**Project Documentation AdvK8S**

**Project Requirement:**



**Demo microservices codes that we are going to use in this project:**

<https://github.com/Coit-IO/coit-simple-microservice>

<https://github.com/GoogleCloudPlatform/microservices-demo>

**Existing process challenges and possible solutions:**

**- Our developers store the code in GitHub** –

**- We run the application across VMs and all the VMs are self-managed.**

* We are going to run application in K8S and with K8S cluster we don’t need to manage VMs anymore.

- **We use a script which runs every 10 seconds and restarts any applications that are**

**failed.**

* In K8S cluster, if there are any issue with application container it will get restarted automatically. We can also use cloud monitoring service to monitor services health status

**- If any extra servers are needed, we use terraform to code the infra and apply the**

**changes manually.**

* We can use Node auto-provisioning\Vertical Pod Autoscaling\Horizontal Pod Autoscaling services in K8S cluster to fulfil the same.

**- We use our monitoring tool, along with some cloud integration, to scale the servers,**

**based on monitoring matrices.**

* We can use HPA resource and include CPU & Memory matrices in manifest file so that auto scaling at PODs level will be done automatically.

**- To set up a newly provisioned server, we use Ansible to apply our application**

**configurations to the new server.**

* We can use HPA for provisioning new PODs automatically and use Config Map & Patching concepts in K8S to apply application level configurations automatically.

**- To make sure that people get only the permission they need, we have a team who**

**ensures access to different systems to people. Revoking / Adding access is done by**

**that team. And this process is hectic**.

* We can use RBAC service(Cluster role, service account and Role bindings) in GKE to achieve the same.

**- To manage stateful applications, we run them on a set of dedicated servers, and we**

**create disks in the cloud provider, mount the disks and then its used by the**

**application.**

* We can use PVC service in GKE to achieve the same.

**- To provide configurations to different servers, we use ansible and managing ansible**

**is a huge responsibility**

* we can use Config Map and Secret service in K8S to manage configurations automatically.

**Project Plan:**

* Let’s develop all manifest files that we require to deploy microservice based application into K8S.
* We need to ensure that our application is deployed into K8S by using almost all Kubernetes features ( Deployment, ConfigMaps,Secrets,PVC Service,HPA,Ingress,Certificates,Liveness&Readyness,ClusterRoles,service account &Rolebindings)
* Let’s us use Kustomize configuration management tool to create all our resources in K8S.
* Let’s also ensure to we follow the proper folder structure and place manifest & other project files in respective folders so that everyone can understand them easily.
* Once our manifest files and other codes are ready in local working directory, we can push them into remote GitHub repository from local Git client.
* We can manually deploy application (manifest files) to K8S from local machine CLI with kubectl client or we can automate application deployment thru Jenkins CI/CD pipeline tool. CI/CD deployment is preferred one.
* Lets create a managed K8S cluster in GKE with min 3 worker nodes.
* Install necessary tools in local machine and Jenkins server to deploy application in GKE.
* Ensure we follow best practices when deploying application into GKE Cluster.

**Project Flow:**

**Day 1 Tasks:**

Discuss about project requirement and implementation plan

**Day 2 Tasks:**

Develop manifest files in local working directory

Upload them into github repository

**Day 3 Tasks:**

GKE cluster setup

Jenkins server setup

Make connection b/w Jenkins to GKE

**Day 4 Task:**

Create CI/CD pipeline to deploy manifest files to GKE

Validate application and configurations.

Deliver the project.

Document

**Day 5 Task:**

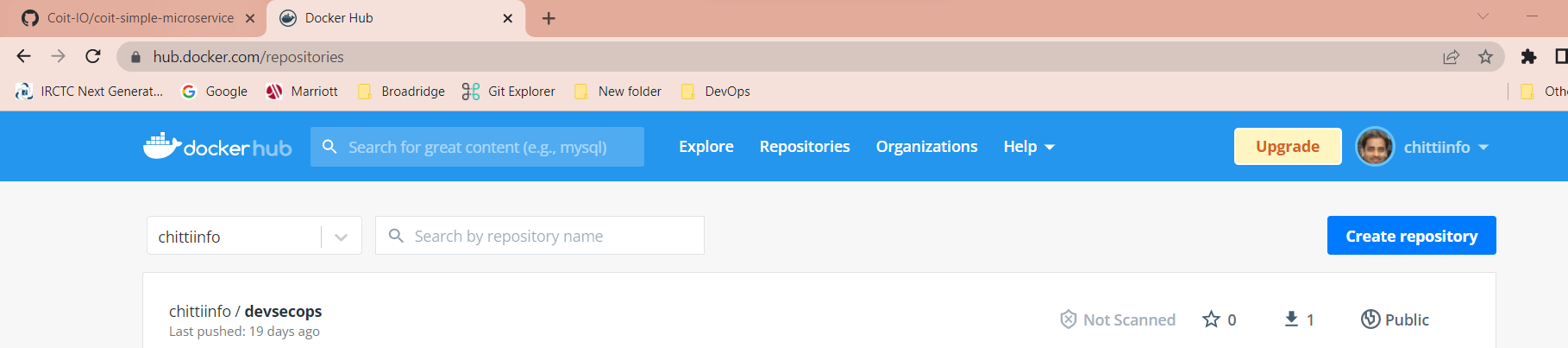
Discuss all the achievements and challenges that we have in this project.

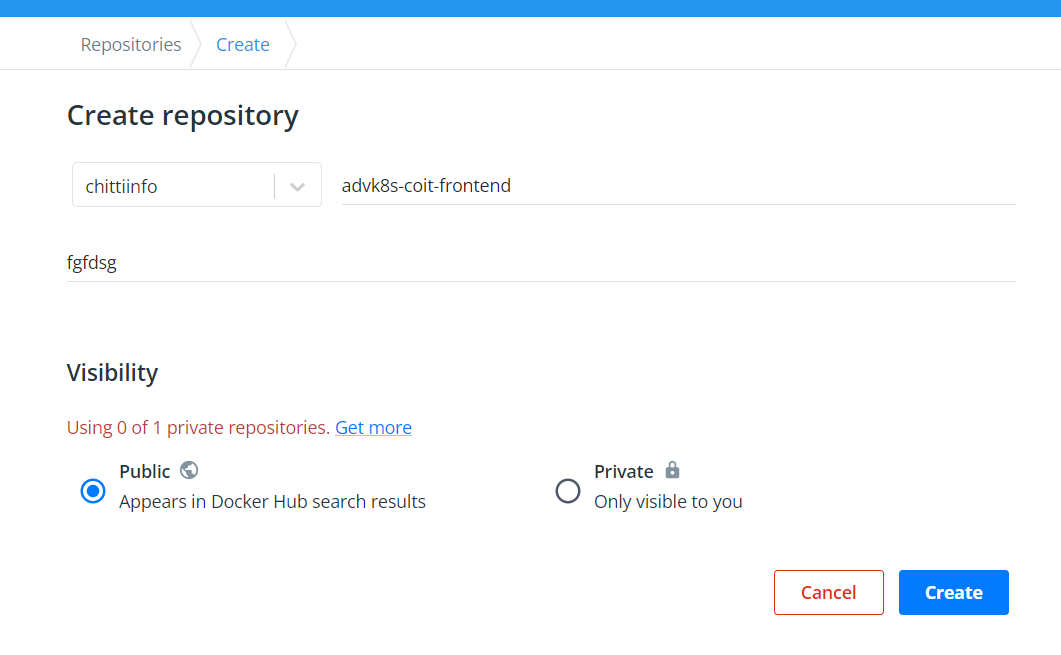
Close the project

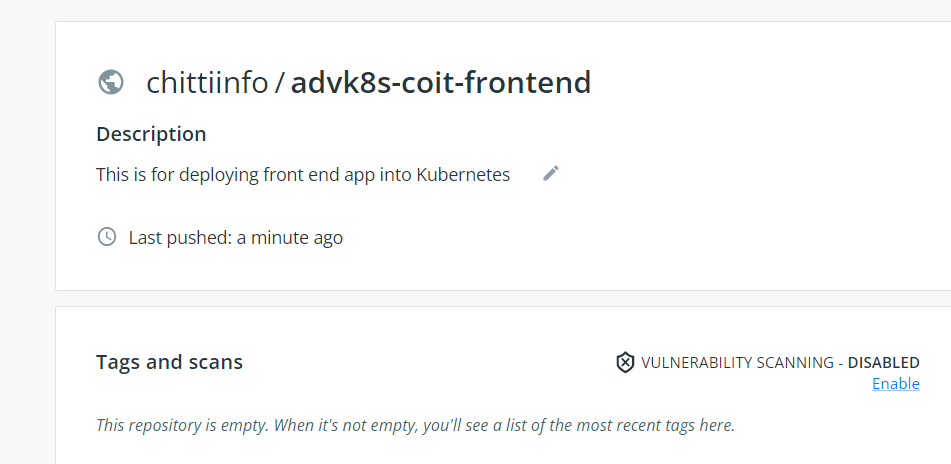
**Docker Work:**

Login into your docker hub account

Create a repository







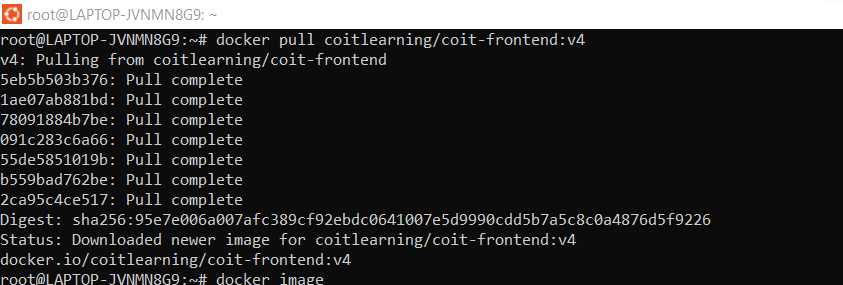
Once docker repository is created go to your local machine where docker client is installed and pull,tag then push the image to your docker repository.

