For my final project, I chose to do Project number 1 for the fictional ABC Technologies company.

Resources can be located at:

https://github.com/clong1969/PGPDevOps

https://hub.docker.com/repository/docker/clong1969/abctech/general

1. The Requirements:

ABC technologies is a leading online retail store. ABC has recently acquired a large retails offline business store. The business store has large number of stores across the globe but is following conventional pattern of development and deployment. As a result, it has landed to great loss and are facing below challenges.

- low available
- low scalable
- low Performance
- Hard built and maintained.
- Developed and deployed.

is time consuming ABC will acquire the data from all these storage systems and plan to use it for analytics and prediction of the firm's growth and sales prospect. In the first phase ABC has to create the servlets to Add a product and Display product details. Add servlet dependencies required to compile the servlets. Create an HTML page which will be used to add a product. Team is using git to keep all the source code. ABC has decided to use DevOps model and Once source code is available in GitHub, we need to integrate it with Jenkins and provide continuous build generation for continuous Delivery, integrate with Ansible and Kubernetes for deployment. Use docker hub to pull and push images between ansible and Kubernetes.

2. Goals:
   •Implement CICD such that ABC Company to is able to be
   - Highly available
   - Highly scalable
   - Highly Performant
   - Easily built and maintained
   - Developed and deployed quickly
3. Tasks:
   We need to develop a CICD pipeline to automate the software development, testing, package, deploy reducing the time to market of app and ensuring good quality service is experienced by end users. In this project we need to
   - Push the code to out GitHub repository.
   - Create a continuous integration pipeline using Jenkins to compile, test and package the code present in GitHub
   - Write docker file to push the war file to tomcat server.

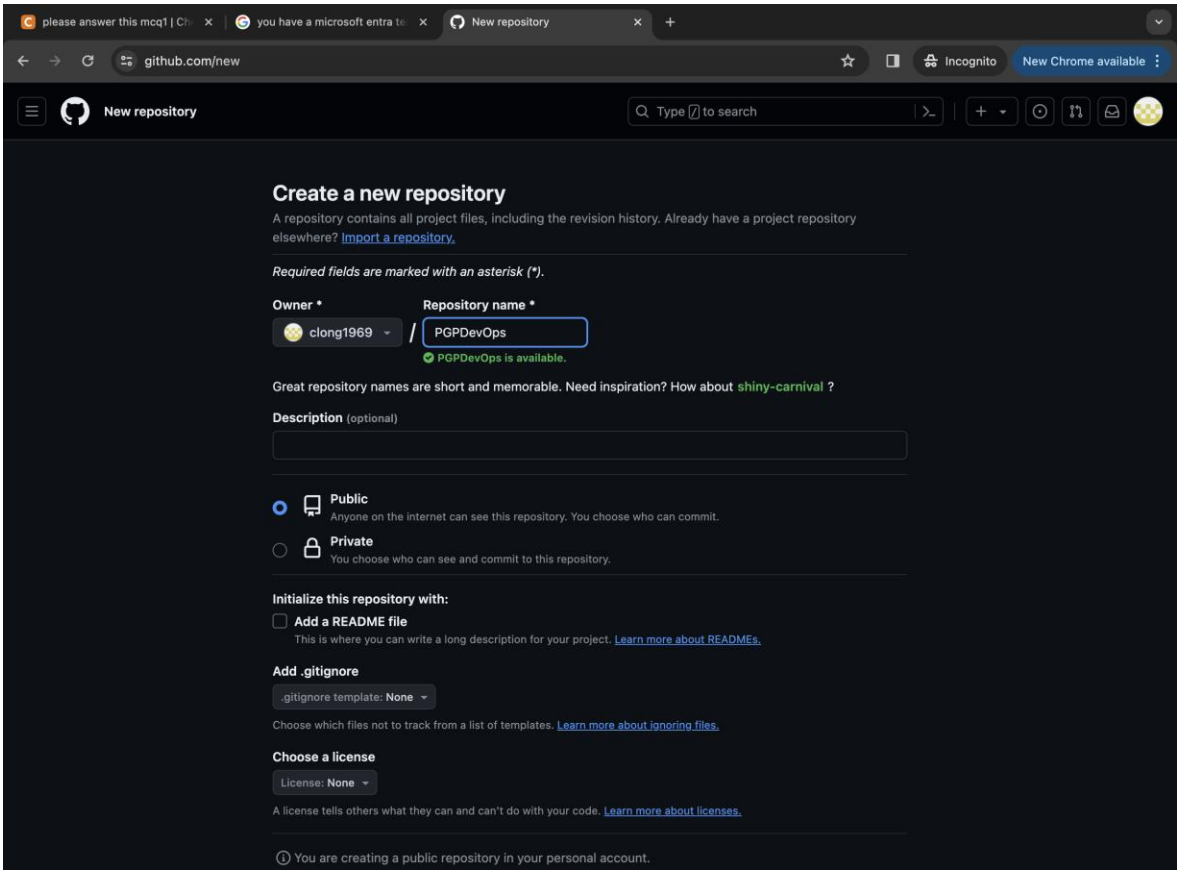
- Integrate docker with Ansible and write playbook5.Deploy artifacts to Kubernetes cluster.
- Monitor resources using Grafana.

Task1: Push supplied code to repository.

The code for this project was supplied by Edureka in their GitHub repository. In order to complete this overall project, I would need to get that code and place it in my repository. To do this, I completed the following steps.

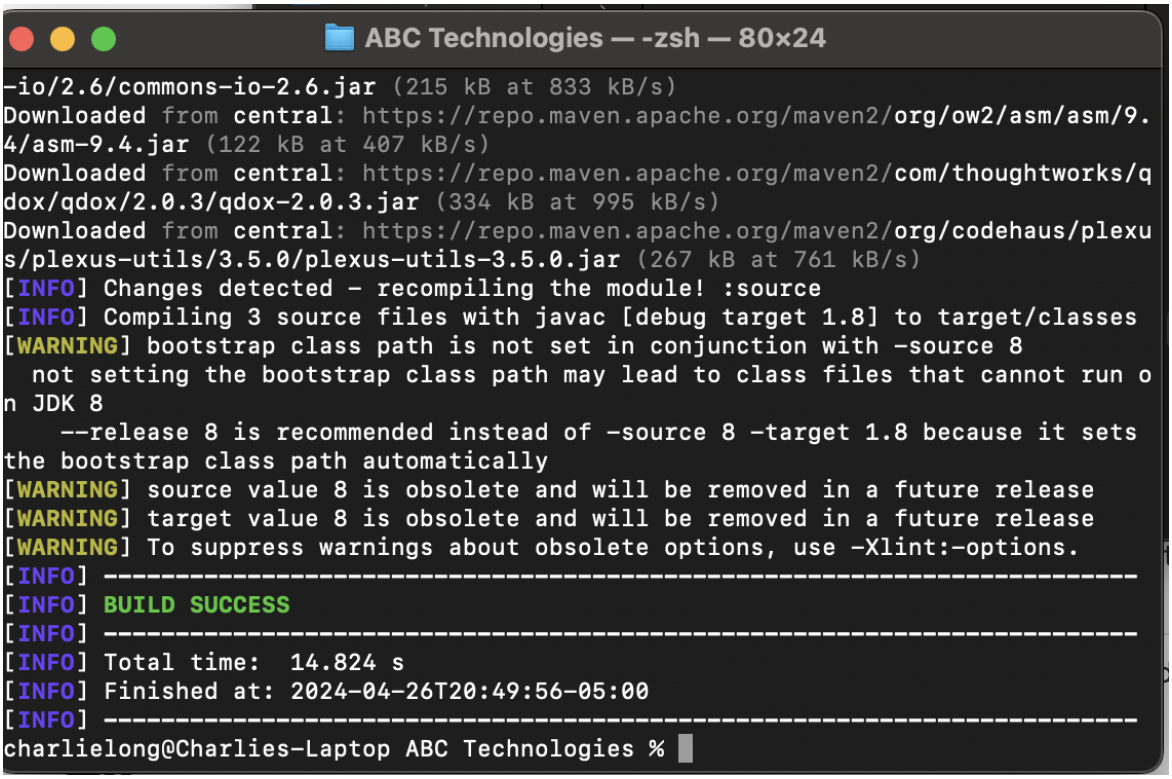
I created a new repository in my GitHub account.

https://github.com/clong1969/PGPDevOps.git

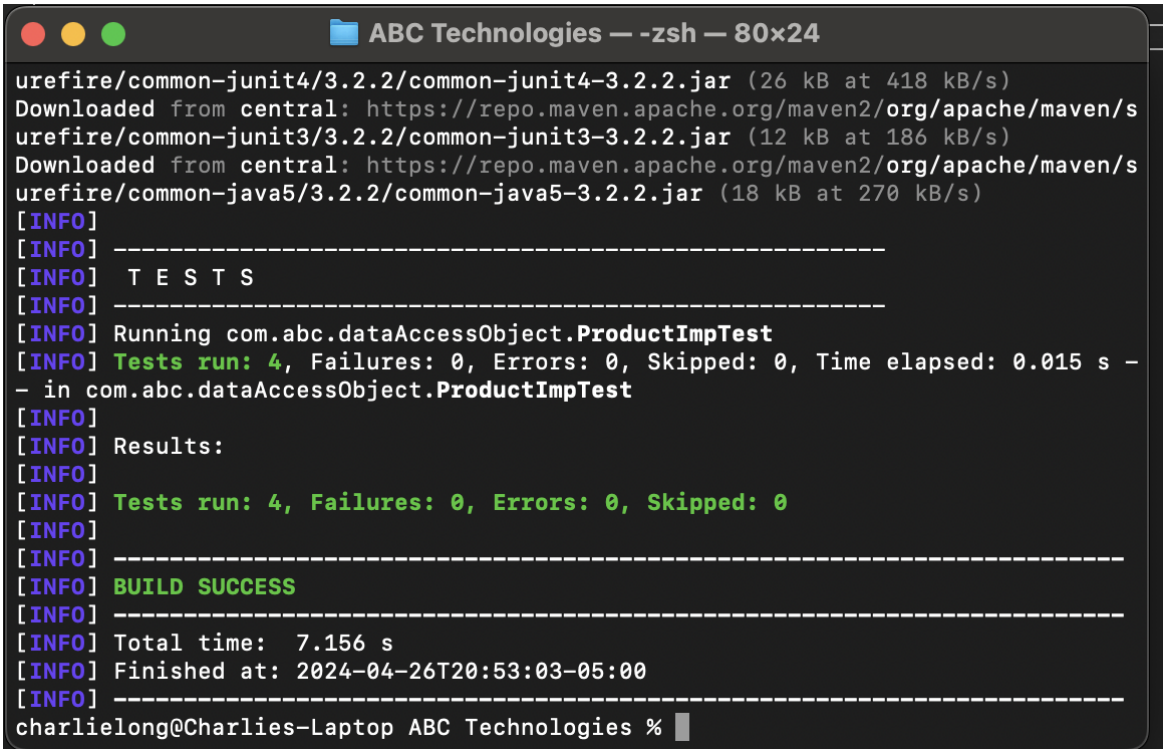


Copied the code to my local machine by issuing a git command. I have used git before, so it was already installed on my machine.

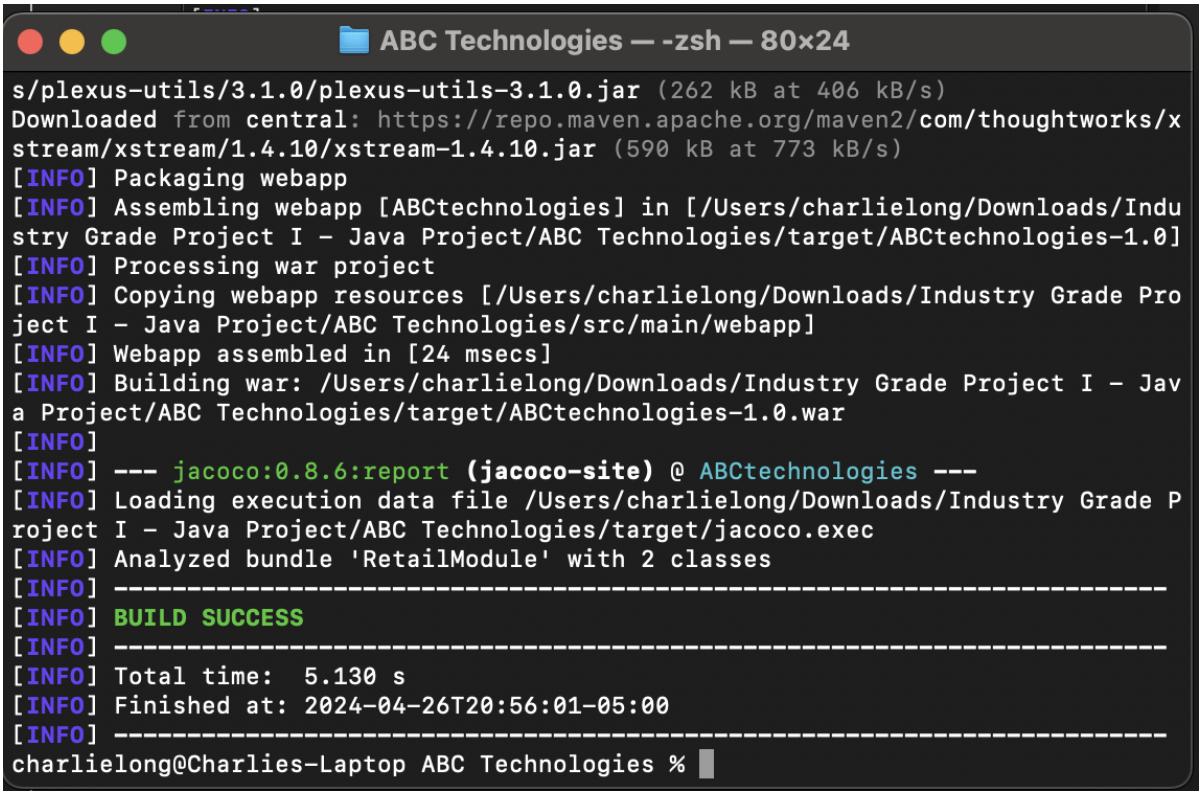
I then compiled the code. In order to do this, I installed Maven locally and compiled the source code that was downloaded in the previous step. This was accomplished by running: mvn compile.



Once the code compile was complete, I tested it to make sure it compiled correctly by running: mvn test.

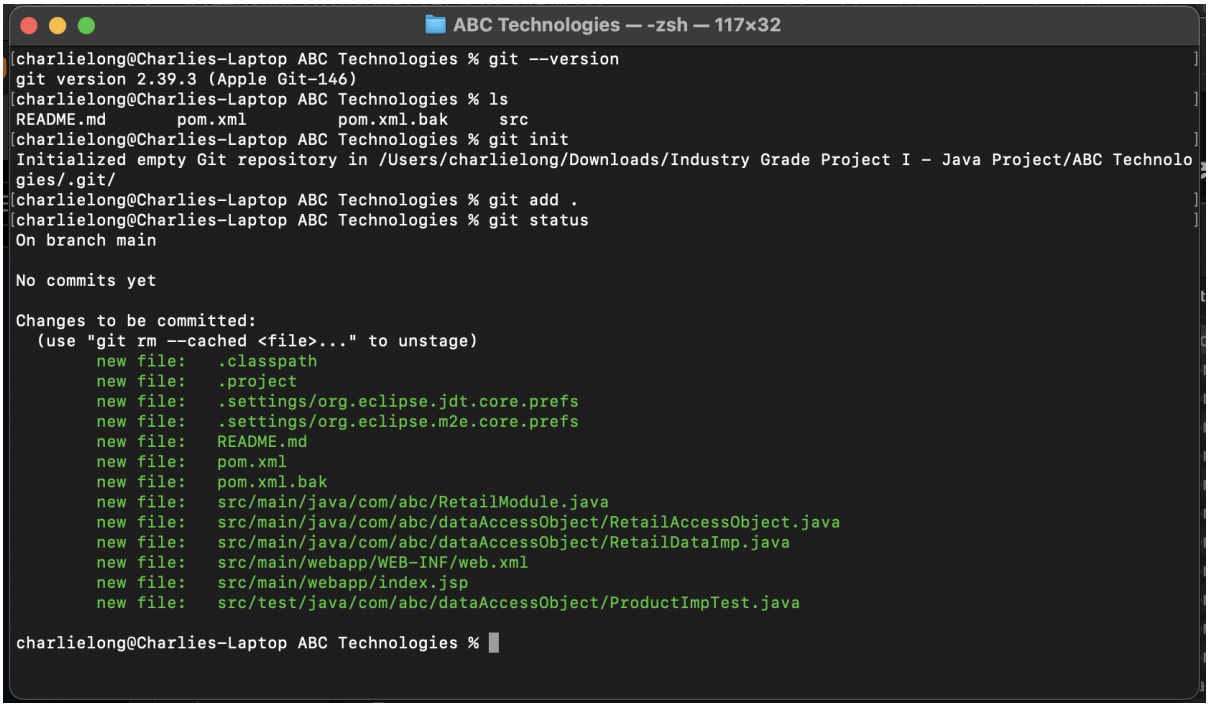


Now that I had a good compile, the code had to be packaged for distribution. This was done by running: mvn package.

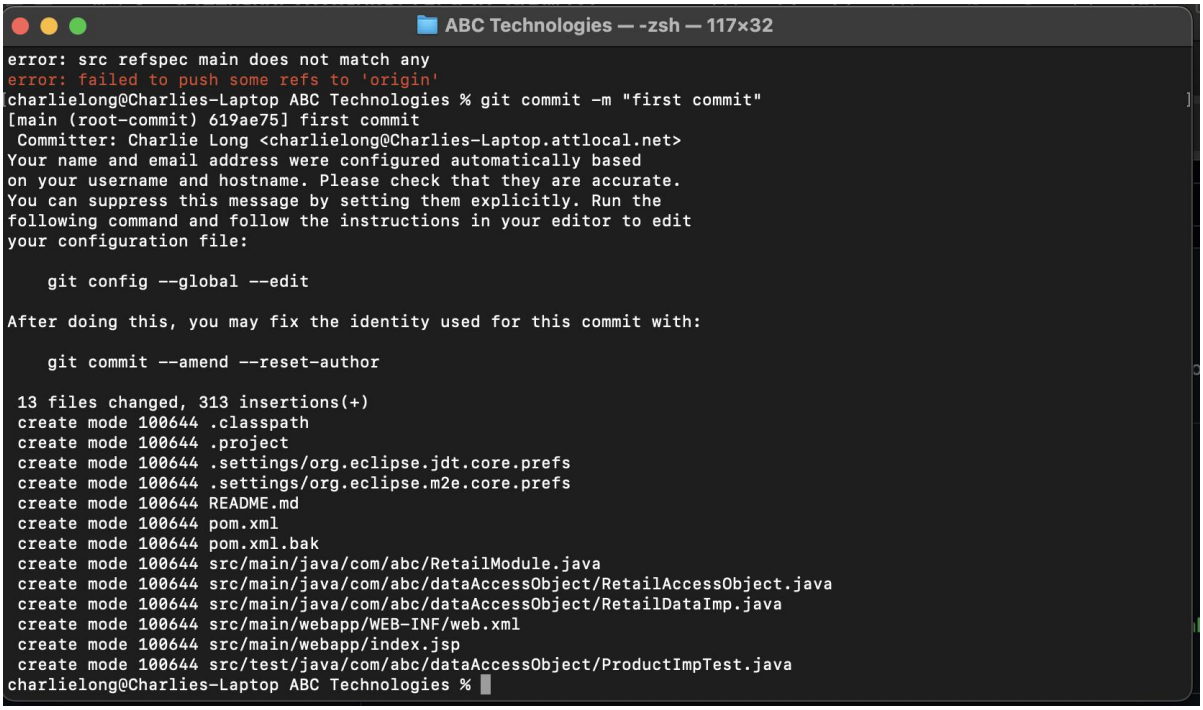


Once the code was compiled, I put all the resources into the same folder on my local machine and got it ready to upload to GitHub. This was done by running: git init.

