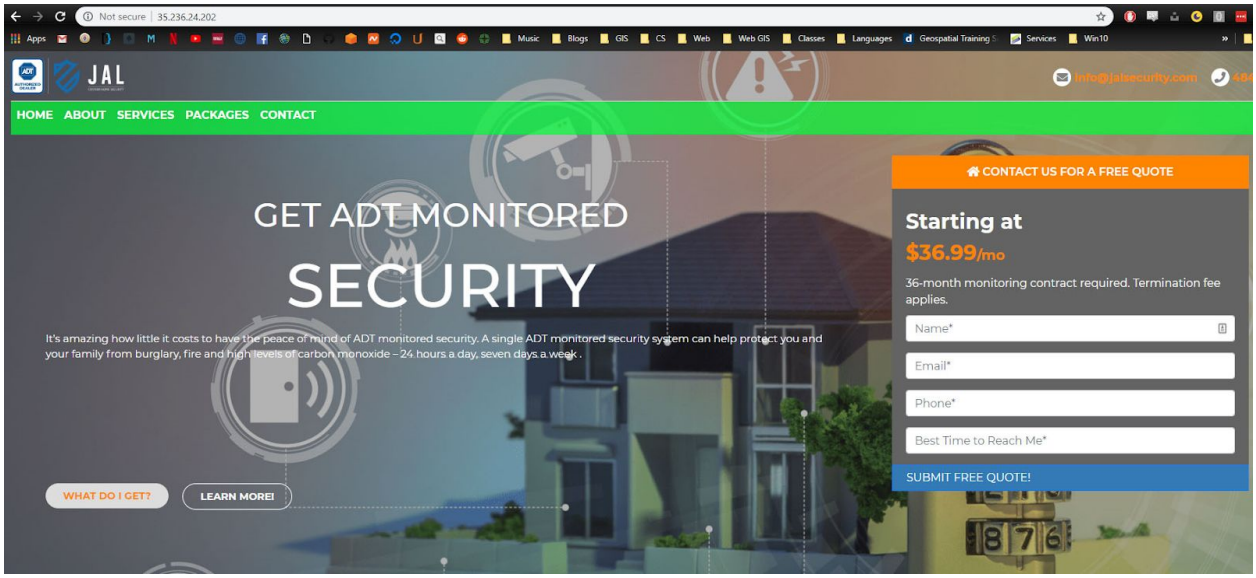




Change results reflected on site

Images of production website after Jenkins pipeline has finished:

Prod:



DOCUMENTATION FOR USER

This section provides documentation for a developer to start contributing to our open source project.

E91 CLOUD DEVOPS FINAL PROJECT PHASE 1

Welcome to Group 1's open source project called Phase 1! The link to the production site can be found here: http://35.236.24.202/. This site is a full scale site that sells ADT security.

MOTIVATION

This project is a final group project for the E91 DevOps class. Utilizing basic and modern Cloud DevOps techniques that we learned from class, we were challenged to develop an open source project that packages our requirements in a single container on public github that accounts for no funding and special treatment of sensitive information. We will use CI/CD techniques to build a pipeline for a website that needs a seamless, end-to-end continuous delivery and deployment workflow. A continuous integration pipeline will support test, configuration, and deployment of a website and a continuous delivery pipeline will support automatic building, testing, and release of code changes to production environment.

HOW IT WORKS

When a user commits to dev, this will trigger our CI/CD pipeline to kickoff. Jenkins will build a dev container in a EC2 instance and run tests. If the tests pass, Jenkins will set up stage EC2 and test, tearing down the dev container. If the tests pass, Jenkins will tear down stage and the changes will reflect in prod and the public github. The building of dev and stage VMs and containers only when a change is initiated ensures that we do not spend money on a development and staging environment.

CONTRIBUTING GUIDELINES

We invite you to contribute! If you wish to contribute to this open source project, follow these steps:

1. Clone the repo if you are a contributor. Else, fork master https://github.com/cwmat/csci-91-final-phase1-public .
2. Create a feature branch and make edits .
3. Commit edits to your feature branch and create pull request to dev branch
4. Include these reviewers

   a. Usernamelias
   b. cwmat
   c. diasyzheng2012
   d. cml2197
   e. philc171
   f. scakaeze
5. Once request is approved and merged by reviewer, it will trigger dev branch and it will go through the jenkins CI/CD process
6. If success, you will be able to see the changes you made in the github repos and on the prod site

AUTHORS

Thanks to all the people who created this project

- Christine Liu
- Charles Mateer
- Jingjing Zheng
- Stephen Akaeze
- Philip Cunningham
- Elias Falconi