docker images – to list the images

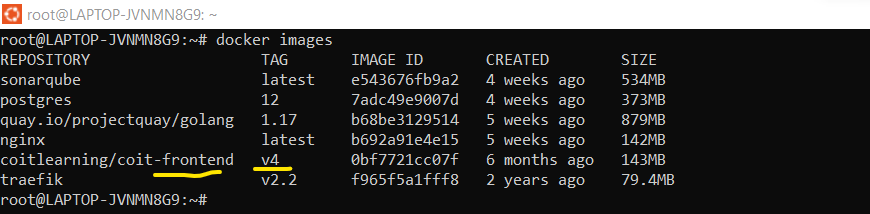
docker image prune -a – to delete unused imags

docker container prune - to delete unused continer

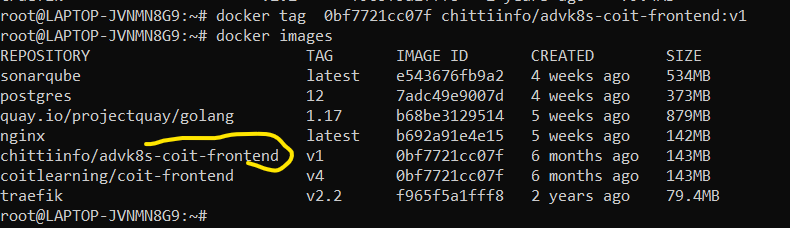
docker pull coitlearning/coit-frontend:v4 - to pull images from docker hub



docker images



docker tag 0bf7721cc07f chittiinfo/advk8s-coit-frontend:v1



docker push chittiinfo/advk8s-coit-frontend:v1

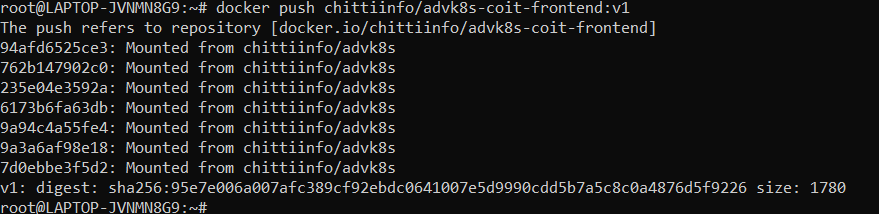
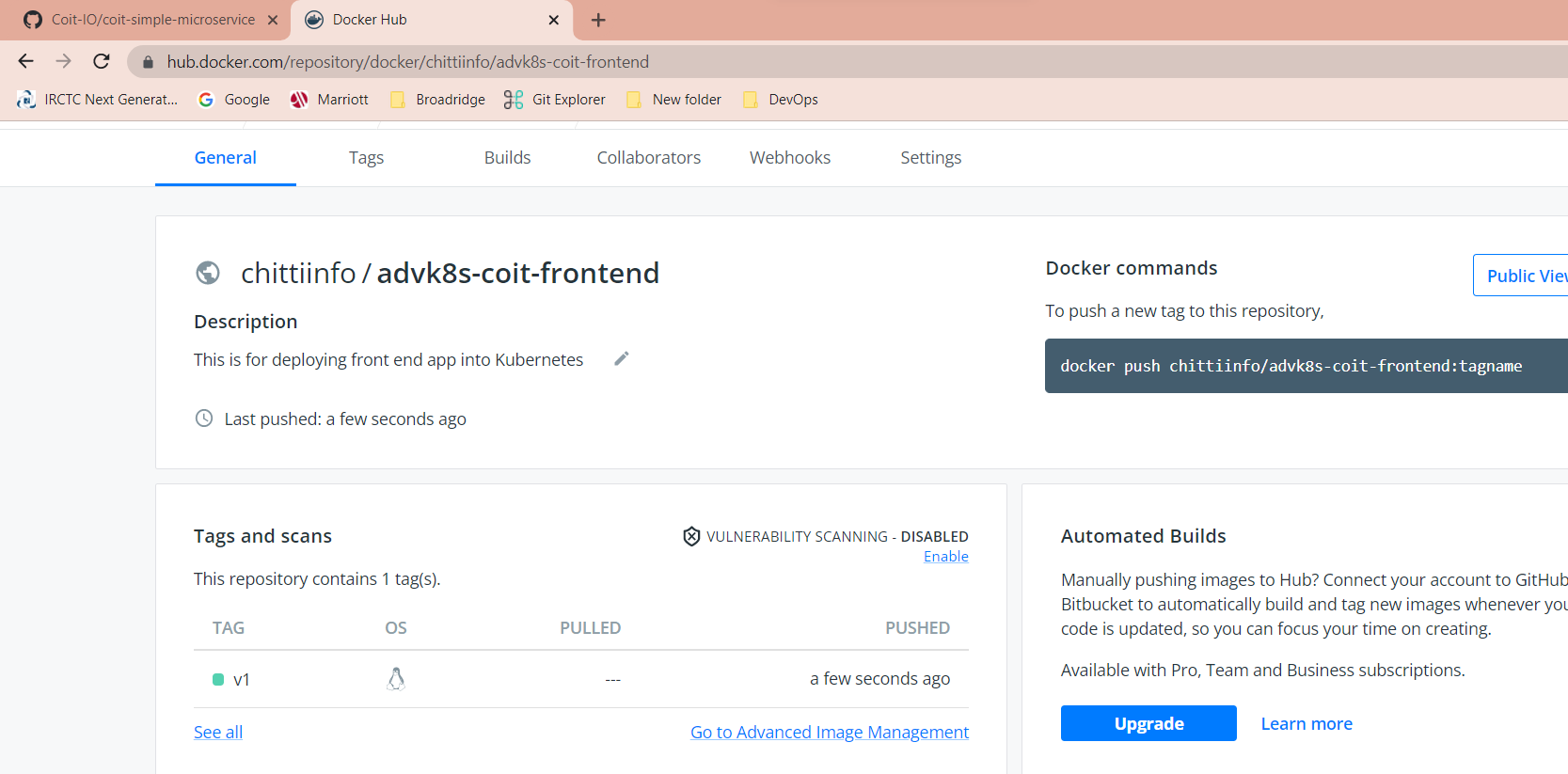
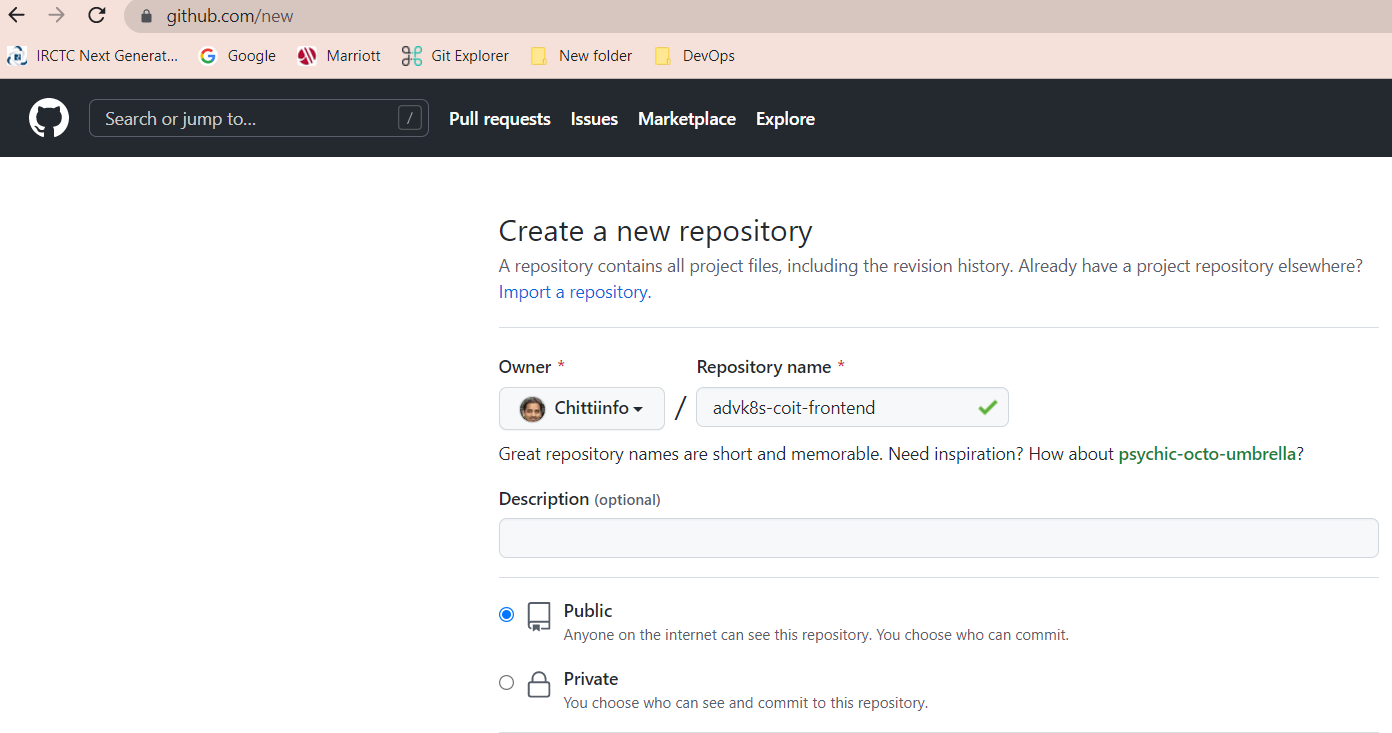