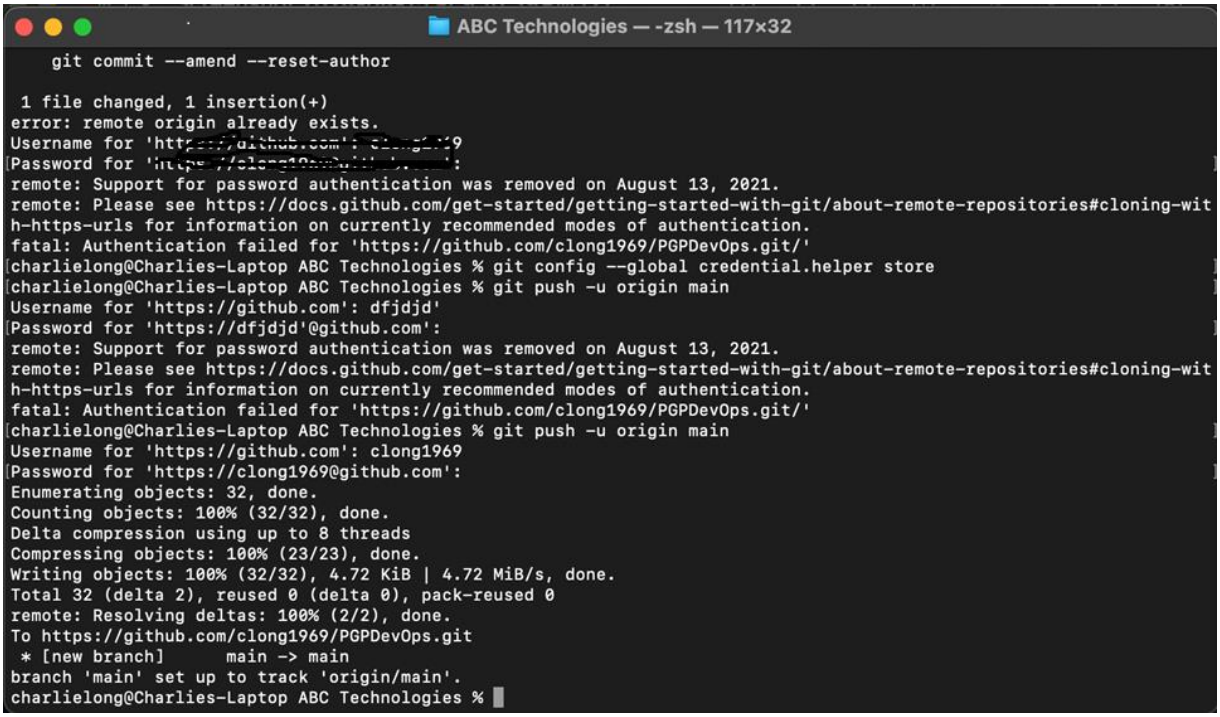
I ran a git add . .



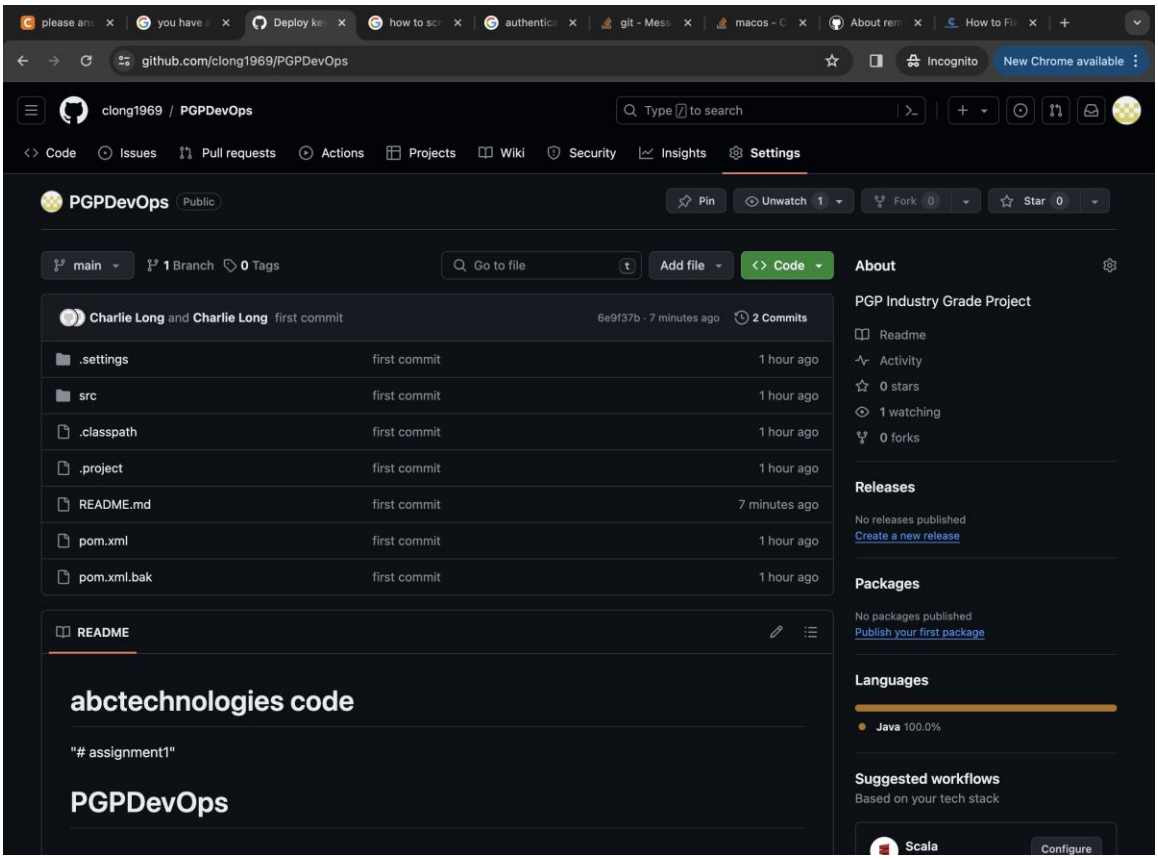
Once the files were added, I issued git commit -m “first commit” to prepare files for upload.



I then tried pushing the code to my GitHb repository. GitHub discontinued support for username and password authentication, so I had to install key on my local machine to provide authentication.

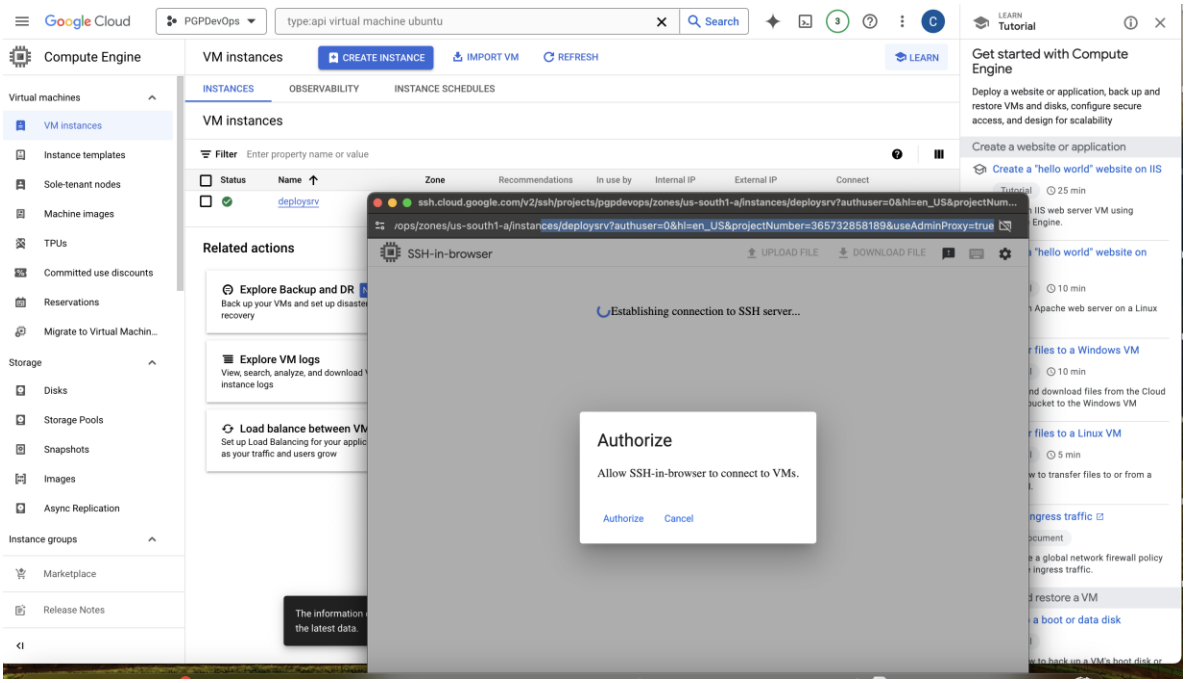


Here is my GitHub repo after the initial upload completed. At this time, all that was present were the original source files compiled.



Task 2: Create a continuous integration pipeline using Jenkins.

For this task, I created a single vm in Google Cloud and installed Jenkins, Java, Git, Docker and Maven.



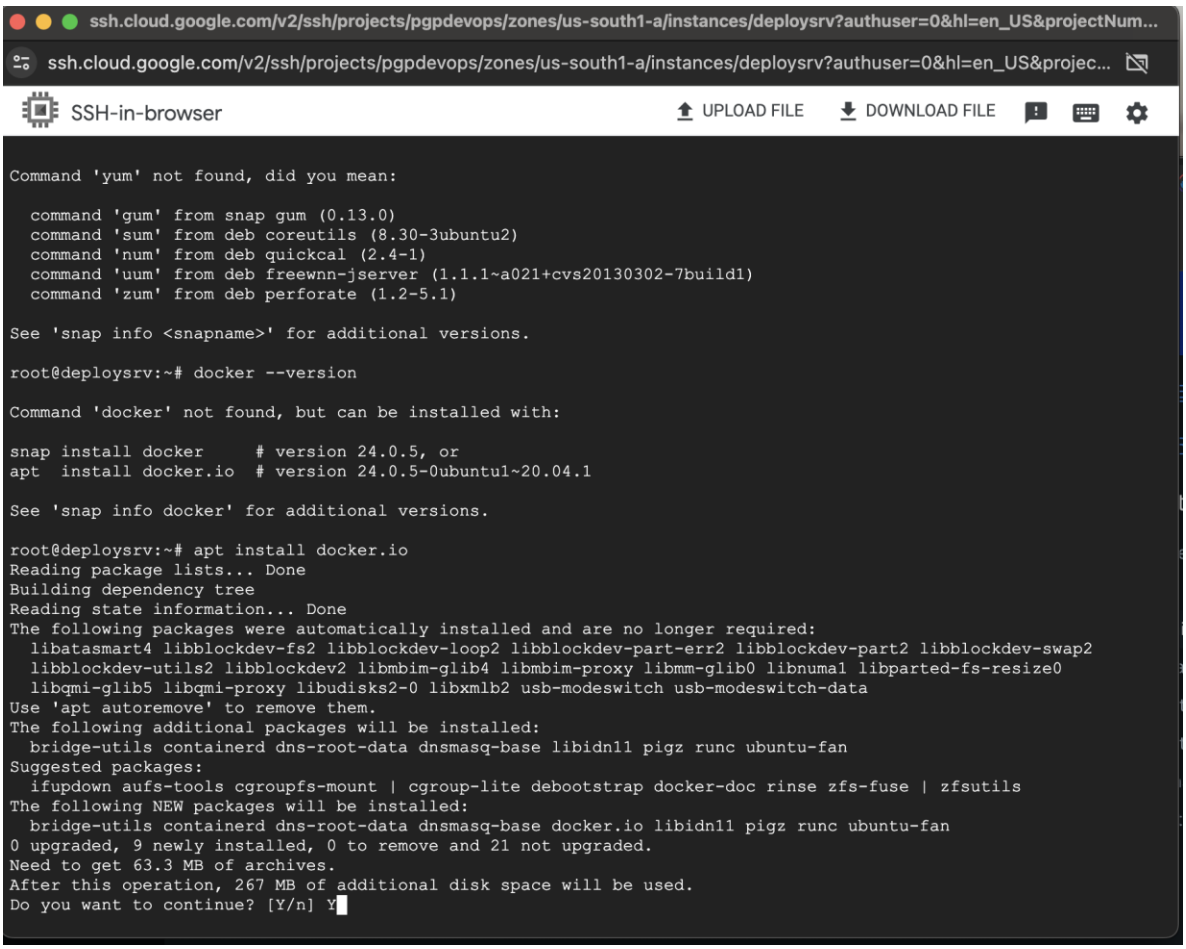
Made SSH connection to newly created VM to install required software.

In order to install Jenkins, I first had to install Java. To do this, I switched to the root user on my Ubuntu VM by typing: sudo su –

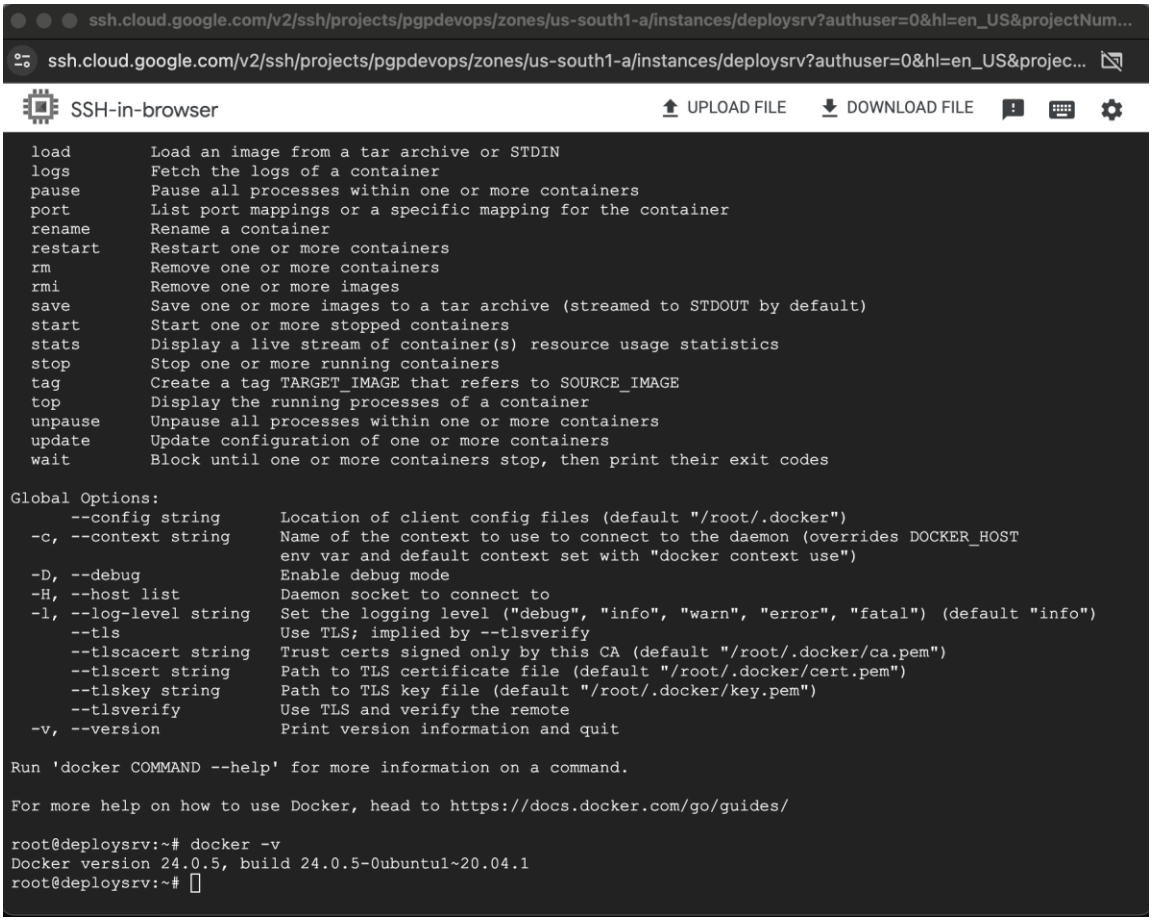
Verify git is installed: git –version. VM has version 2.25.1

```
Install Java: sudo apt install openjdk-17-jdk -y
```

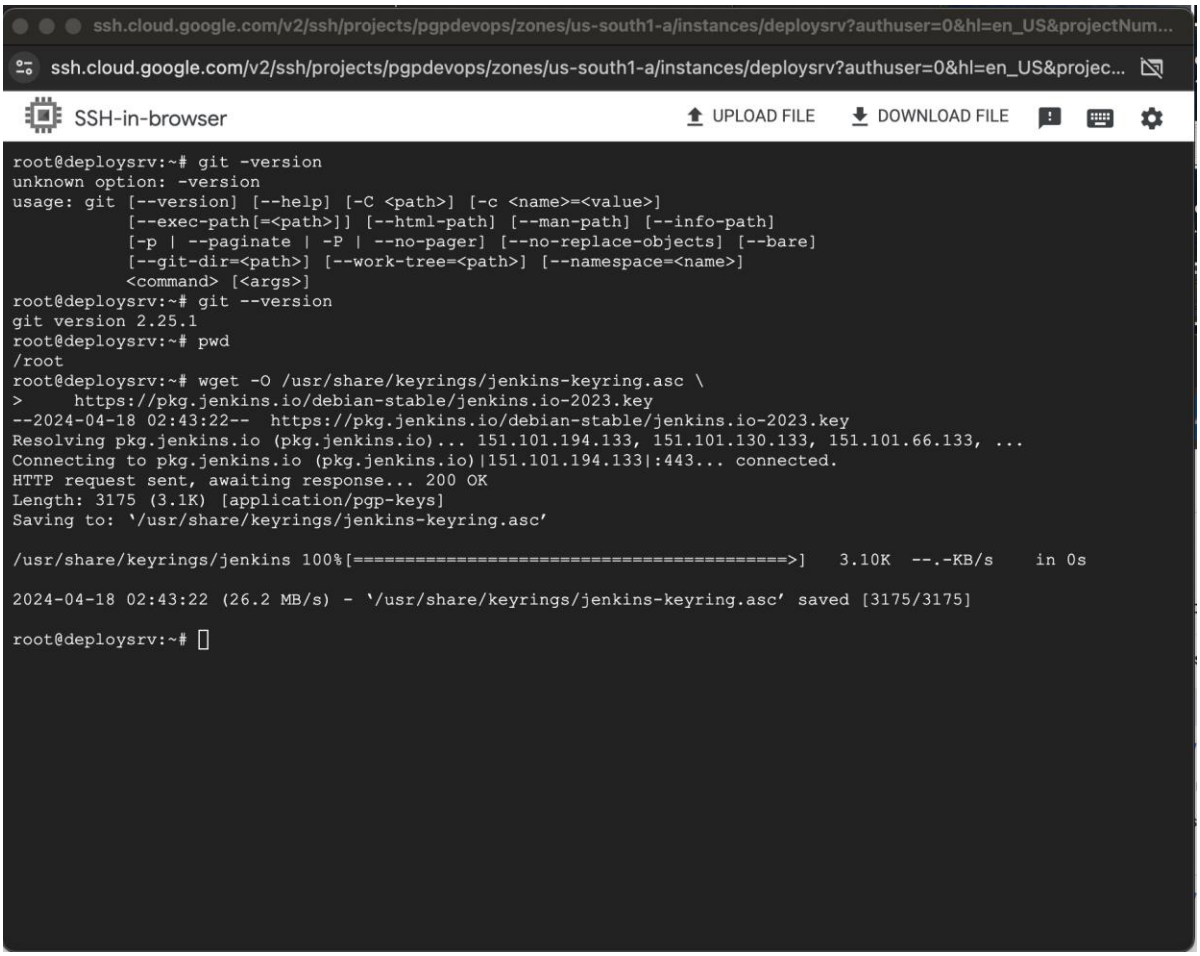
I then installed Docker on the same machine where Jenkins would be installed.



Verified that installation was successful by issuing docker -v to check the version.