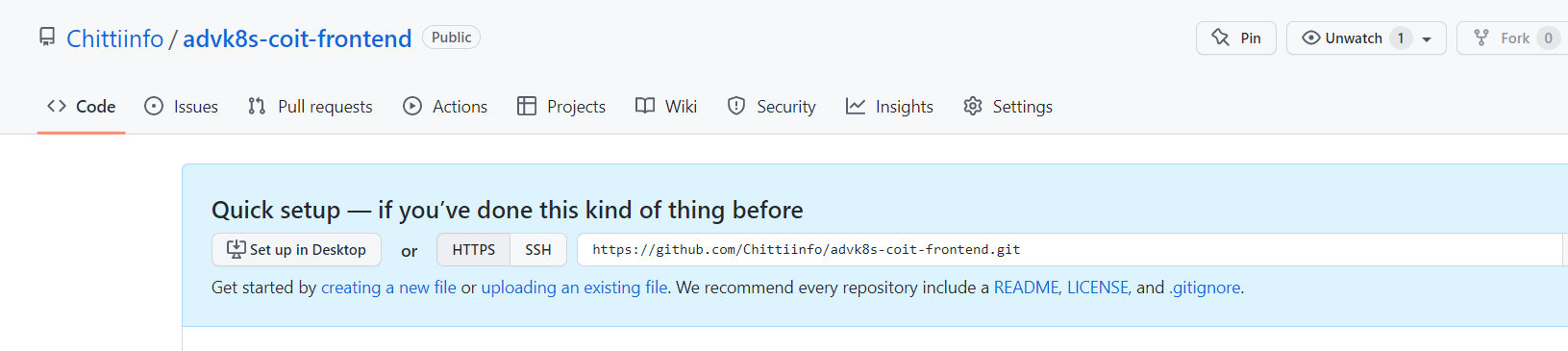
Image is stored in docker hub now.



**GitHub Work:**

Create a github repository that we are going to use for storing code files. Below I created GitHub repository and the URL is <https://github.com/Chittiinfo/advk8s-coit-frontend.git>





**Git Work:**

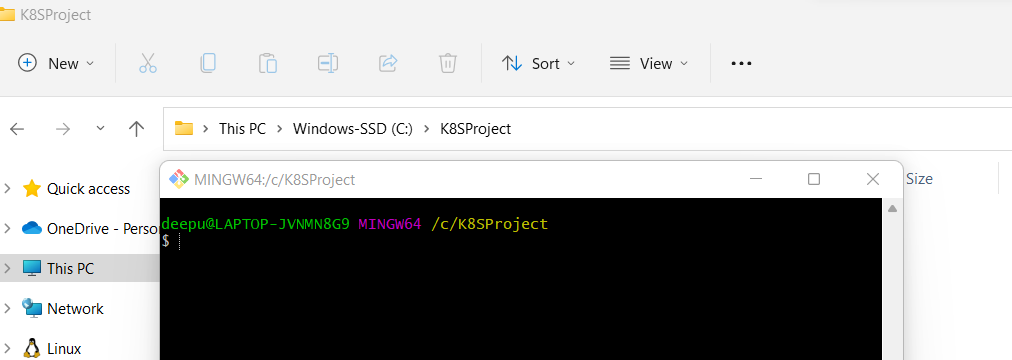
**NOTE:** You can pull the orginal code from<https://github.com/Coit-IO/coit-simple-microservice> and make the necessary modifications or else you can download code from my github repository that’s already modified as per this project requirement

<https://github.com/Chittiinfo/advk8s-coit-frontend/tree/dev>

Using gitbash client local to work on local repository changes. Lets follow Git flow branching strategy in this project. We will have two branches(main & dev) and dev will have actual code.

**Git Setup: Follow below link to install Git client on your machine.**

[Git - Downloads (git-scm.com)](https://git-scm.com/downloads)



git init

git clone <https://github.com/Coit-IO/coit-simple-microservice>

cd coit-simple-microservice/

cd coit-simple-microservice/

git log --- checking previous commits

git branch – Checking to how many branches we have now

git add -A

git commit -am "frist commit"

git branch -D main

git branch -m main

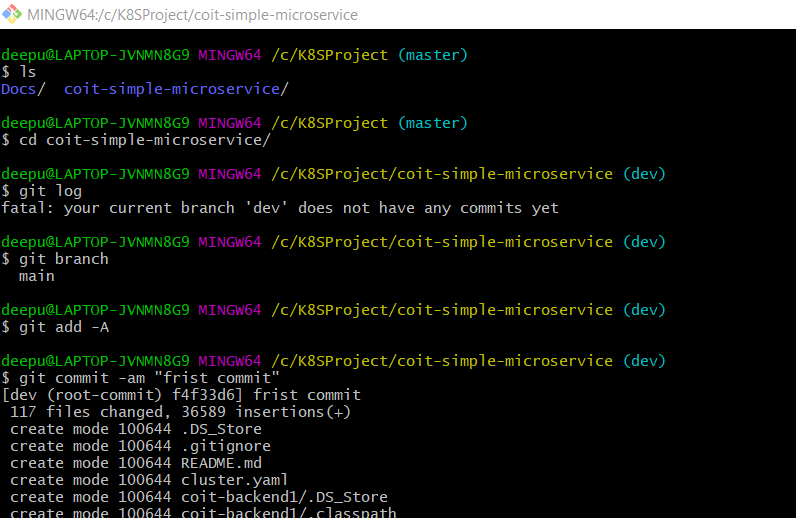
git log

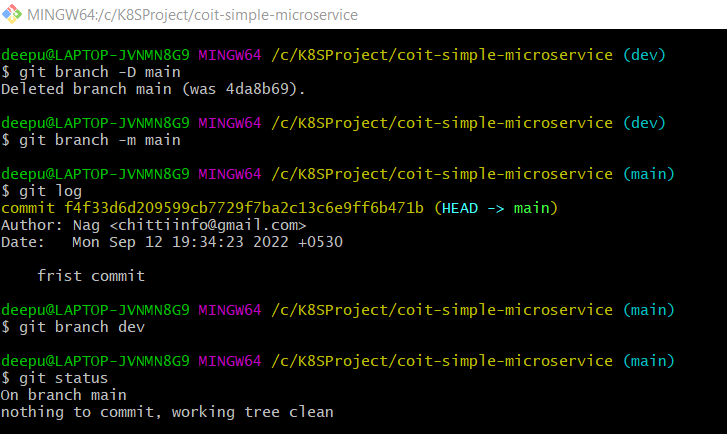
git branch dev

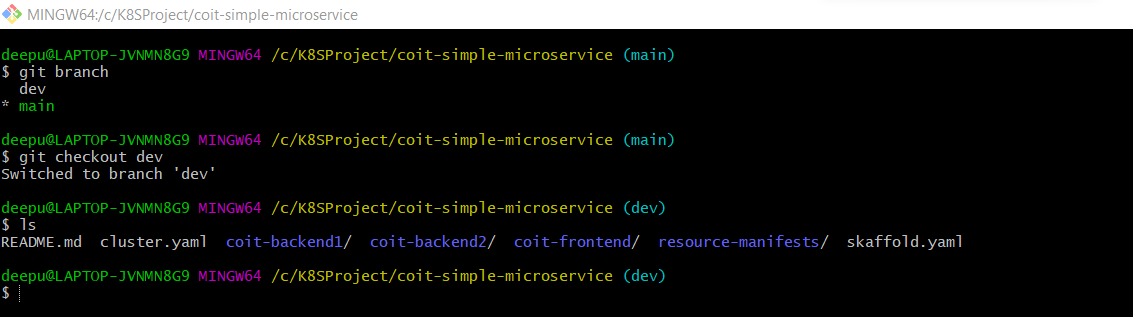
git status

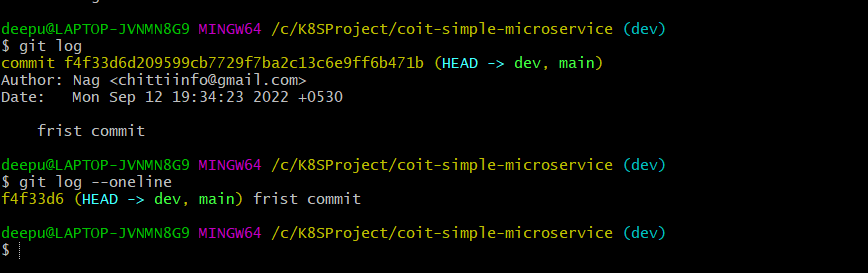
git add .