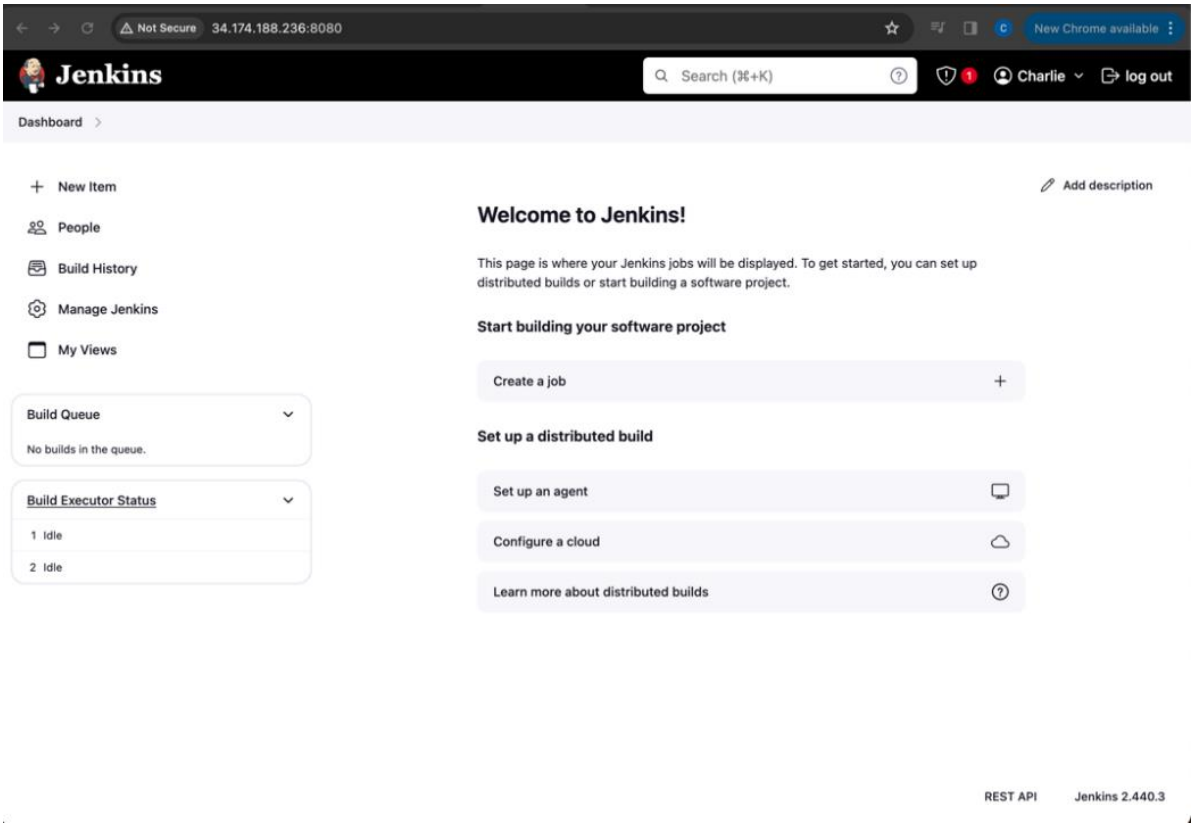


I then started an SSH terminal from within Google Cloud and started Jenkins installation.

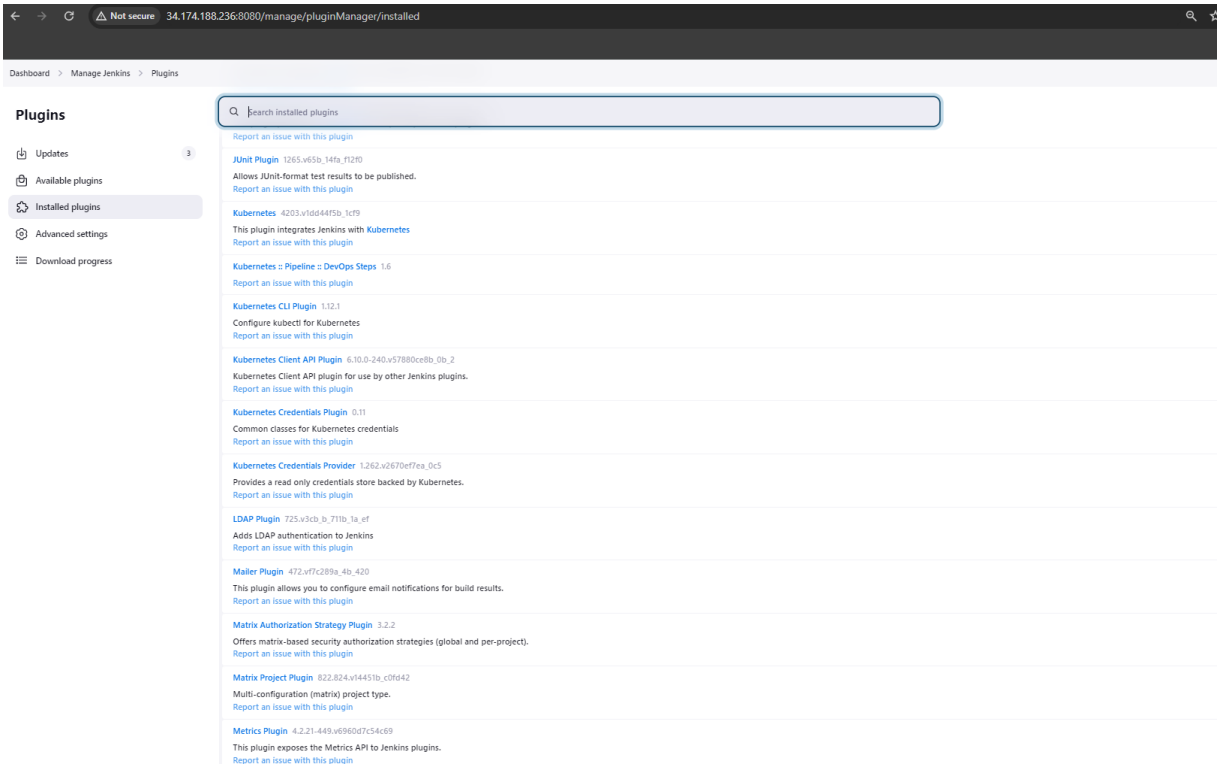


Once the installation was completed, I could Start Jenkins: systemctl start Jenkins. Throughout the course of completing this project I repeatedly checked the status by issuing a systemctl status Jenkins. There were also some instances where Jenkins froze on me, so I had to restart the application by issuing a systemctl restart Jenkins command.

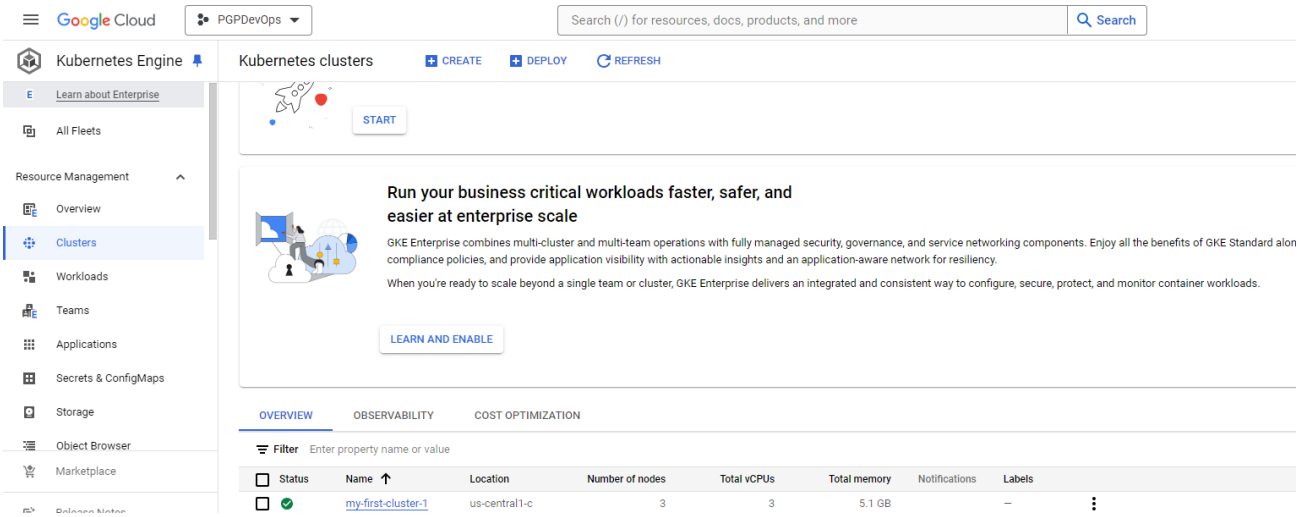
Connect to Jenkins from my local web browser by typing VMs public IP address and port 8080. Enter initial password and go through initial setup. Once done, I was directed to Jenkins dashboard.



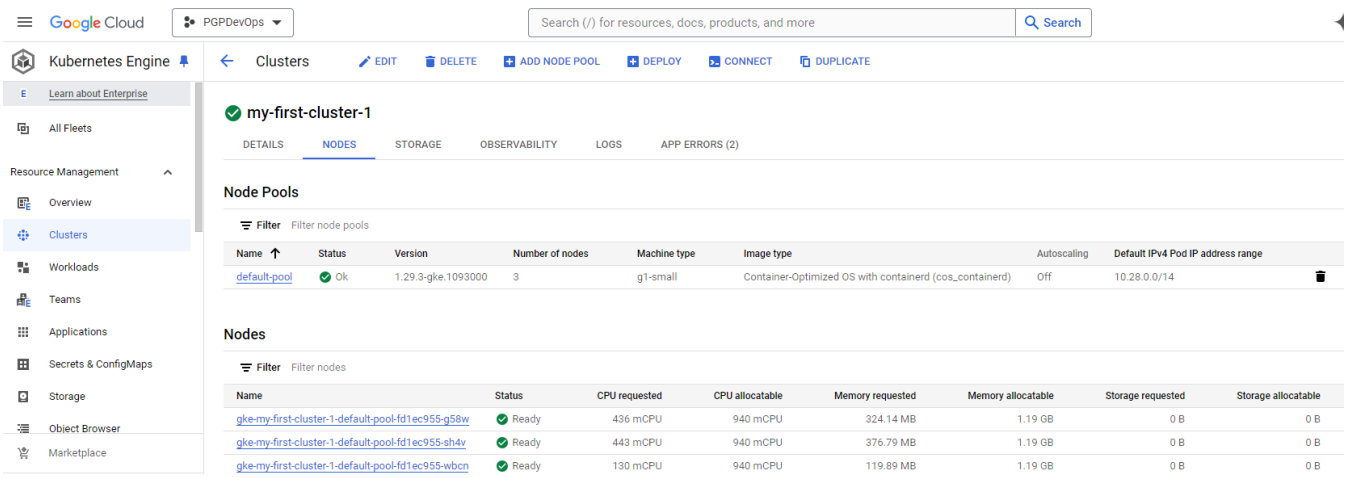
Throughout the use of Jenkins to complete this project, multiple plugins would be required to complete all the steps. This image shows some of the required plugins. Most notably, Kubernetes plugins.



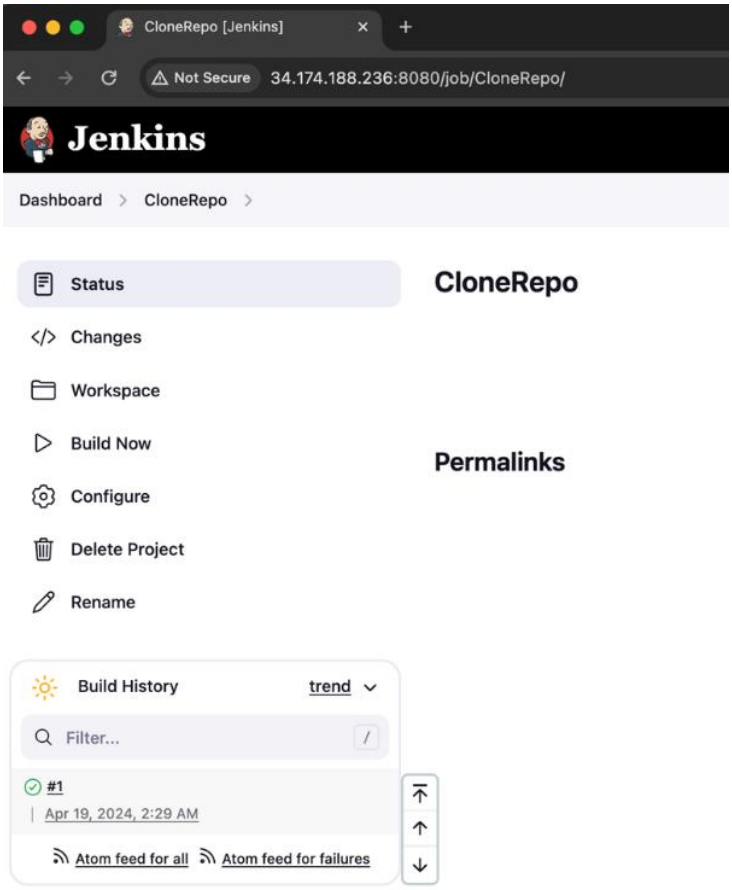
Before I could start with Jenkins, I needed to set up a Kubernetes environment where I could push out the completed container image. I chose to use GKE in Google Cloud. Google provides a nice service for first-time users that deploys a generic cluster at a lower price.

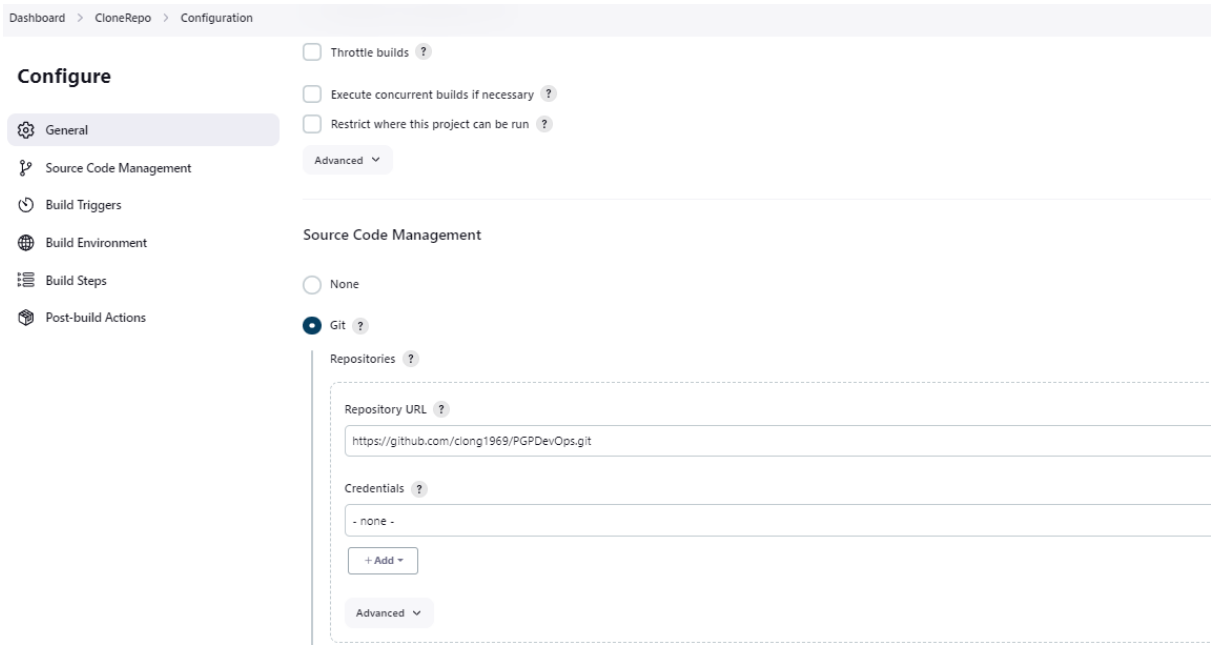


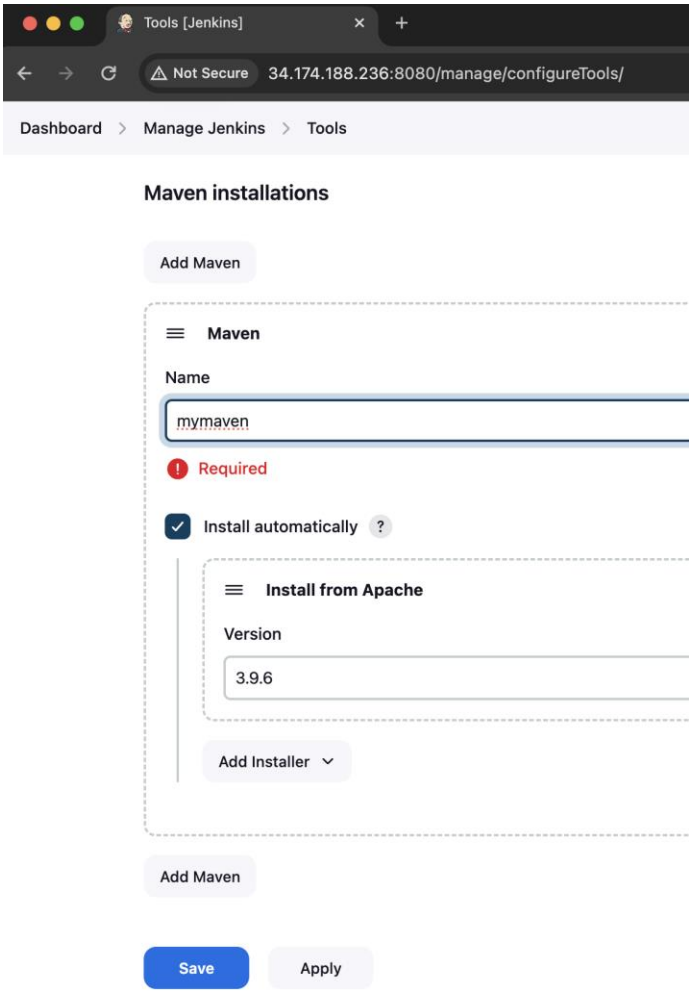
Nodes in the Kubernetes cluster.



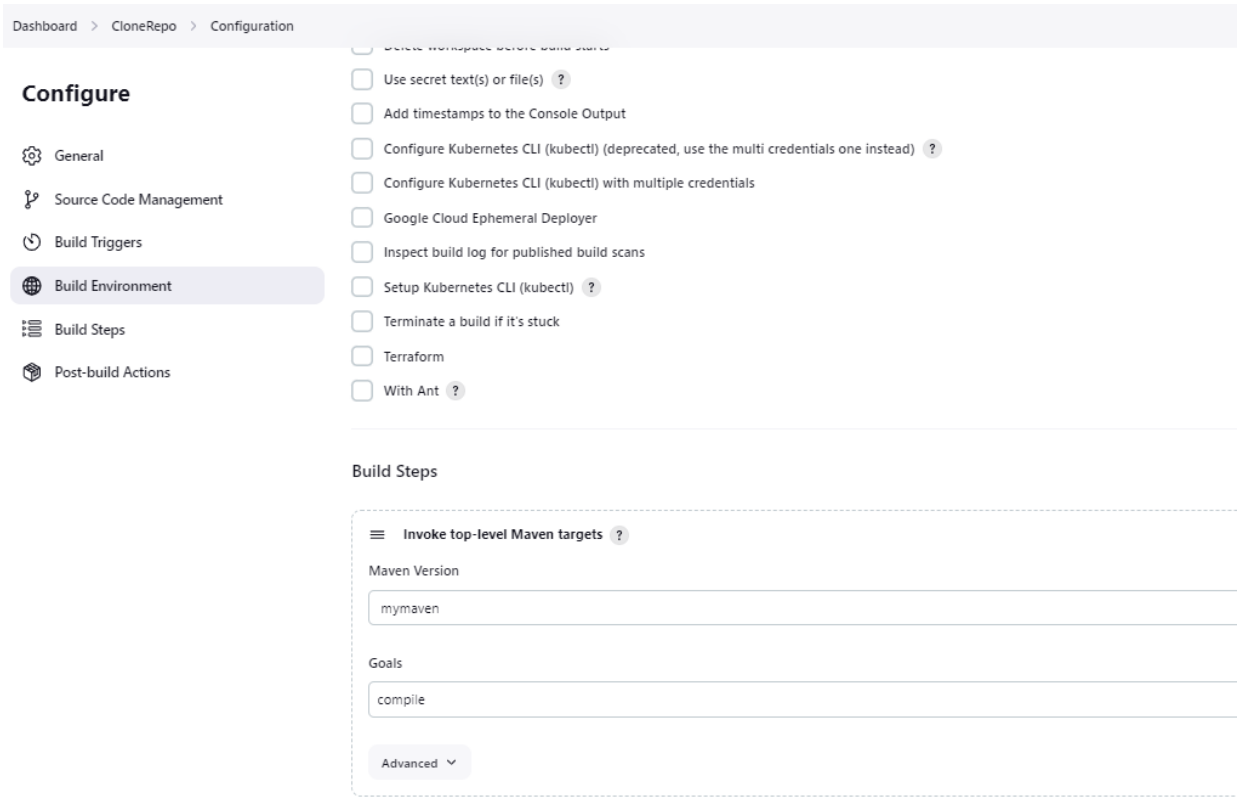
My first run through the Jenkins process consisted of running a series of Freestyle Projects. The first objective was to clone and compile my repo to the local Jenkins VM.



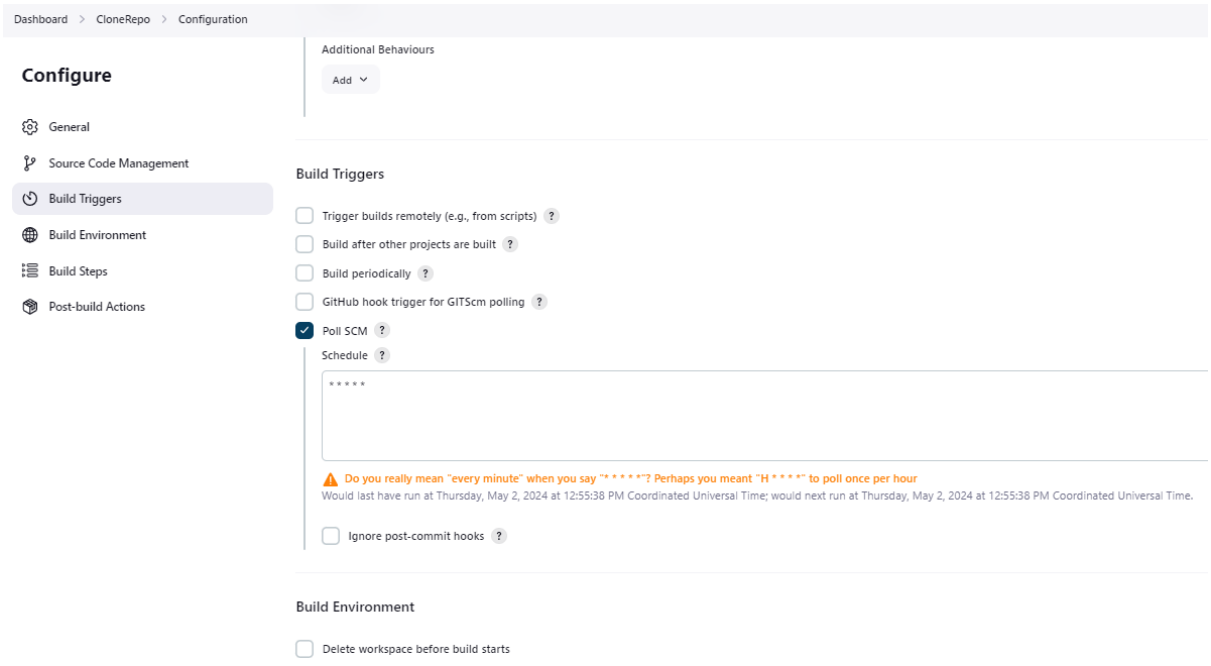

Dashboard > CloneRepo > Configuration
Configure
General
Source Code Management
Build Triggers
Build Environment
Build Steps
Post-build Actions
Throttle builds
Execute concurrent builds if necessary
Restrict where this project can be run
Advanced
Source Code Management
None
Git
Repositories
Repository URL
https://github.com/clong1969/PGPDevOps.git
Credentials
- none -
+ Add
Advanced

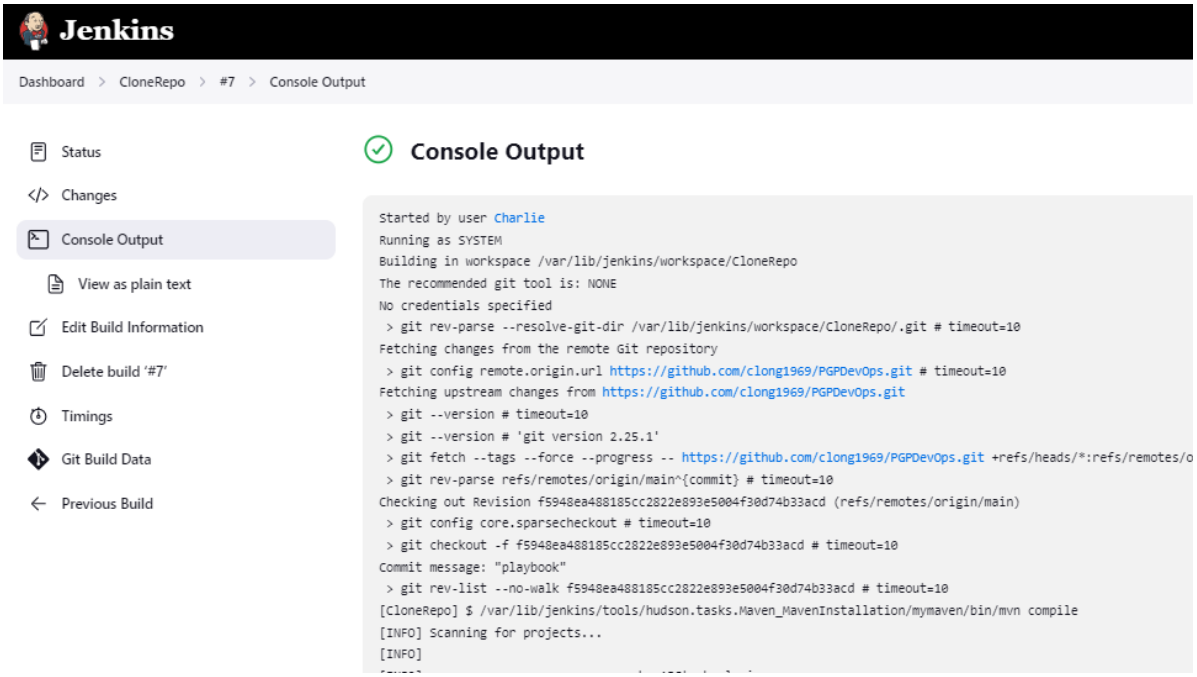


To compile the code, I invoked the TOP-LEVEL MAVEN target from plugins and added the goal of compile.

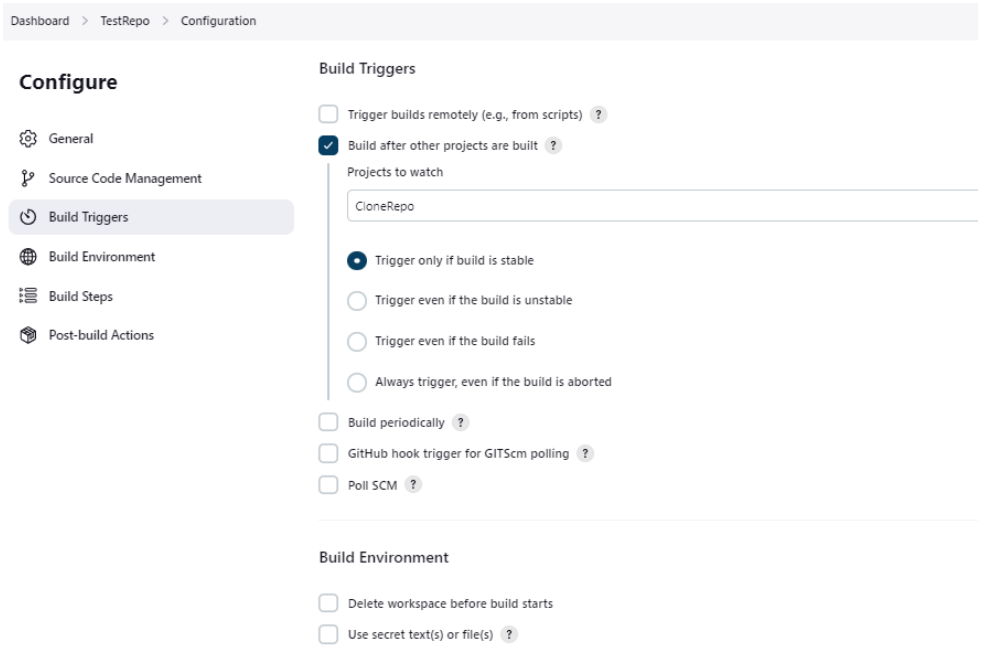


I entered a cron job in the BUILD TRIGGER so that the job would check the GitHub repo and start running if new files were added.

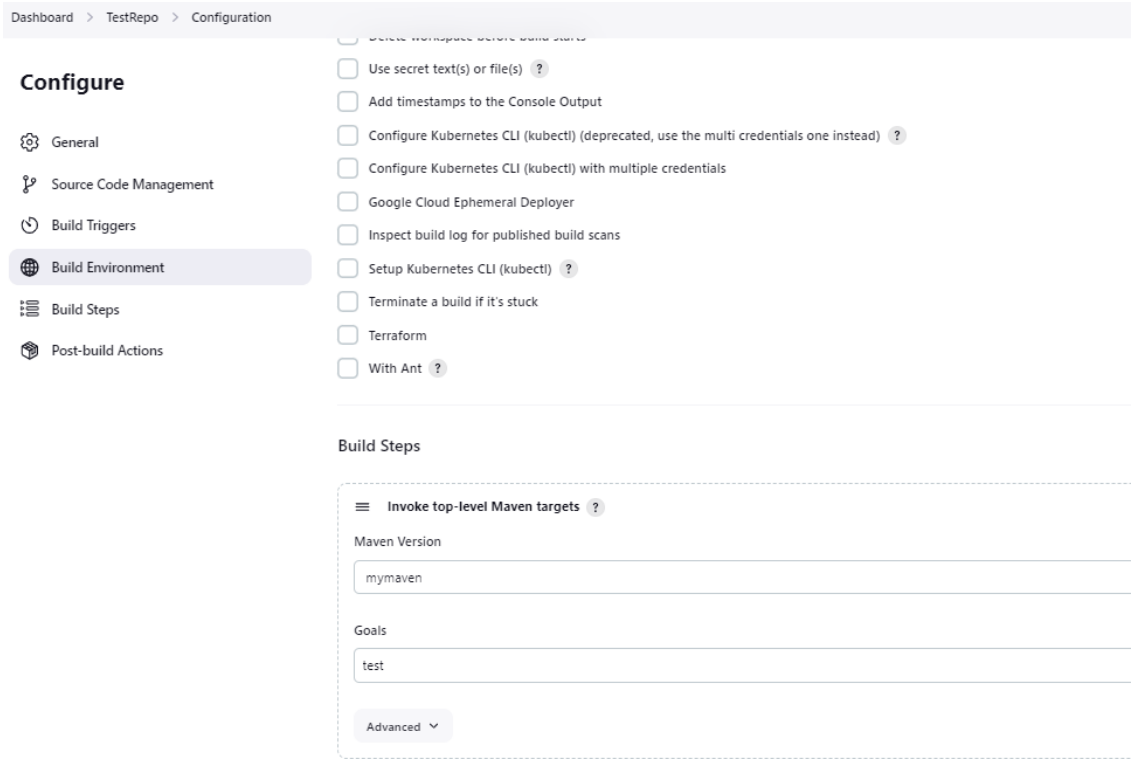


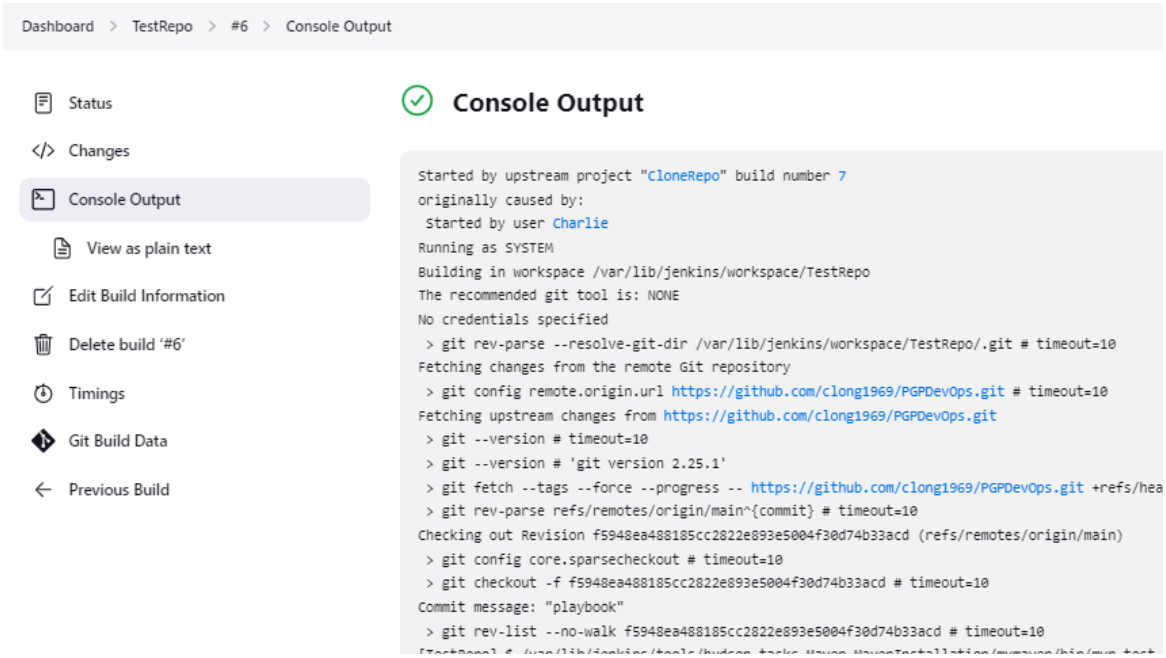


The next step was a FREESTYLE project to test the compiled code this was triggered by adding a project to watch of the CloneRepo project.

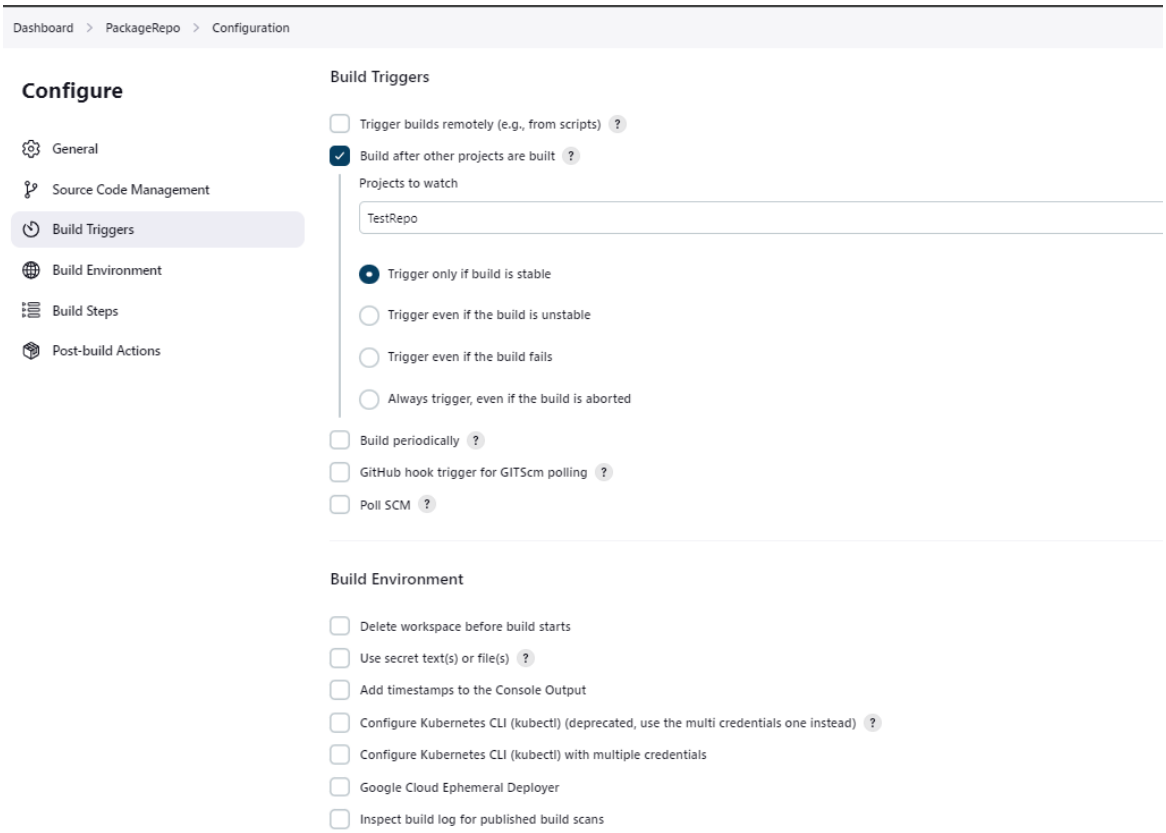


By adding the TOP-LEVEL Maven target and adding a goal to test, the project would test the compiled code.