git commit -m "second commit"

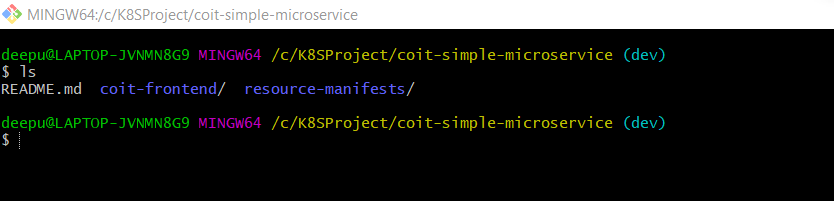






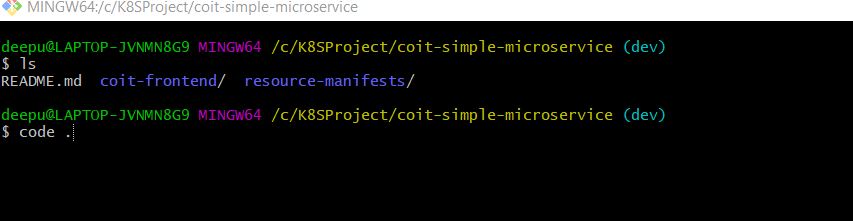


In this project, we just want to deploy front end application into k8s so cleaned up unnecessary files/folders and committed changes in local git repositor

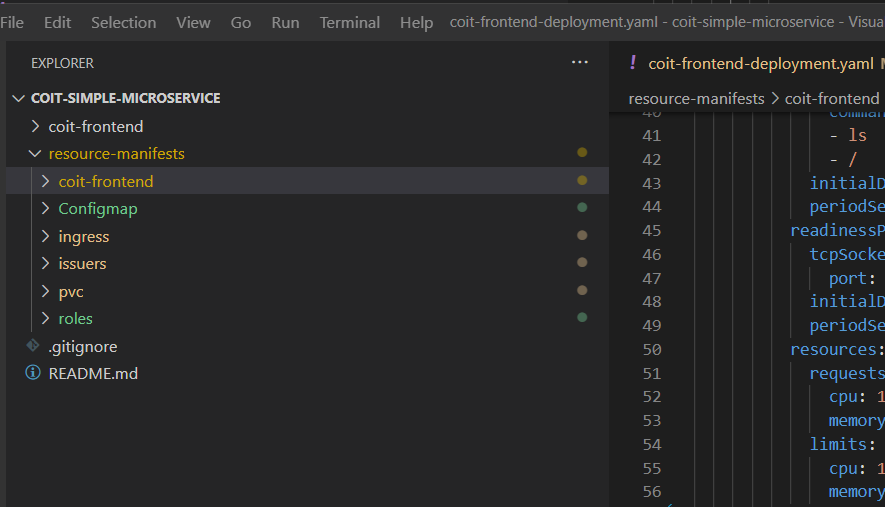


Once downloaded the code from <https://github.com/Coit-IO/coit-simple-microservice> repositor delete the unnecessary files and make sure we have only files which are related to Coit Front End application.

Use cod . command to open code directly in VSC.



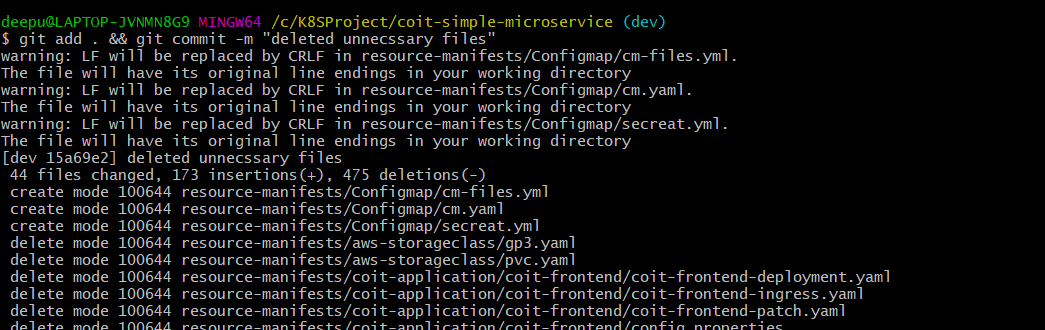
Ensure all the manifest files are modified as per our requirement and place them in respective folders.



Once all manifest files are ready then we can commit the changes into local repository then push them into Github remote repository.

**Committing the changes to local repository:**

git add . && git commit -m "deleted unnecssary files"



Push the changes to GitHub remote repository:

**git login**

git remote show origin - To check what remote repository url is currently set

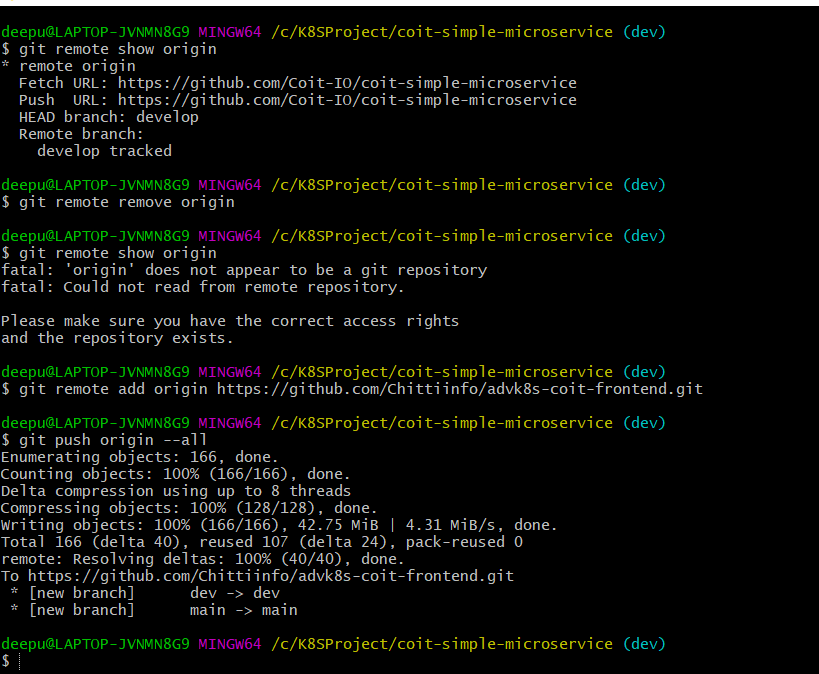
git remote remove origin

**git config --global user.name "Nag"**

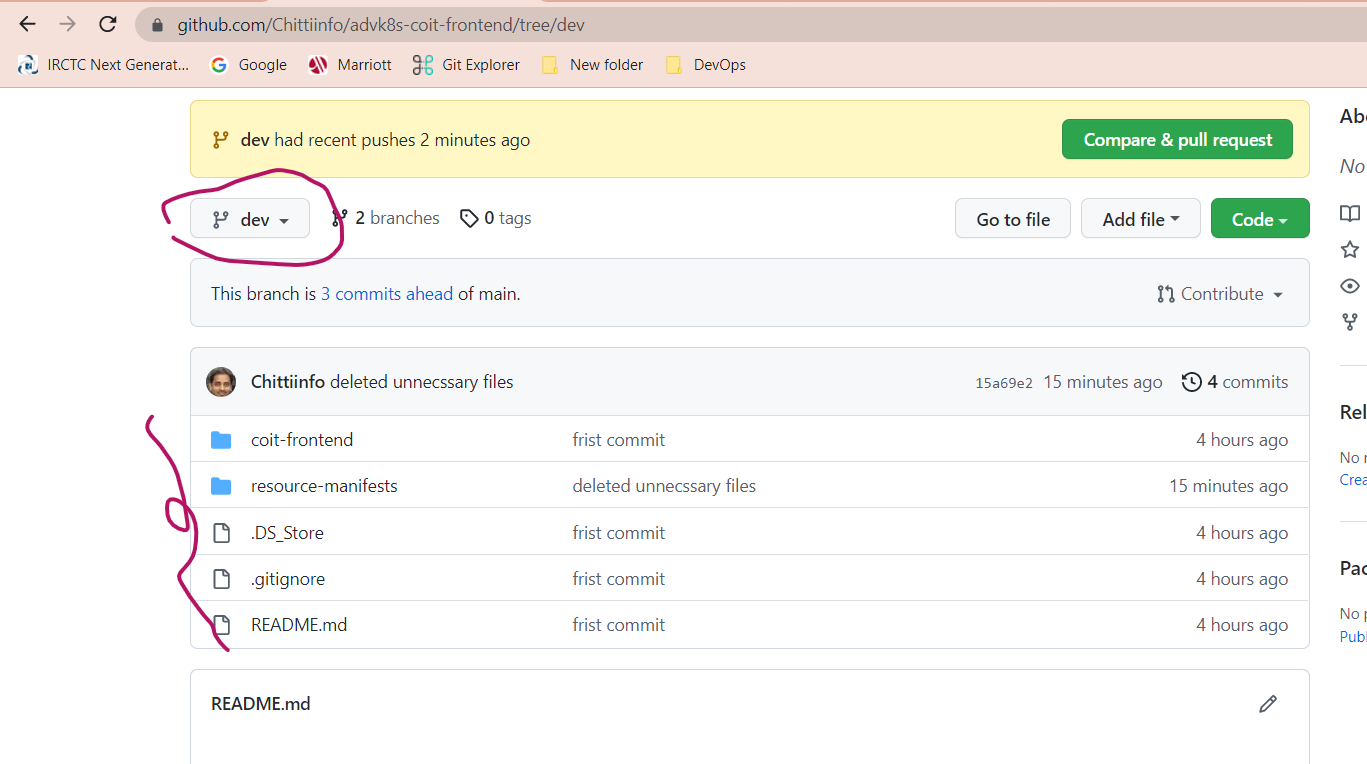
**git config --global user.email "chittiinfo@gmail.com"**

**git remote add origin** <https://github.com/Chittiinfo/advk8s-coit-frontend.git>

**git push origin –all - To push the code to remote repository.**

****

Once code is successfully pushed into Github repository we can check it from github console.

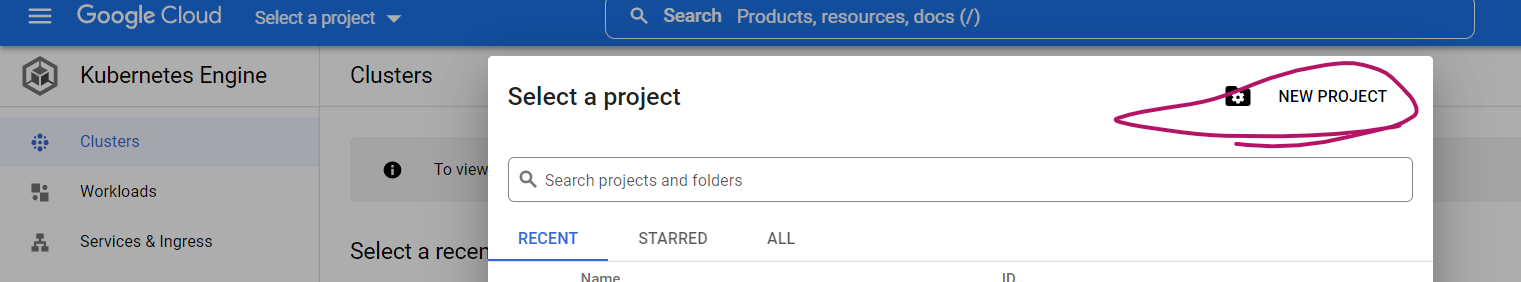


**GKE Work:**

Login into gcloud console

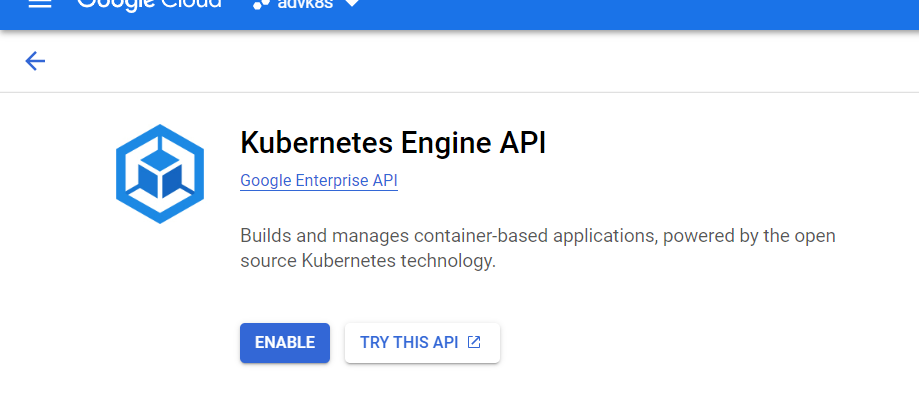
<https://console.cloud.google.com/kubernetes/list/overview?project=tidal-copilot-359606>