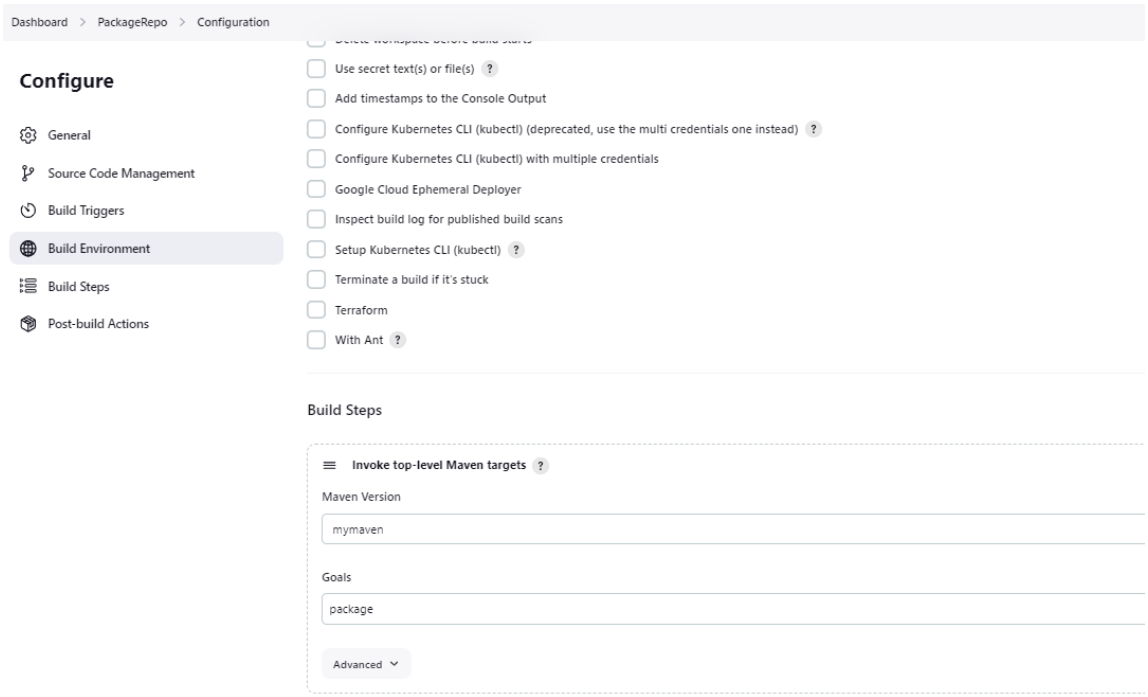


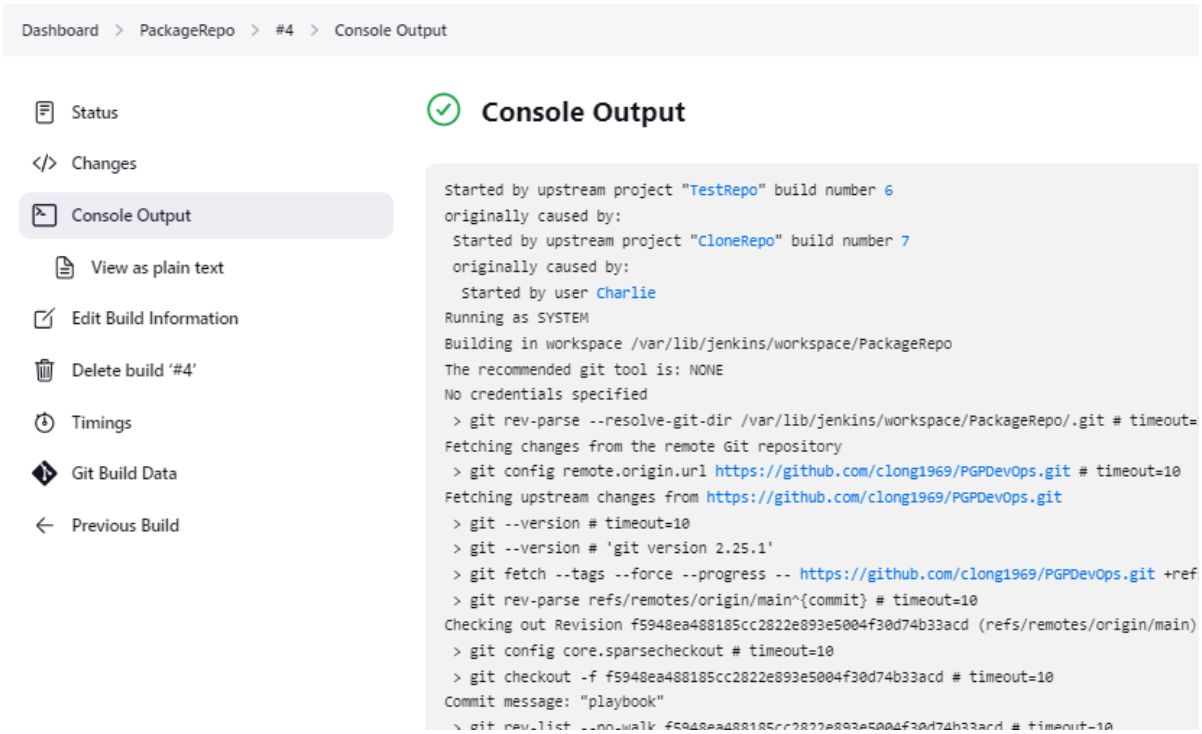


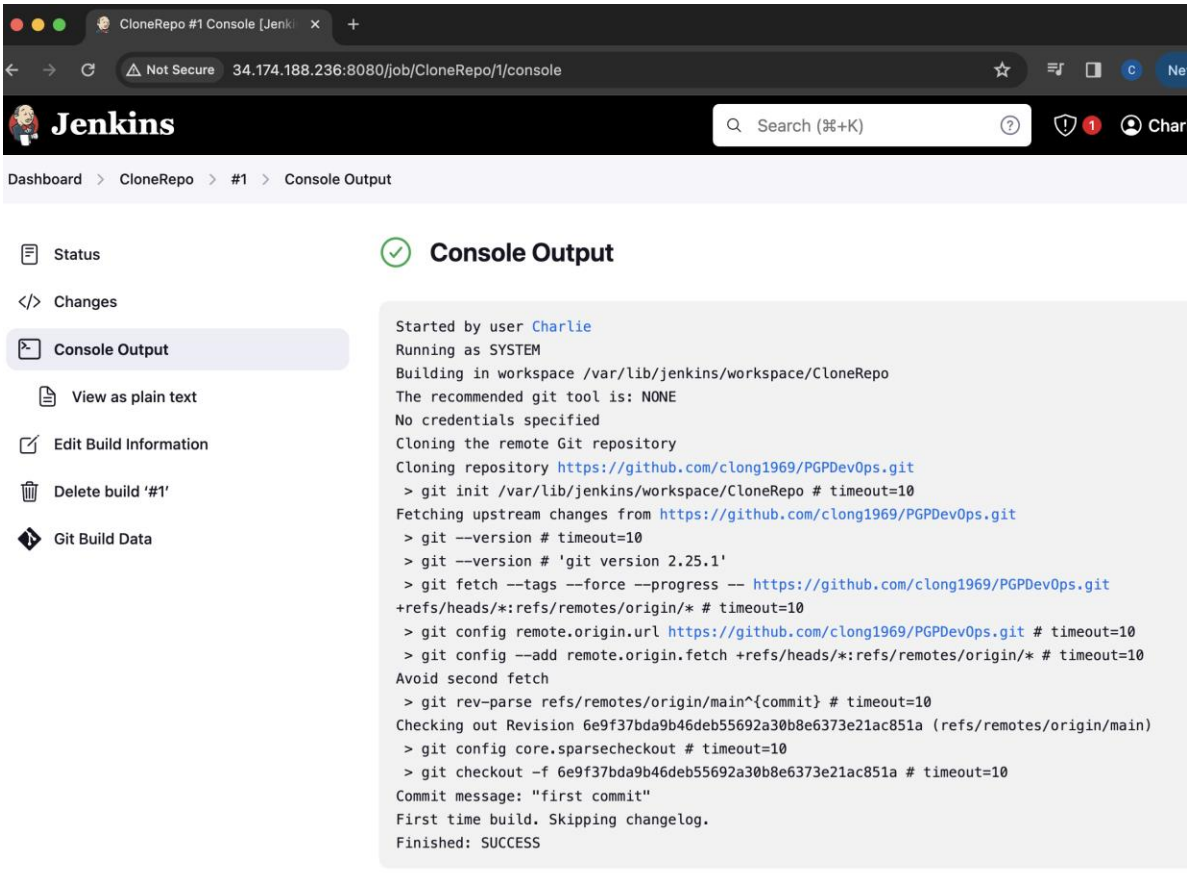
The next step was to package the code. This would be initiated after the TestRepo project completed.



The project would package the code due to adding the BUILD STEP, INVOKE TOP-LEVEL MAVEN targets and adding the goal of package.

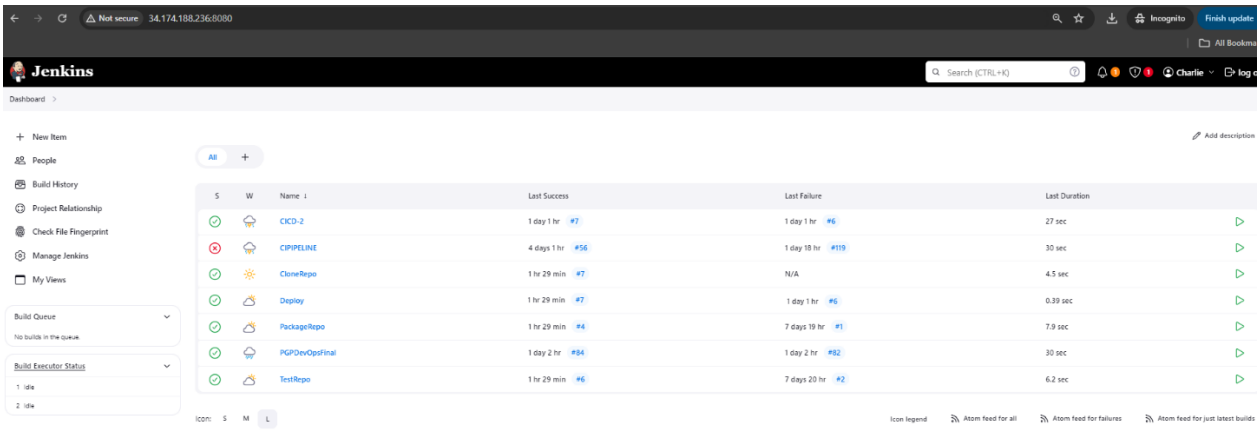



Dashboard > PackageRepo > #4 > Console Output
Status
Changes
Console Output
View as plain text
Edit Build Information
Delete build '#4'
Timings
Git Build Data
Previous Build
Console Output
Started by upstream project "TestRepo" build number 6
originally caused by:
Started by upstream project "CloneRepo" build number 7
originally caused by:
Started by user Charlie
Running as SYSTEM
Building in workspace /var/lib/jenkins/workspace/PackageRepo
The recommended git tool is: NONE
No credentials specified
> git rev-parse --resolve-git-dir /var/lib/jenkins/workspace/PackageRepo/.git # timeout=
Fetching changes from the remote Git repository
> git config remote.origin.url https://github.com/clong1969/PGPDevOps.git # timeout=10
Fetching upstream changes from https://github.com/clong1969/PGPDevOps.git
> git --version # timeout=10
> git --version # 'git version 2.25.1'
> git fetch --tags --force --progress -- https://github.com/clong1969/PGPDevOps.git +ref
> git rev-parse refs/remotes/origin/main^{commit} # timeout=10
Checking out Revision f5948ea488185cc2822e893e5004f30d74b33acd (refs/remotes/origin/main)
> git config core.sparsecheckout # timeout=10
> git checkout -f f5948ea488185cc2822e893e5004f30d74b33acd # timeout=10
Commit message: "playbook"

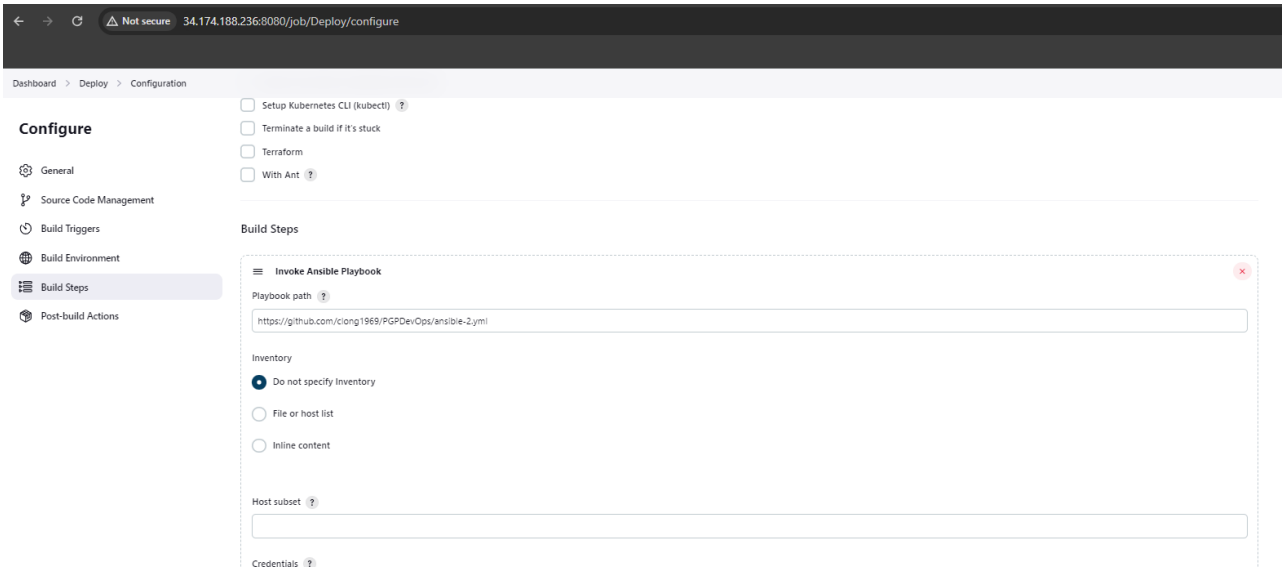

CloneRepo #1 Console [Jenki
Not Secure
34.174.188.236:8080/job/CloneRepo/1/console
Jenkins
Search (⌘+K)
Dashboard > CloneRepo > #1 > Console Output
Status
Changes
Console Output
View as plain text
Edit Build Information
Delete build '#1'
Git Build Data
Console Output
Started by user Charlie
Running as SYSTEM
Building in workspace /var/lib/jenkins/workspace/CloneRepo
The recommended git tool is: NONE
No credentials specified
Cloning the remote Git repository
Cloning repository https://github.com/clong1969/PGPDevOps.git
> git init /var/lib/jenkins/workspace/CloneRepo # timeout=10
Fetching upstream changes from https://github.com/clong1969/PGPDevOps.git
> git --version # timeout=10
> git --version # 'git version 2.25.1'
> git fetch --tags --force --progress -- https://github.com/clong1969/PGPDevOps.git
+refs/heads/*:refs/remotes/origin/* # timeout=10
> git config remote.origin.url https://github.com/clong1969/PGPDevOps.git # timeout=10
> git config --add remote.origin.fetch +refs/heads/*:refs/remotes/origin/* # timeout=10
Avoid second fetch
> git rev-parse refs/remotes/origin/main^{commit} # timeout=10
Checking out Revision 6e9f37bda9b46deb55692a30b8e6373e21ac851a (refs/remotes/origin/main)
> git config core.sparsecheckout # timeout=10
> git checkout -f 6e9f37bda9b46deb55692a30b8e6373e21ac851a # timeout=10
Commit message: "first commit"
First time build. Skipping changelog.
Finished: SUCCESS

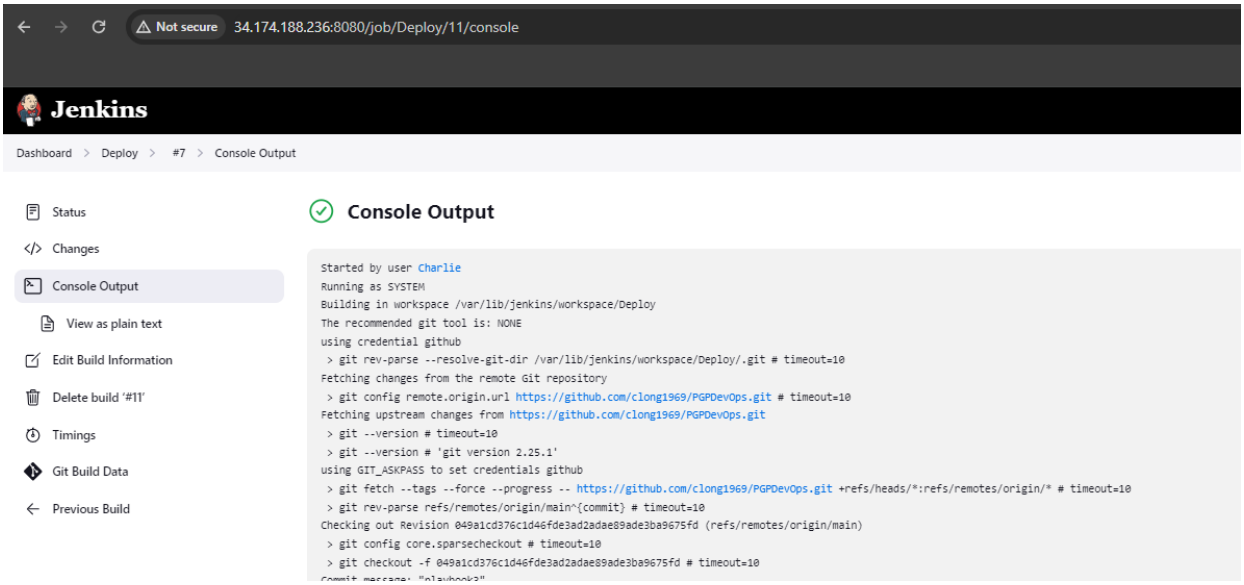
Steps 3, 4 and 5: Write docker file to push war to tomcat server, integrate docker with Ansible and deploy to Kubernetes.

The final step would be to create a container that would include the application and be deployed to container orchestration solution. For this, I included a dockerfile that had the path to the source code, included a web server image and an ansible playbook.



Used the INVOKE ANSIBLE PLAYBOOK and included playbook path https://github.com/clong1969/PGPDevOps/ansible-2.yml





After the projects were configured, I simply clicked the BUILD NOW icon on the CloneRepo project, and the completion of each project would trigger the next and the whole process completed.

I tested the image by running the docker container locally on the Jenkins server.

Running nodes and pods with workload:

```
charlielong1969@cloudshell:~ (pgpdevops)$ kubectl get nodes
NAME                                                  STATUS   ROLES    AGE   VERSION
gke-my-first-cluster-1-default-pool-fd1ec955-cczy   Ready    <none>   28h   v1.29.3-gke.1282000
gke-my-first-cluster-1-default-pool-fd1ec955-jyi8   Ready    <none>   28h   v1.29.3-gke.1282000
gke-my-first-cluster-1-default-pool-fd1ec955-rj0z   Ready    <none>   28h   v1.29.3-gke.1282000
charlielong1969@cloudshell:~ (pgpdevops)$ kubectl get pods
NAME                          READY   STATUS             RESTARTS   AGE
abctech-a-577fbf444b-ms7wm    1/1     Running            0          78s
abctech-a-577fbf444b-rdt27    1/1     Running            0          78s
abctech-a-577fbf444b-sn52v    1/1     Running            0          78s
```

Dockerfile consisted of these few lines:

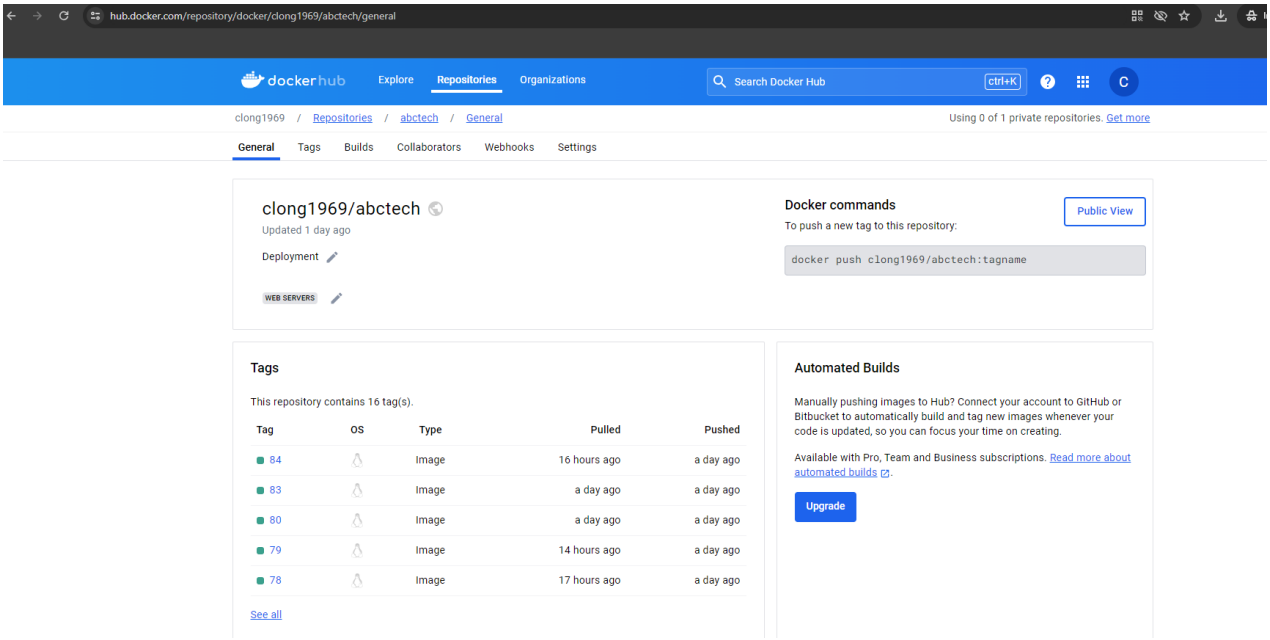
FROM tomcat:9

ADD ABCtechnologies-1.0.war /usr/local/tomcat/webapps

CMD ["catalina.sh", "run"]