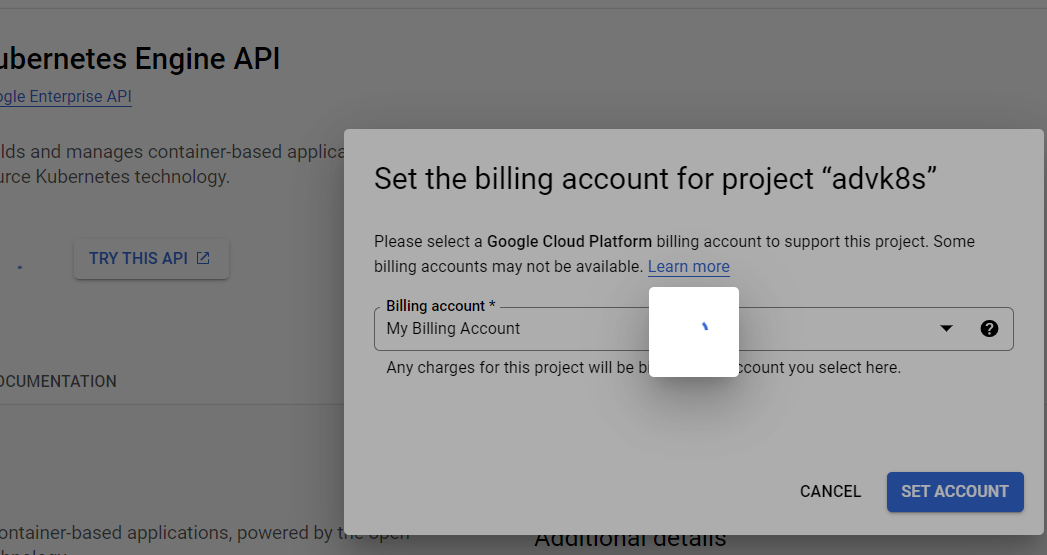
Create a Project with any name



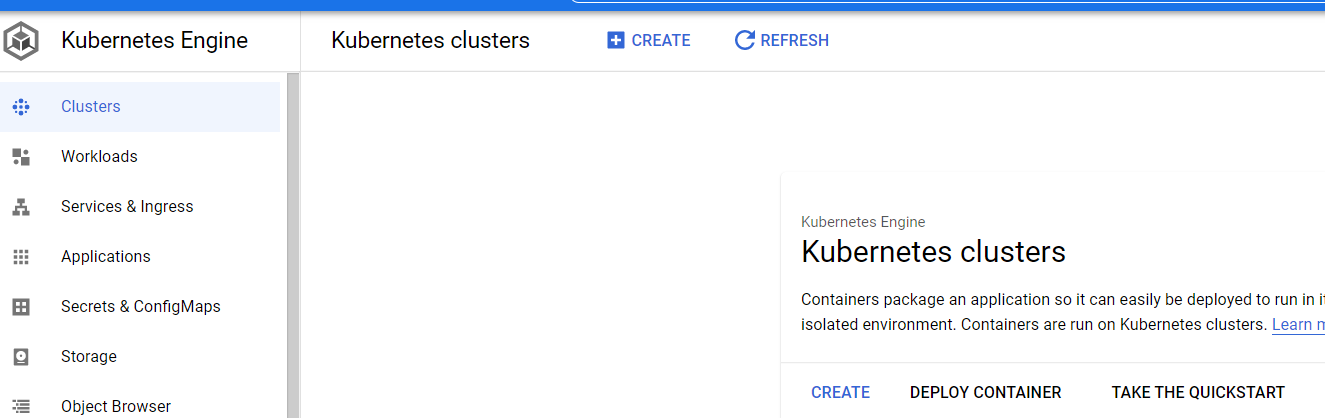


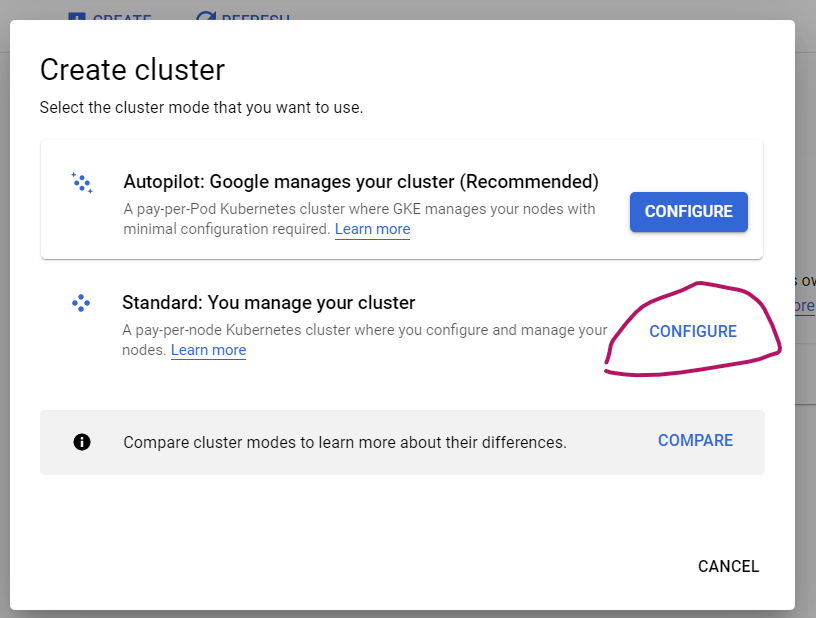
Enable Kubernetes engine api

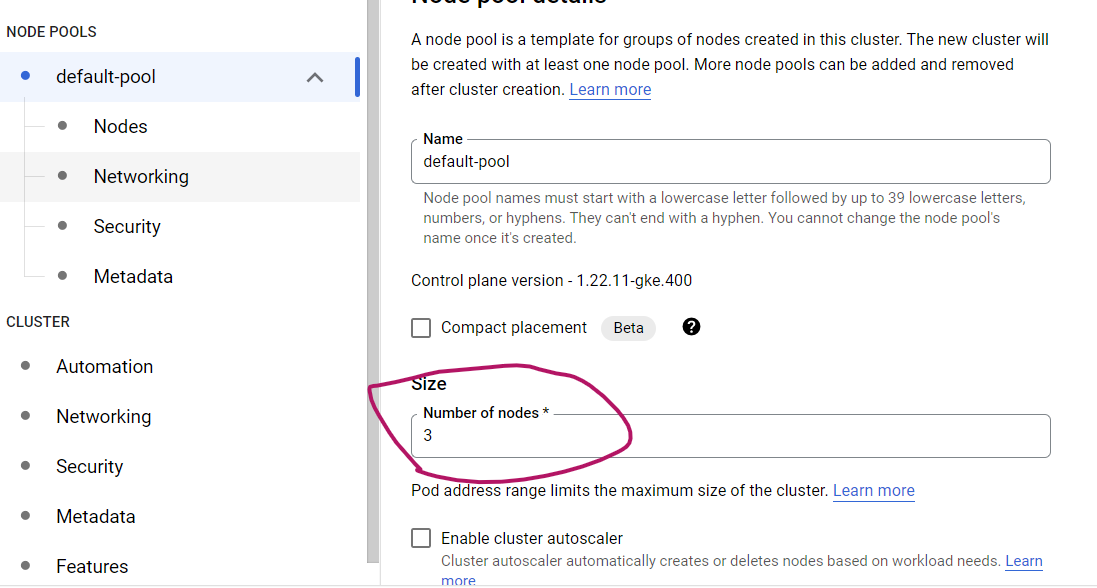


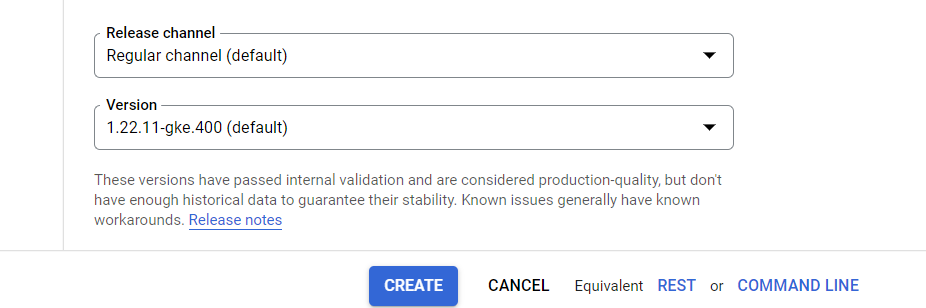


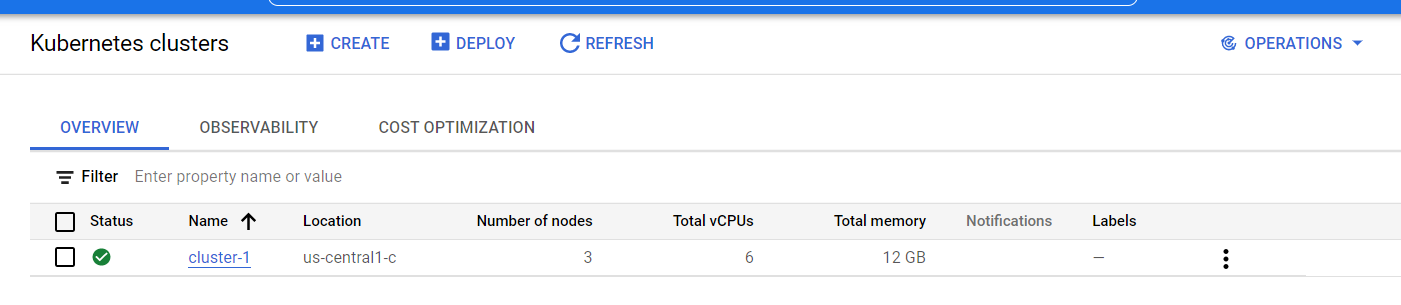
Click on Create GKE cluster



****

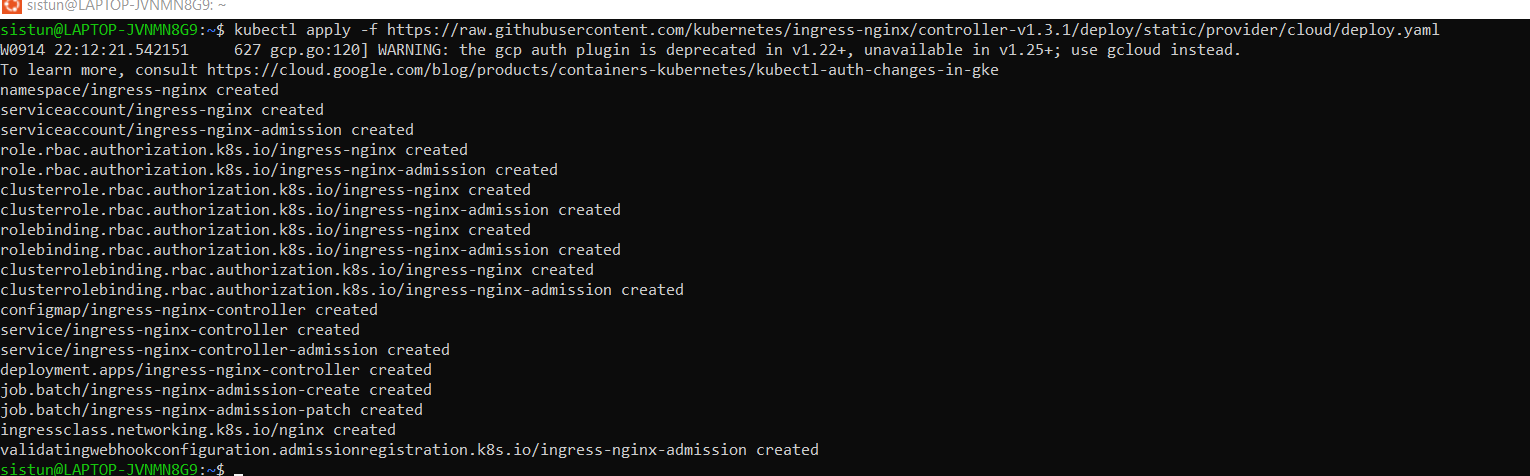
****

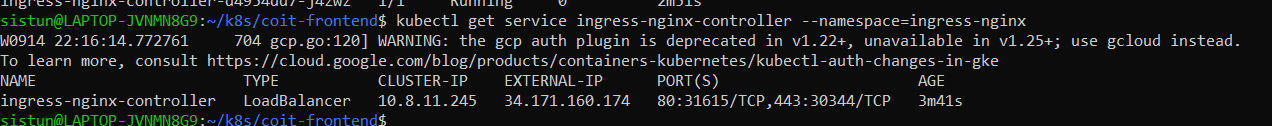
****

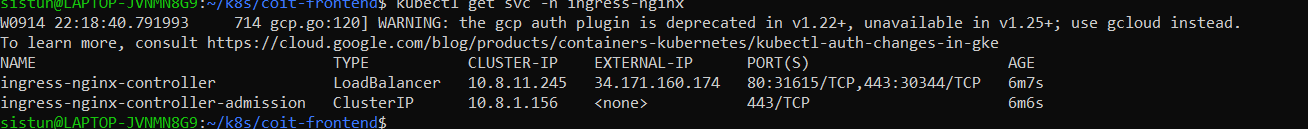
****

**Install ingress controller**

kubectl apply -f <https://raw.githubusercontent.com/kubernetes/ingress-nginx/controller-v1.3.1/deploy/static/provider/cloud/deploy.yaml>

****

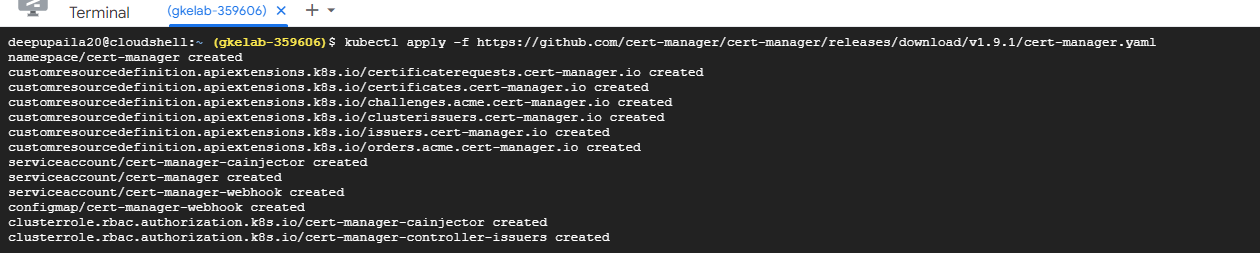
****

****

URL:[**https://kubernetes.github.io/ingress-nginx/deploy/**](https://kubernetes.github.io/ingress-nginx/deploy/)

**Install Custom Resource Definitions with below command once GKE cluster is up**

kubectl apply -f <https://github.com/cert-manager/cert-manager/releases/download/v1.9.1/cert-manager.yaml>

****

URL: <https://cert-manager.io/docs/installation/kubectl/>

**GoDaddy Work:**

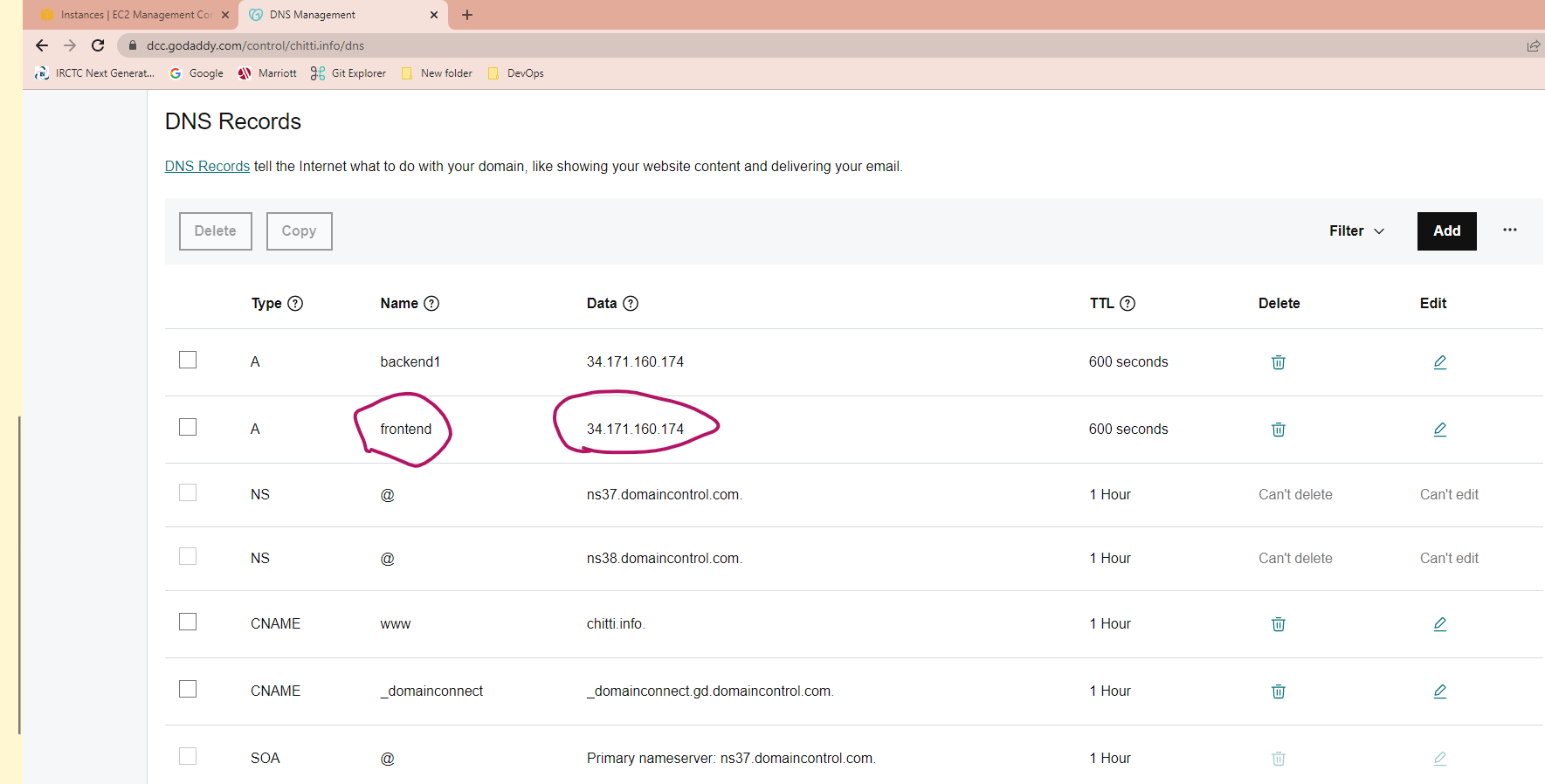
Once application deployment is successful, you can get ingress controller external IP from GKE cluster with below command.

kubectl get service ingress-nginx-controller --namespace=ingress-nginx



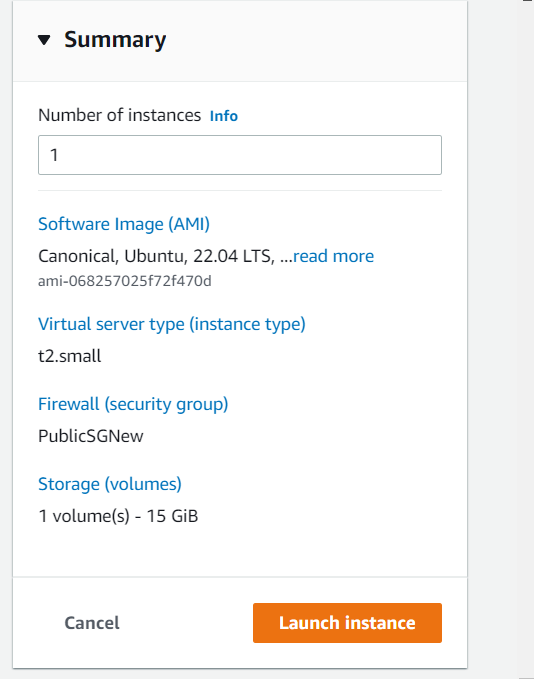
If your website domain is hosted in GoDaddy then create a DNS A record for this site (frontend.chitti.info) and point it to ingress controller external IP (34.171.160.174) in GoDaddy

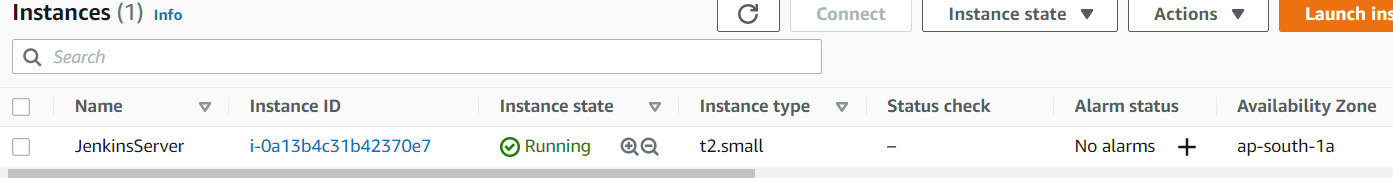
(<https://dcc.godaddy.com/control/chitti.info/dns>)

****

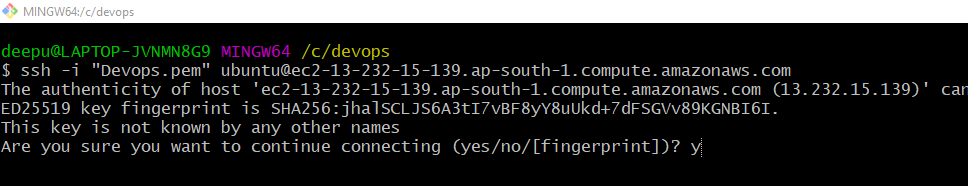
**Jenkins work:**

**Create an instance in AWS (Use instance type Medium or Small)**

****

****

**Connect to Instance**

****

**Install Java and Jenkins**

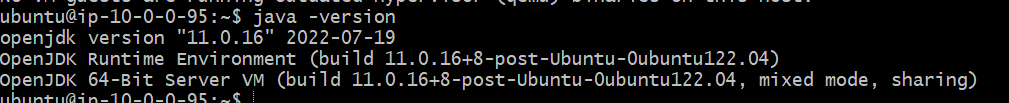
[**https://www.jenkins.io/doc/book/installing/linux/**](https://www.jenkins.io/doc/book/installing/linux/)