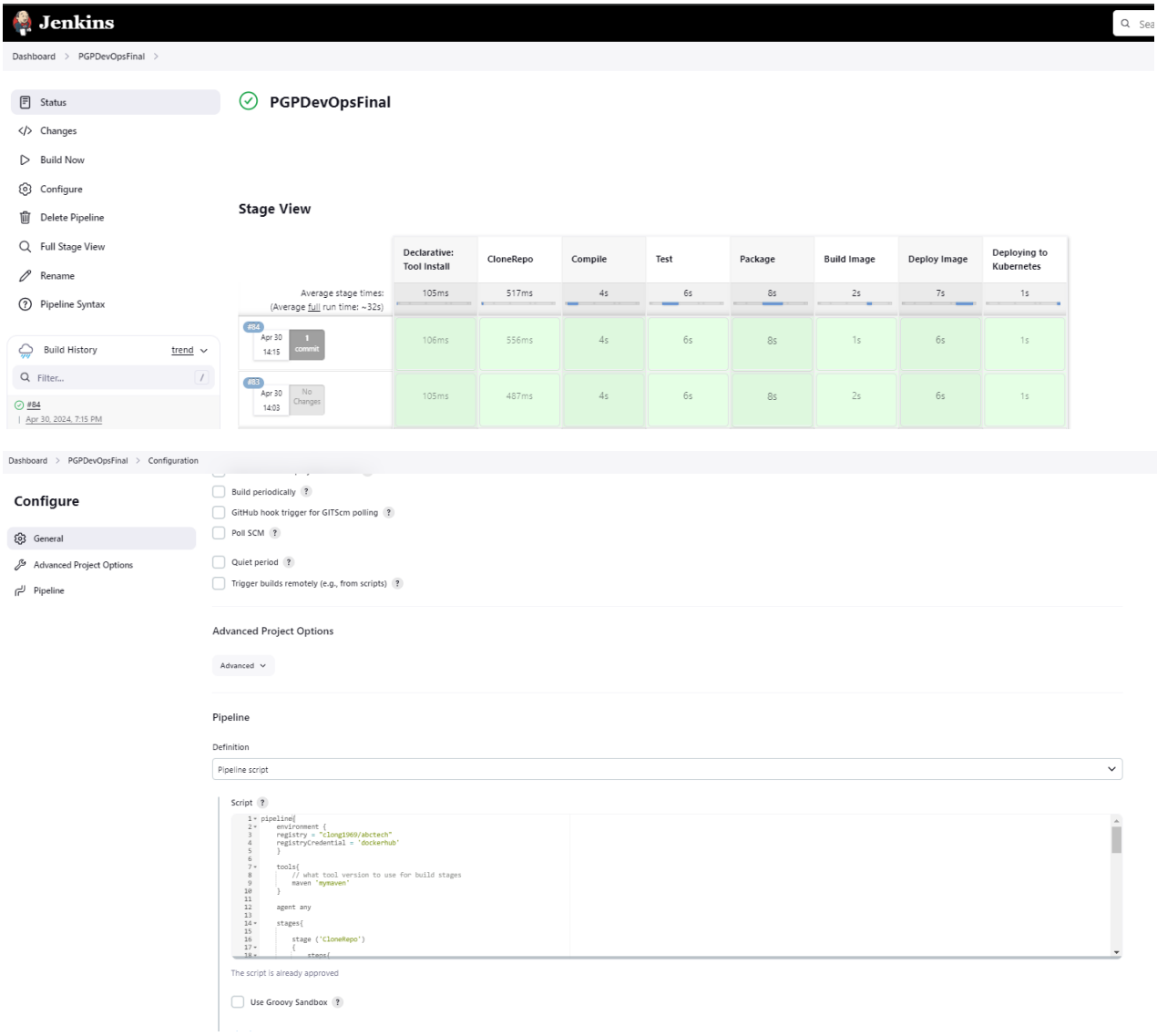
EXPOSE 8080

This process was run numerous times, but the attached image shows that the image was successfully deployed to dockerhub on each successful run. All images from runs are located here: https://hub.docker.com/repository/docker/clong1969/abctech/general

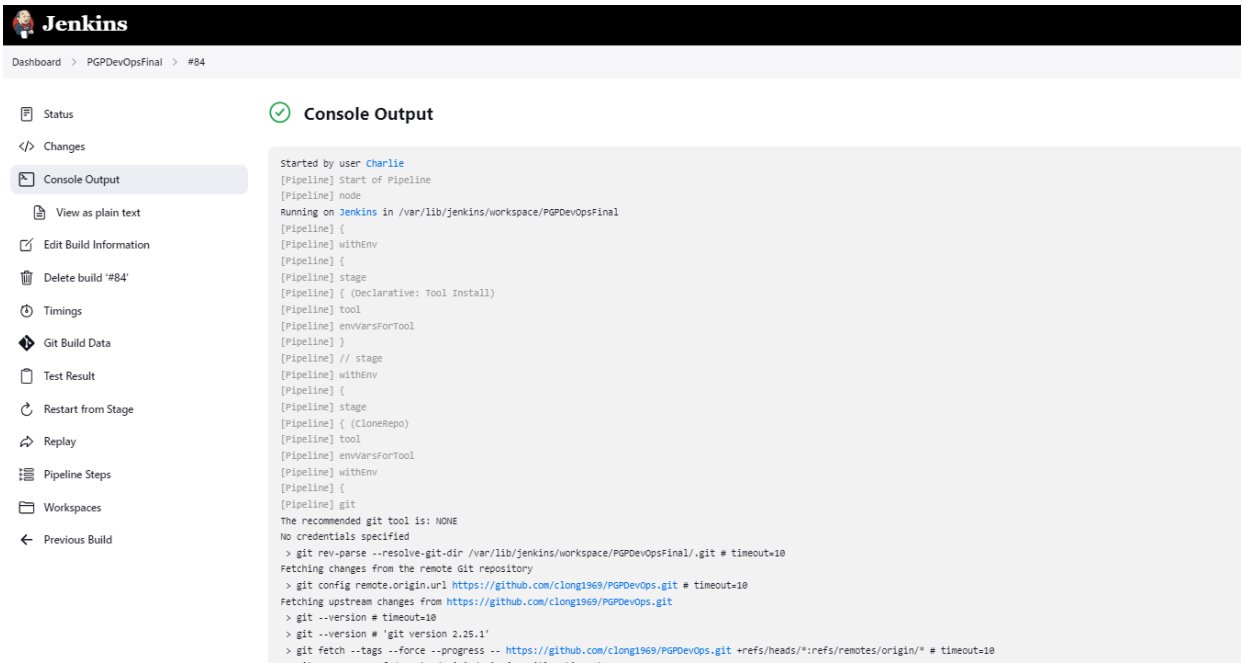


While this process was a good solution for this type of project, in a production environment something more robust would be better suited. For the next attempt, I did a complete pipeline by going into Jenkins, selecting NEW ITEM and selecting PIPELINE.

This would be a single pipeline with a series of stages that would accomplish the same process as the multiple projects above. The process would provide a graphical representation of where the build was at each stage.


Jenkins
Dashboard > PGPDevOpsFinal >
Status
Changes
Build Now
Configure
Delete Pipeline
Full Stage View
Rename
Pipeline Syntax
Build History
trend
Filter...
#84
Apr 30, 2024, 7:15 PM
PGPDevOpsFinal
Stage View
Declarative: Tool Install
CloneRepo
Compile
Test
Package
Build Image
Deploy Image
Deploying to Kubernetes
Average stage times:
(Average full run time: ~32s)
105ms
517ms
4s
6s
8s
2s
7s
1s
#84
Apr 30
14:15
1 commit
106ms
556ms
4s
6s
8s
1s
6s
1s
#83
Apr 30
14:03
No Changes
105ms
487ms
4s
6s
8s
2s
6s
1s

Dashboard > PGPDevOpsFinal > Configuration
Configure
General
Advanced Project Options
Pipeline
Build periodically
GitHub hook trigger for GITScm polling
Poll SCM
Quiet period
Trigger builds remotely (e.g., from scripts)
Advanced Project Options
Advanced
Pipeline
Definition
Pipeline script
Script
pipeline{
environment {
registry = "clong1969/abctech"
registryCredential = 'dockerhub'
}
tools{
// what tool version to use for build stages
maven 'mymaven'
}
agent any
stages{
stage ('CloneRepo')
The script is already approved
Use Groovy Sandbox



The entire deployment required less then 80 lines of code.

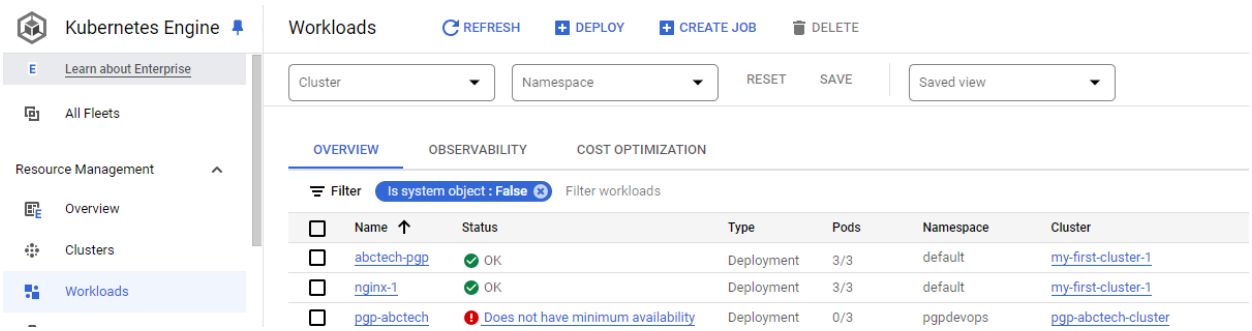
```
pipeline{
   environment {
   registry = "clong1969/abctech"
   registryCredential = 'dockerhub'
   }
   tools{
      // what tool version to use for build stages
      maven 'mymaven'
   }
   agent any
   stages{
      stage ('CloneRepo')
      {
         steps{
           git (branch: 'main', url:'https://github.com/clong1969/PGPDevOps.git')
```

```
        }
}
    stage ('Compile')
    {
       steps{
        sh  'mvn compile'
       }
    }
    stage ('Test')
    {
       steps{
        sh  'mvn test'
       }
    post{
       success{
          junit 'target/surefire-reports/*.xml'
       }
    }
    }
    stage('Package')
    {
       steps{
          sh 'mvn package'
       }
    }
    stage('Build Image')
    {
       steps{
       sh 'cp /var/lib/jenkins/workspace/PGPDevOpsFinal/target/ABCtechnologies-1.0.war .'
```

```
    script {
        dockerImage = docker.build registry + ":$BUILD_NUMBER"
        }
    }
}
stage('Deploy Image')
{
    steps{
      script {
        dockerImage = docker.build registry + ":$BUILD_NUMBER"
        docker.withRegistry( '', registryCredential ) {
        dockerImage.push()}
        }
    }
}
stage('Deploying to Kubernetes')
{
    steps{
      sh("sed -ie s/abctech:latest/dockerImage/g deployment.yml")

      step([$class: 'KubernetesEngineBuilder', projectId: 'pgpdevops', clusterName: 'my-first-cluster-1', location: 'us-central1-c', manifestPattern: 'deployment.yml', credentialsId: 'pgpjenkins'])
    }
    }
}
```

Once the deployment was complete, I had workloads running in Google Cloud GKE.



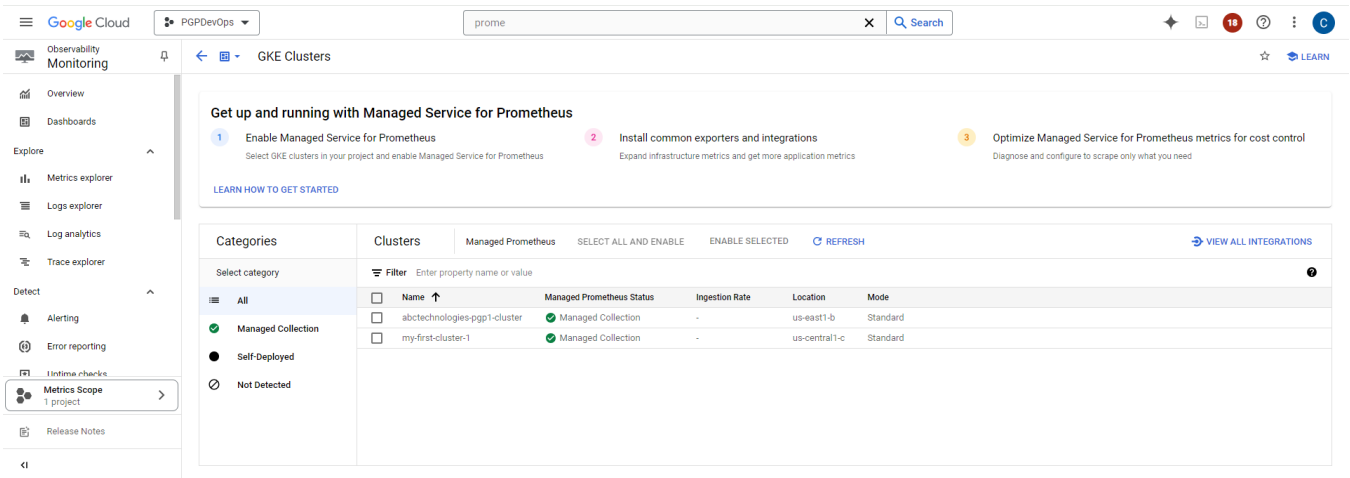
Tested workload by connecting to application:

Step 6 Monitor resources with Grafana.

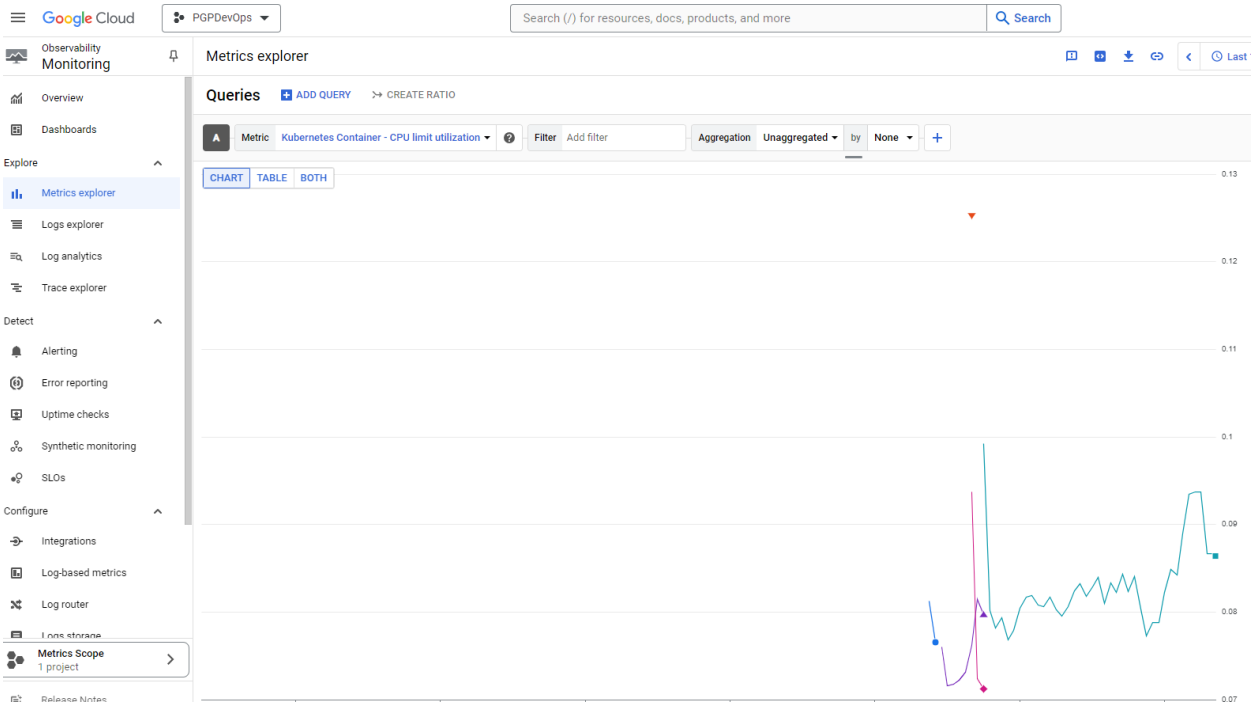
This should really be two steps. Prometheus monitors your Kubernetes cluster and Grafana takes the data from the log collections and puts it into an easy to read graphical format which can be customized to the end-user's needs.

For this step, I selected the Prometheus managed service that is included within Google Cloud. This was fairly straightforward process that included selecting the service, deploying the service and selecting the resources that would be monitored. For resources that required an agent, the service deployed agents to selected devices.

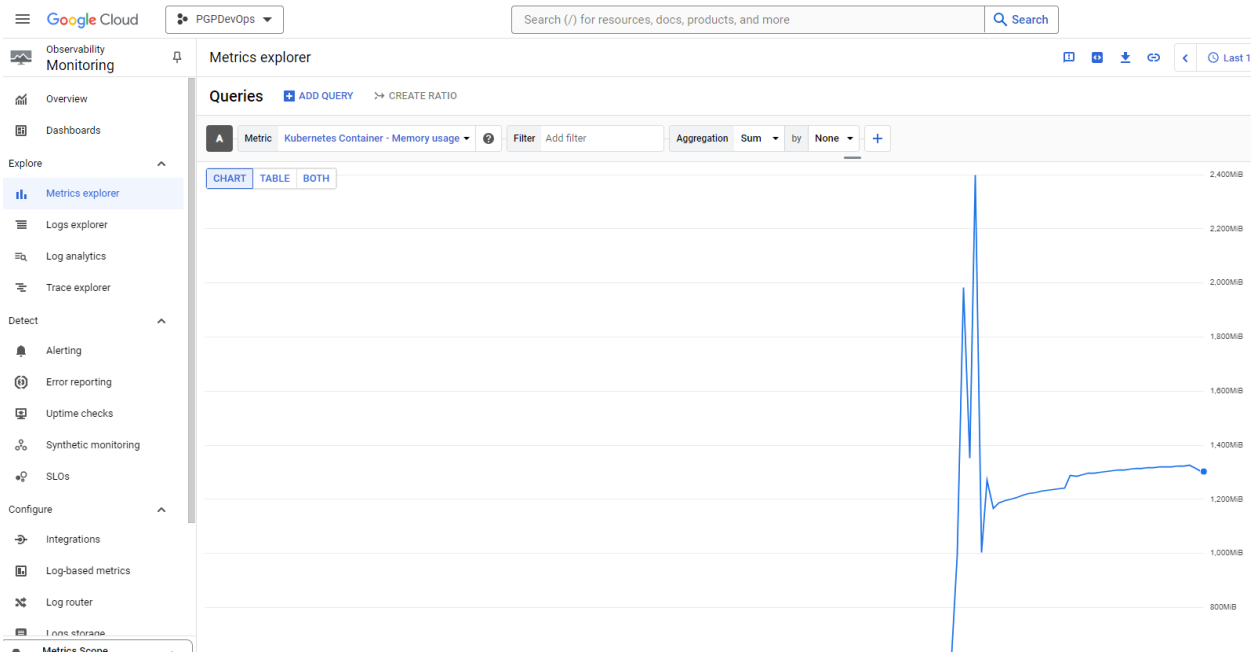
This is the Google Cloud interface for monitoring with Prometheus.



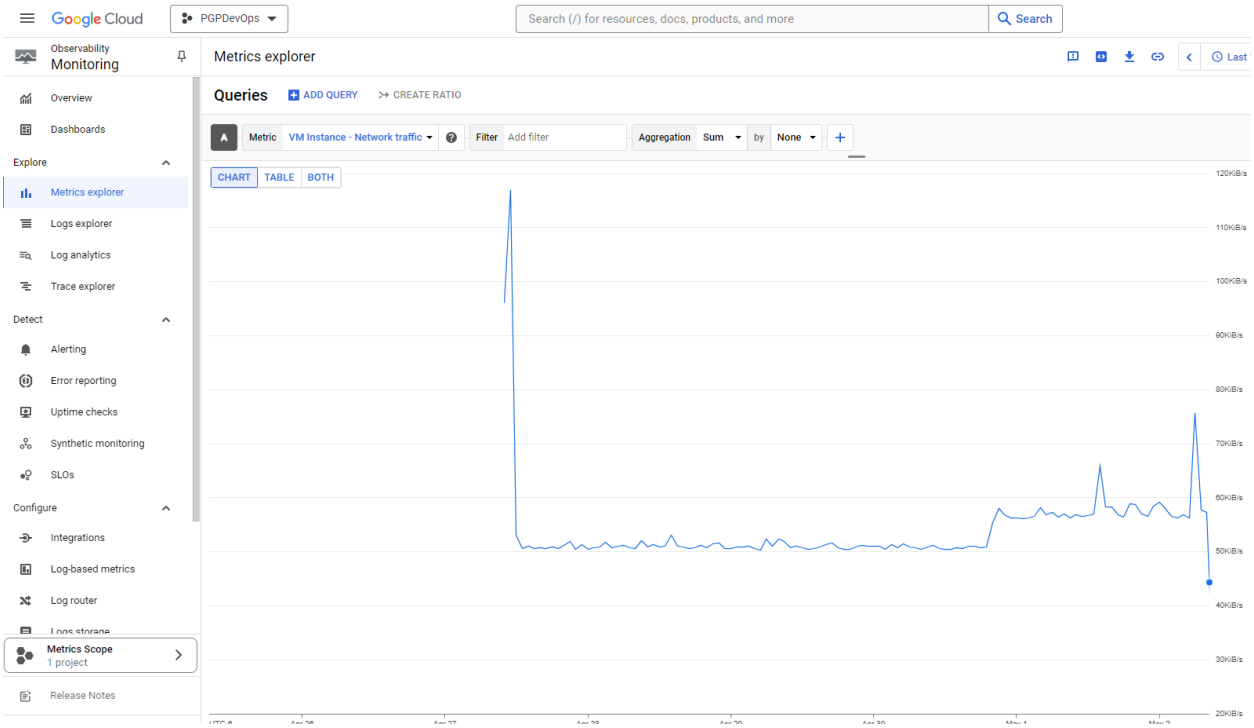
CPU metrics monitoring:



Memory metrics monitoring:

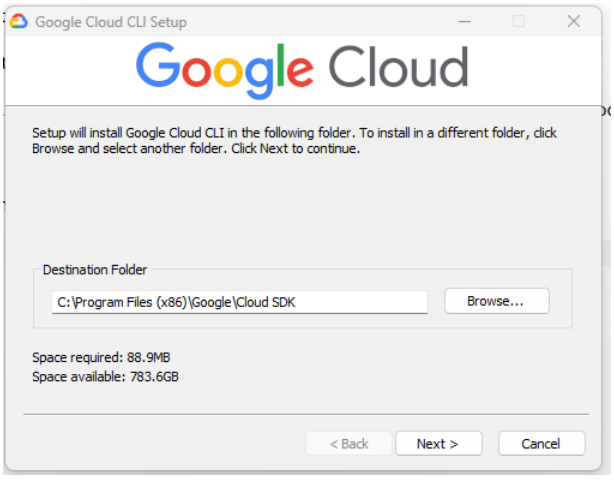


Network traffic from the Kubernetes nodes:

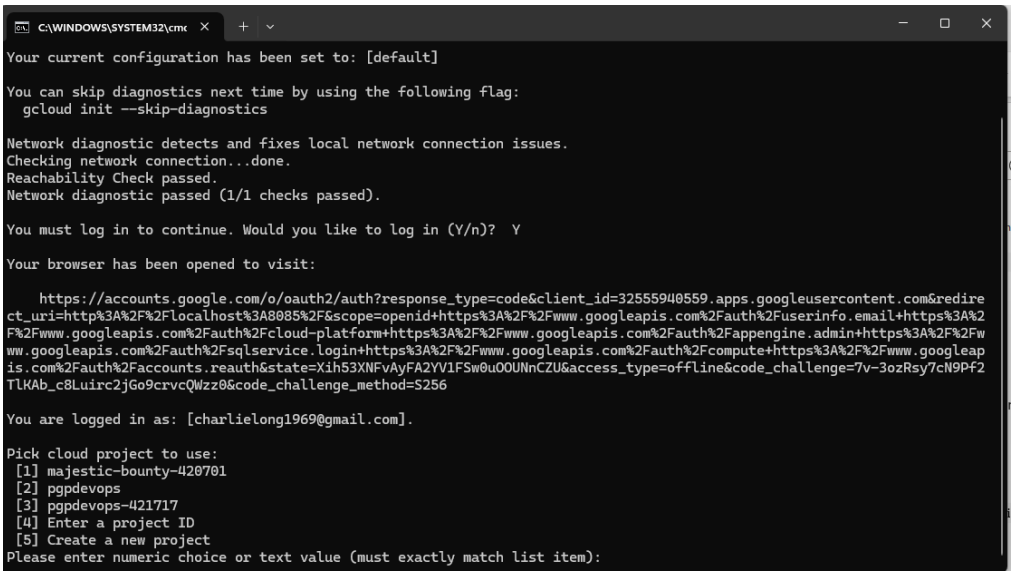


Once I selected resources to monitor, I installed Grafana to translate the logs into easily readable graphs. This would require installing Google Cloud CLI, Node.JS and Grafana on my local machine.

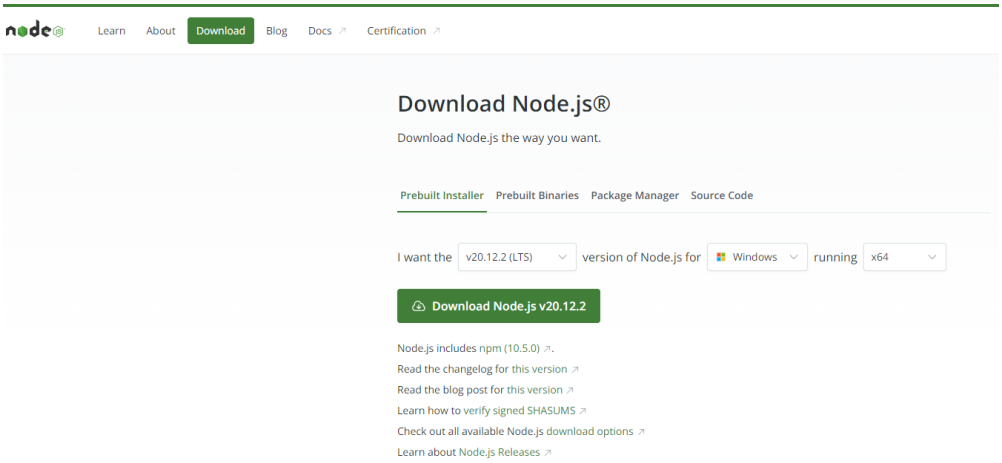
Install Google Cloud CLI:



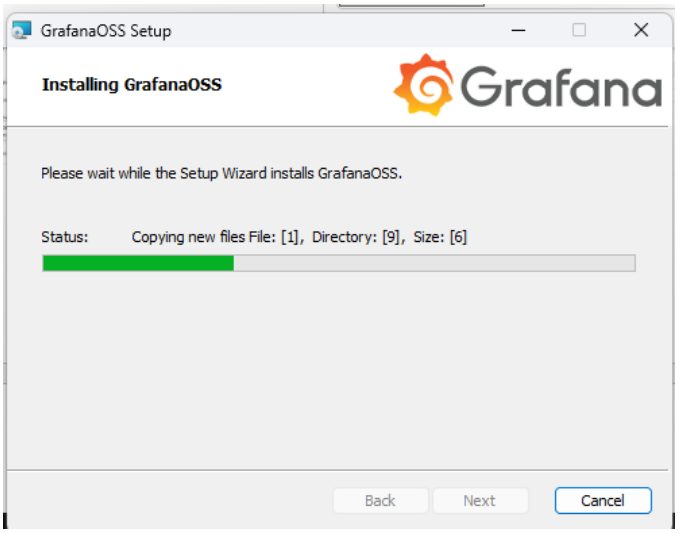
Select which environment you would like to connect to.



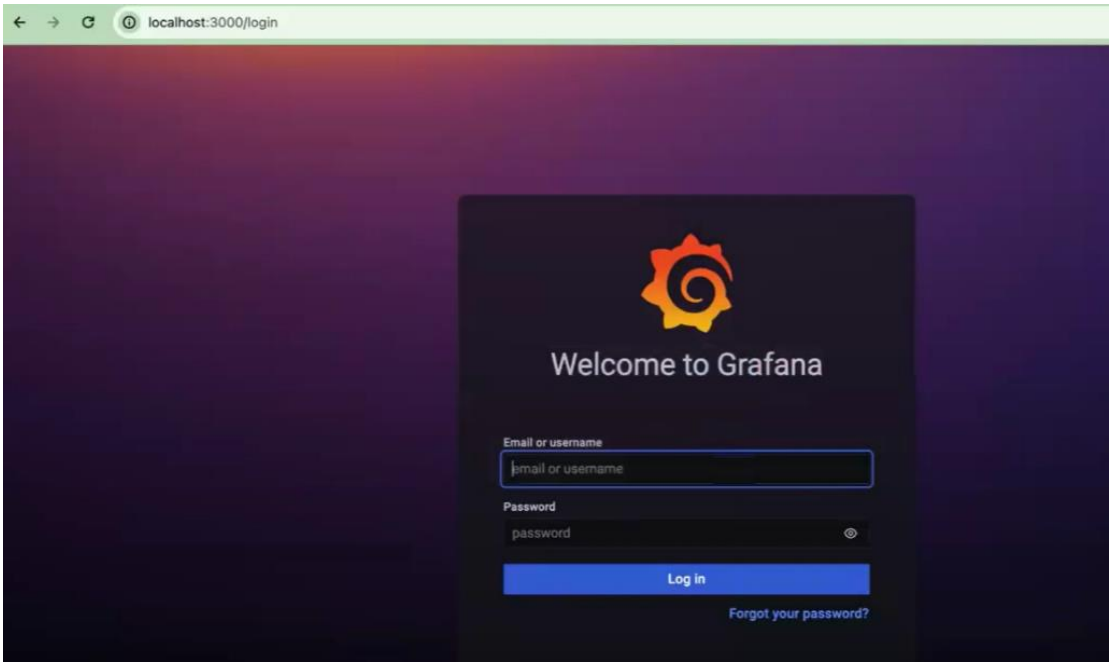
Install Node.JS



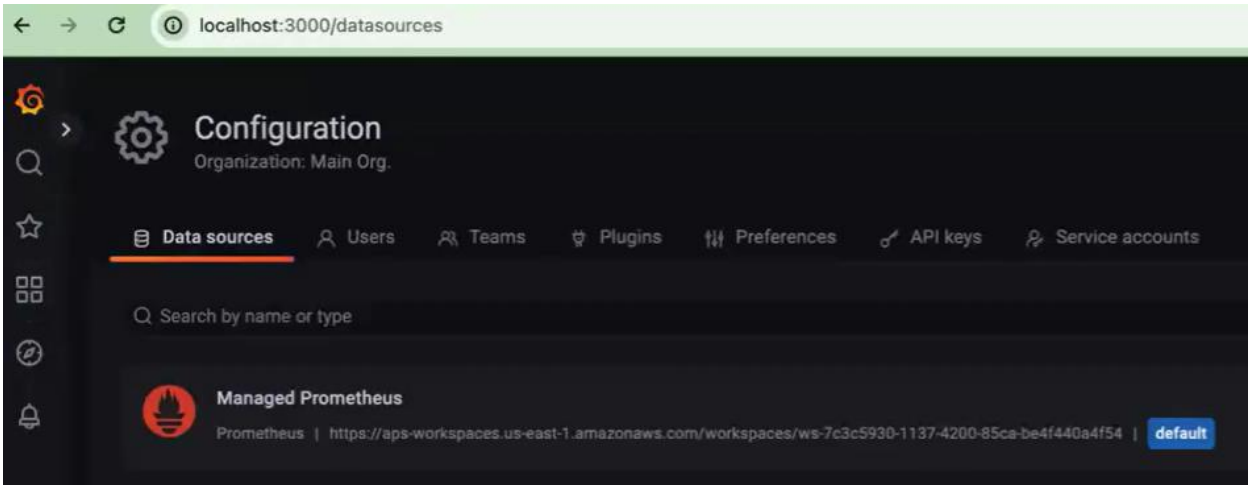
Install Grafana:



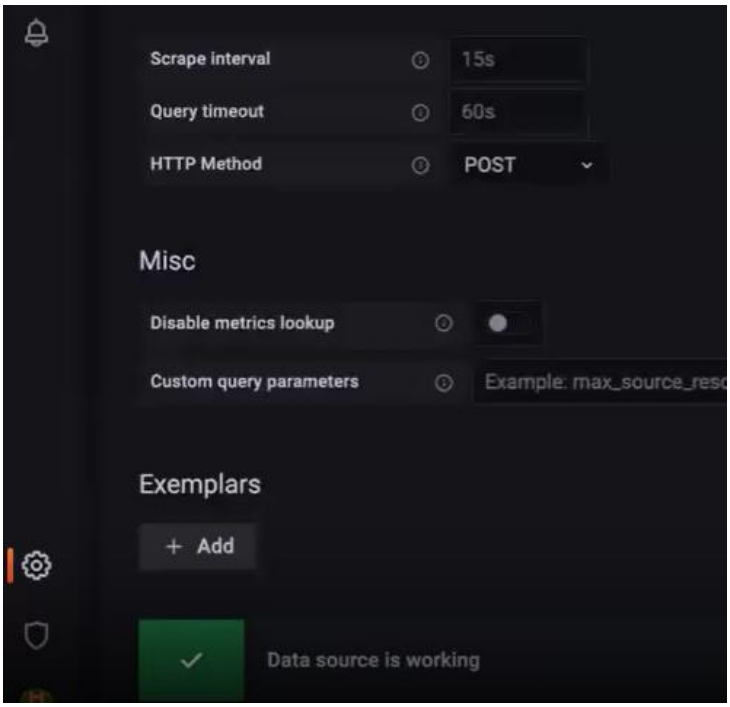
Log into Grafana after installation is completed. In order to log into Grafana, users must first forward port in Kubernetes environment to port 3000.



Select your data source as Prometheus.



Verify that your data source is working.



Finally, view your collected data in a graphical format. Grafana offers many types of displays and widgets, such as gauges, bar charts, line graphs, etc.

Those are all the tasks that were outlined for completion. Overall, this was a challenging project the encompassed the breadth of material that has been covered over the past several months. Most of the issues I had was choosing to use Google as my Kubernetes platform. I was not familiar with the environment and there were things I ran into that were not discussed in class sessions. I probably spent close to 50% of my time on this project figuring out how to configure Google Cloud. Using Amazon probably would have been easier as it was used more heavily during the live courses.

Addendum:

Before deciding to use the GKE service in Google Cloud I deployed two VMs to do a small Kubernetes installation. To do this, I ran the following commands. After completing the process, I ran into multiple issues and opted to go for the provided service from Google.

Building K8s cluster.

Stand up two Ubuntu VMs in Google Cloud.

Connect to each node via SSH.

Commands to create nodes:

Switched to root user by issuing sudo su – command.

Ran following commands on both nodes:

`apt-get update`

`apt-get install -y apt-transport-https ca-certificates curl gpg`

```
echo 'deb [signed-by=/etc/apt/keyrings/kubernetes-apt-keyring.gpg]
https://pkgs.k8s.io/core:/stable:/v1.30/deb/ /' | sudo tee /etc/apt/sources.list.d/kubernetes.list
```

```
apt-get update
apt-get install -y kubelet kubeadm kubectl
apt-mark hold kubelet kubeadm kubectl
```

```
curl -s https://packages.cloud.google.com/apt/doc/yum-key.gpg | apt-key add -
echo "deb http://apt.kubernetes.io/ kubernetes-xenial main" >/etc/apt/sources.list.d/kubernetes.list
apt-get update
apt -y install kubeadm kubectl kubelet
sed -i 's|disabled_plugin|#disabled_plugin|g' /etc/containerd/config.toml
service containerd restart
```

Ran following commands on master node only:

```
kubeadm init --ignore-preflight-errors=all
```

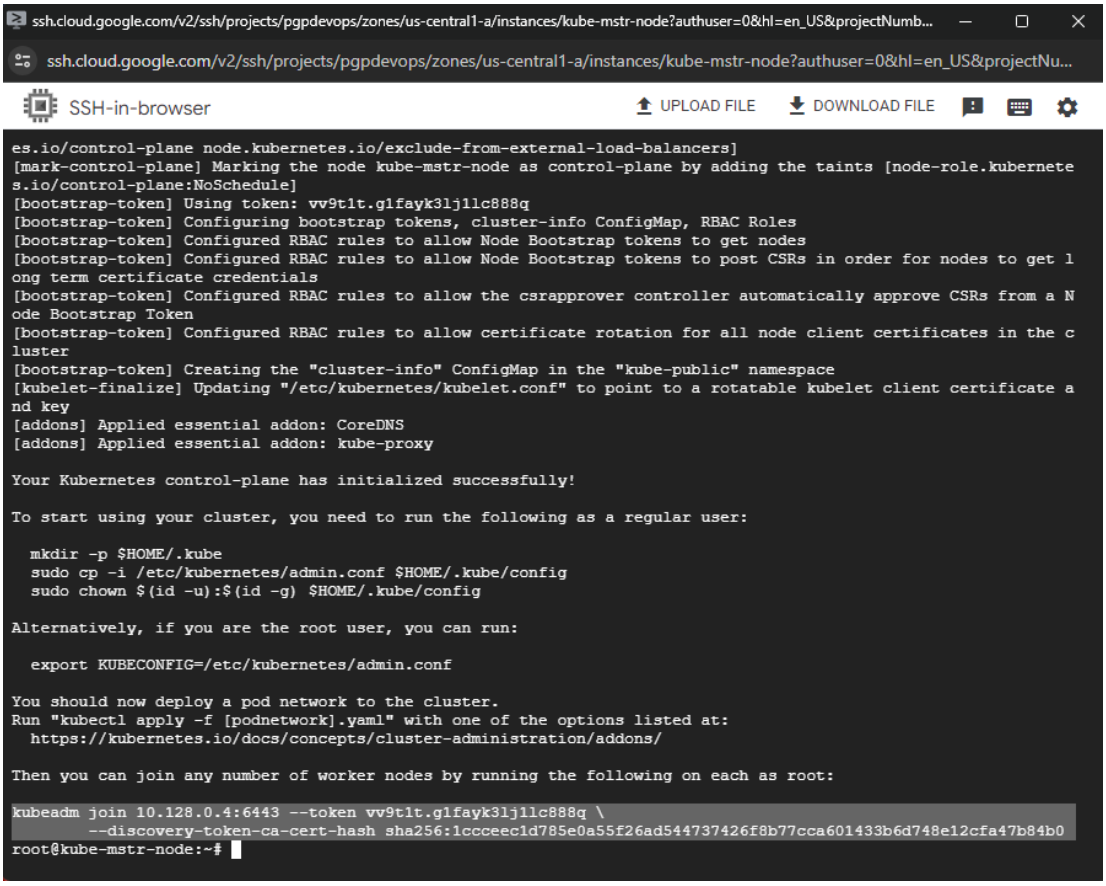
Execute these commands

```
mkdir -p $HOME/.kube
sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
sudo chown $(id -u):$(id -g) $HOME/.kube/config
```

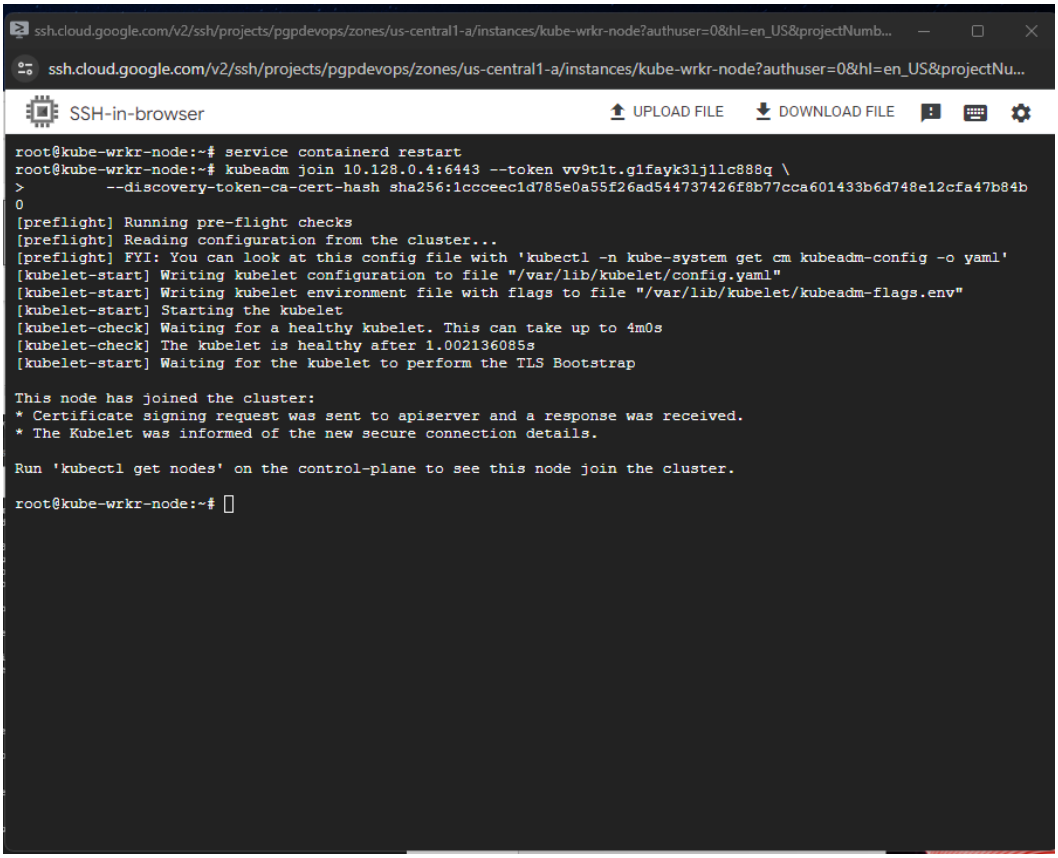
Execute below command to install weave network driver

```
kubectl apply -f https://github.com/weaveworks/weave/releases/download/v2.8.1/weave-daemonset-k8s.yaml
```

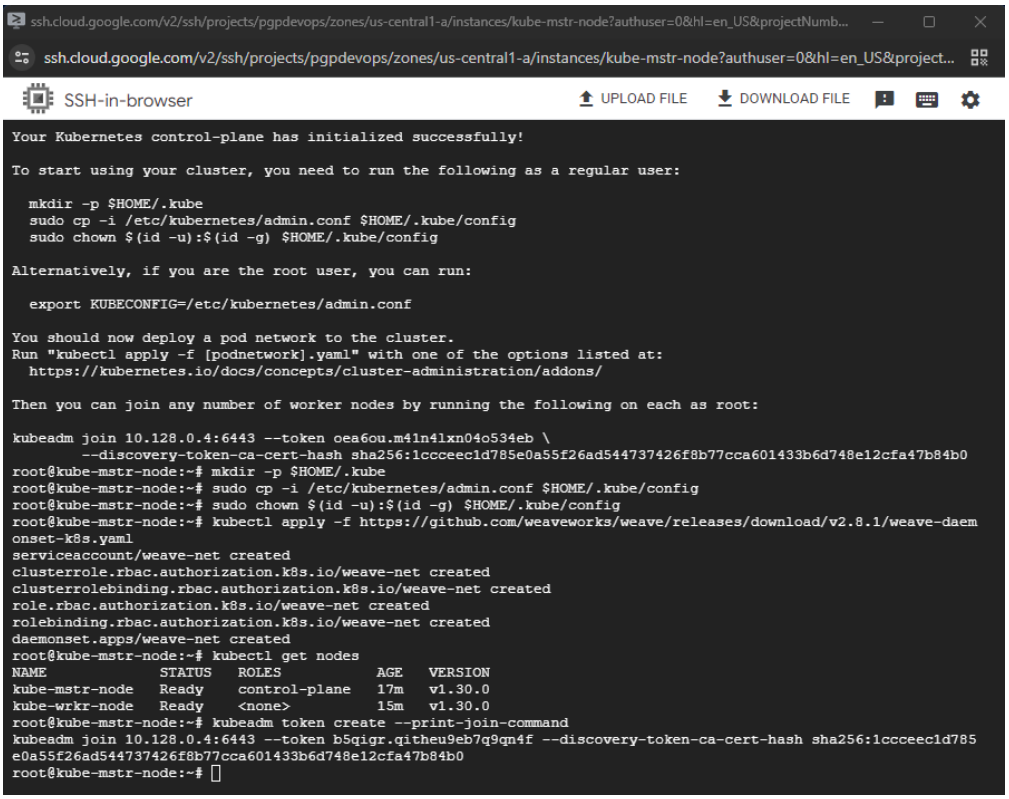
After initializing master node, get command to initialize worker node.



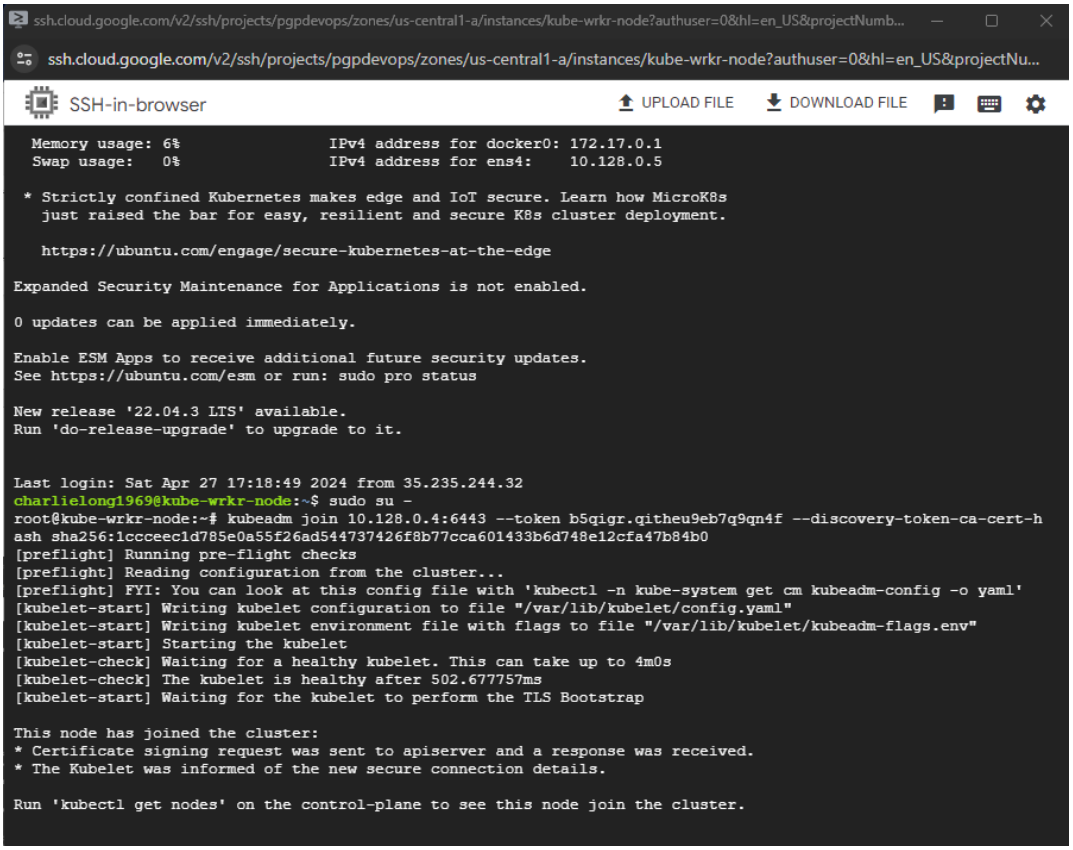
Worker node joined:



Generate token:



After applying token to worker node:

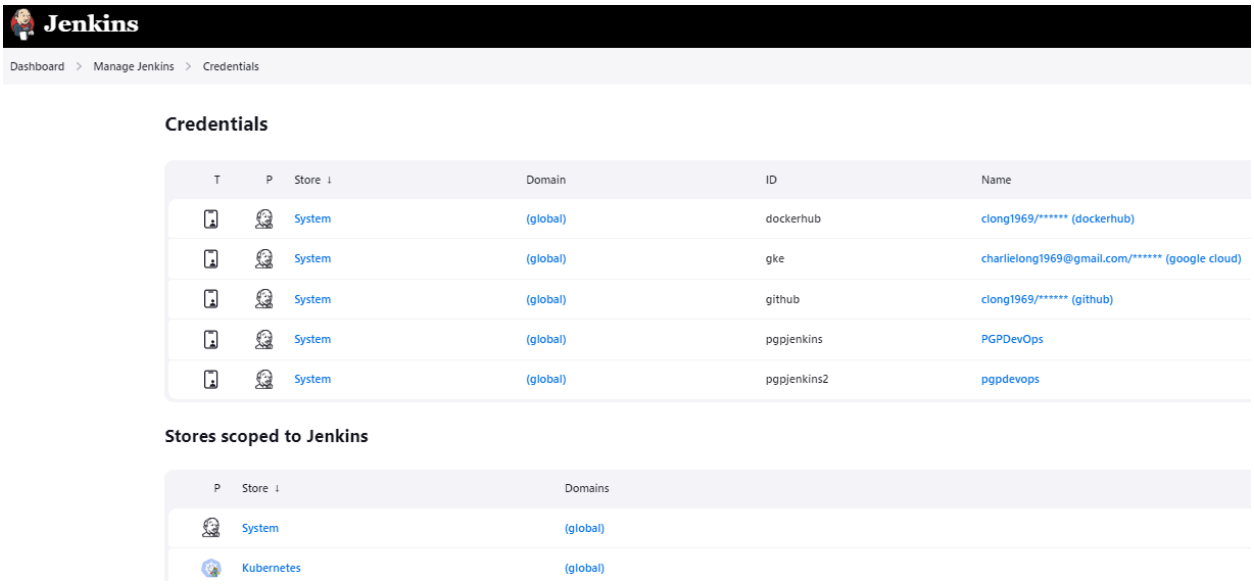



```
charlielong1969@cloudshell:~ (pgpdevops)$ kubectl get nodes
NAME                                                  STATUS   ROLES    AGE    VERSION
gke-my-first-cluster-1-default-pool-fd1ec955-g58w   Ready    <none>   2d1h   v1.29.3-gke.1093000
gke-my-first-cluster-1-default-pool-fd1ec955-sh4v   Ready    <none>   2d1h   v1.29.3-gke.1093000
gke-my-first-cluster-1-default-pool-fd1ec955-wbcn   Ready    <none>   14h    v1.29.3-gke.1093000
```

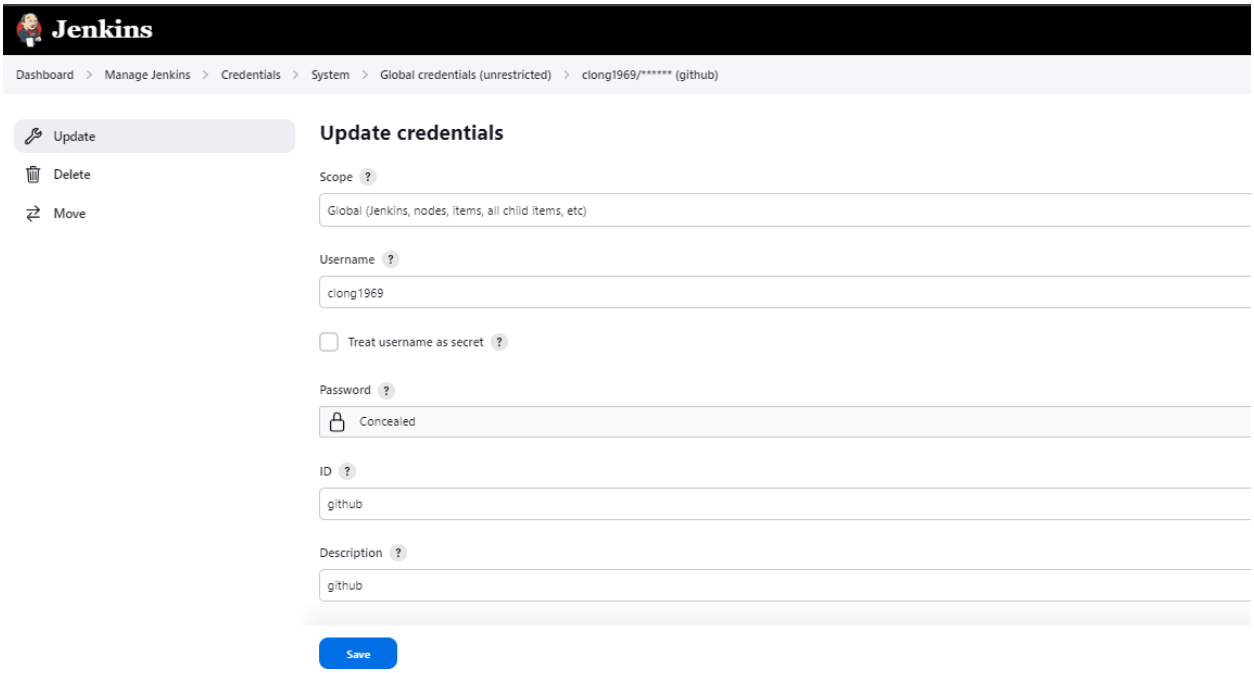
Pods running in Kubernetes: Issued kubectl command to show running pods.

```
charlielong1969@cloudshell:~ (pgpdevops)$ kubectl get pods
NAME                          READY   STATUS    RESTARTS   AGE
abctech-pgp-67b8b99f79-f6czn   1/1     Running   0          27m
abctech-pgp-67b8b99f79-jqb9d   1/1     Running   0          27m
abctech-pgp-67b8b99f79-vdjk7   1/1     Running   0          27m
charlielong1969@cloudshell:~ (pgpdevops)$
```

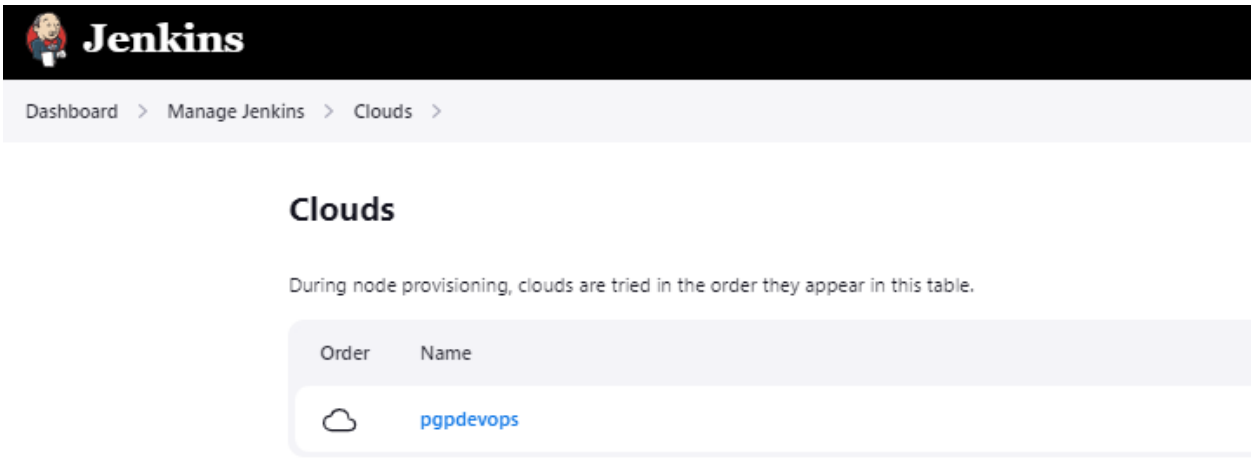
For Jenkins to work with the various cloud services, it required several credentials to be configured:

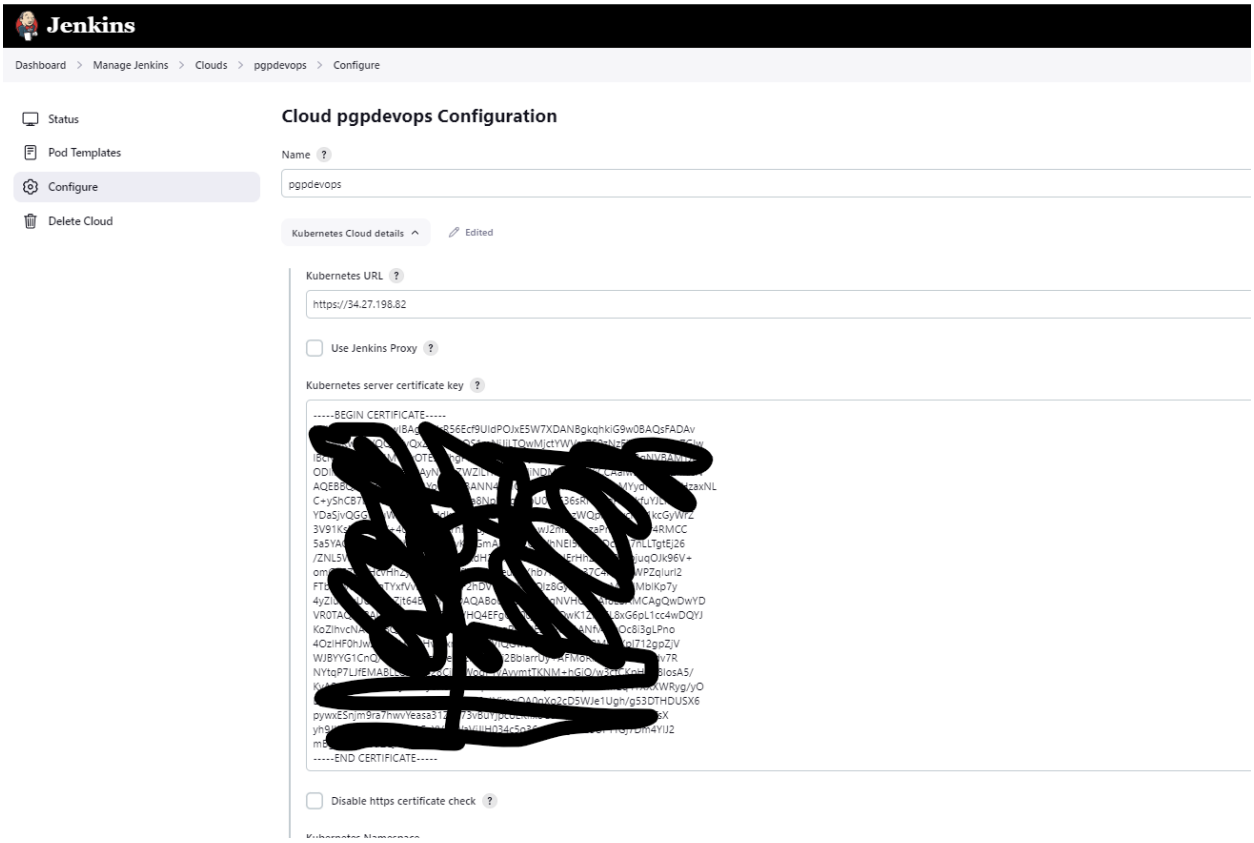


Example with GitHub credentials.



In order to interact with Google Cloud, and cloud provider needed to be configured as well.


Jenkins
Dashboard > Manage Jenkins > Clouds >
Clouds
During node provisioning, clouds are tried in the order they appear in this table.
Order
Name
pgpdevops


Jenkins
Dashboard > Manage Jenkins > Clouds > pgpdevops > Configure
Status
Pod Templates
Configure
Delete Cloud
Cloud pgpdevops Configuration
Name
pgpdevops
Kubernetes Cloud details
Edited
Kubernetes URL
https://34.27.198.82
Use Jenkins Proxy
Kubernetes server certificate key
-----BEGIN CERTIFICATE-----
-----END CERTIFICATE-----
Disable https certificate check