**Java Commands:**

$ sudo apt update

$ sudo apt install openjdk-11-jre

$ java -version

****

**Jenkins commands:**

curl -fsSL https://pkg.jenkins.io/debian/jenkins.io.key | sudo tee \

/usr/share/keyrings/jenkins-keyring.asc > /dev/null

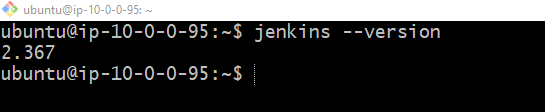
echo deb [signed-by=/usr/share/keyrings/jenkins-keyring.asc] \

https://pkg.jenkins.io/debian binary/ | sudo tee \

/etc/apt/sources.list.d/jenkins.list > /dev/null

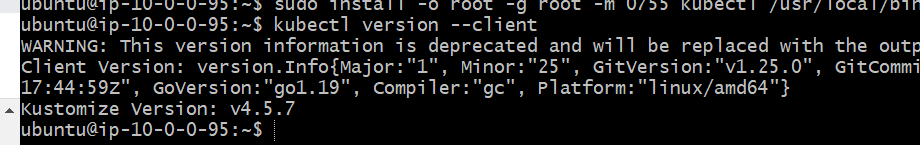
sudo apt-get update

sudo apt-get install Jenkins

****

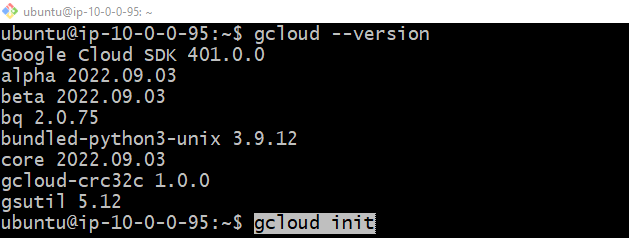
**Install kubectl**

[**https://kubernetes.io/docs/tasks/tools/install-kubectl-linux/#install-kubectl-binary-with-curl-on-linux**](https://kubernetes.io/docs/tasks/tools/install-kubectl-linux/#install-kubectl-binary-with-curl-on-linux)

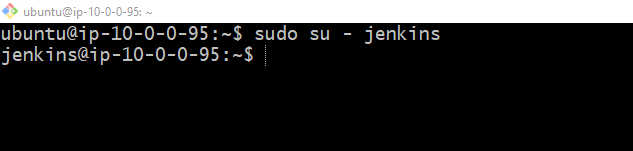
****

**Install Gcloud**

[**https://cloud.google.com/sdk/docs/install**](https://cloud.google.com/sdk/docs/install)

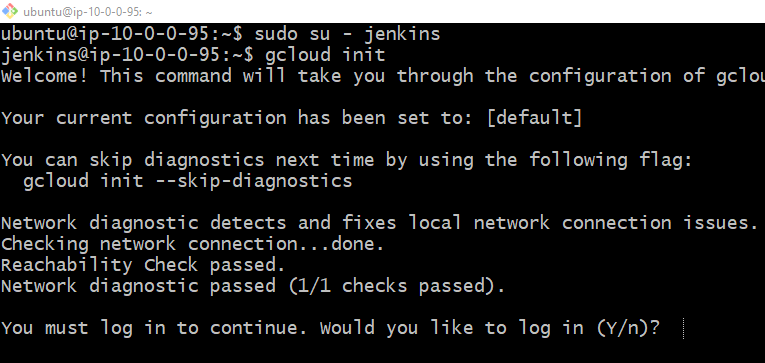
****

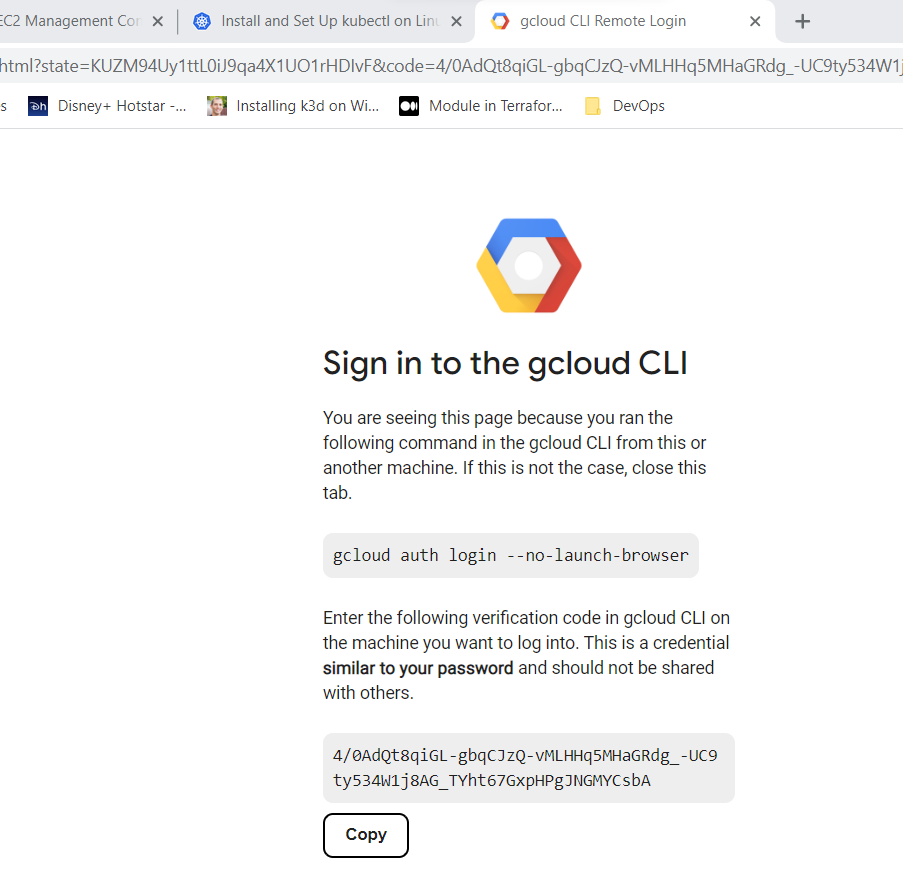
**Switch to Jenkins user and make connections b/w Jenkins and GCloud:**

****

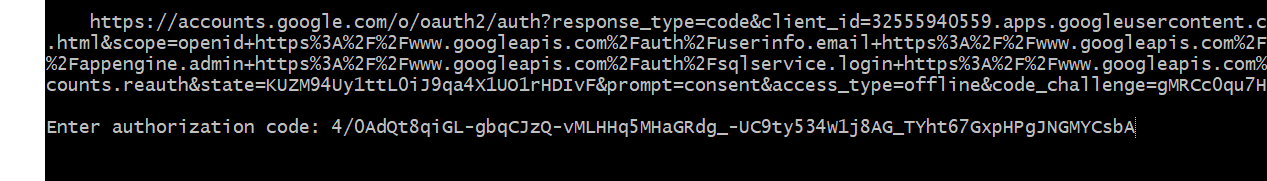
**Make Connection to GCloud:**

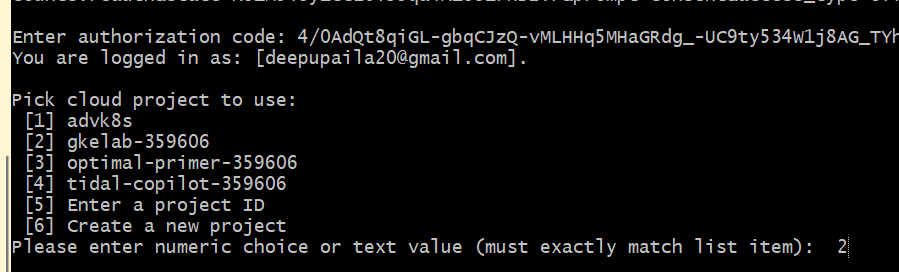
Gcloud init

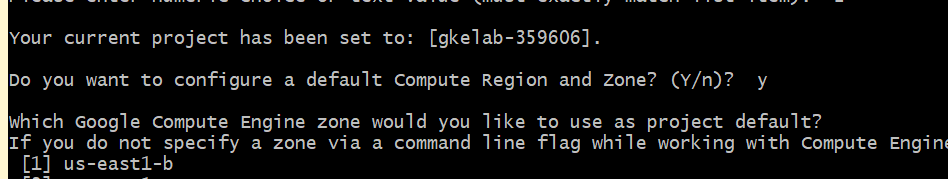
****

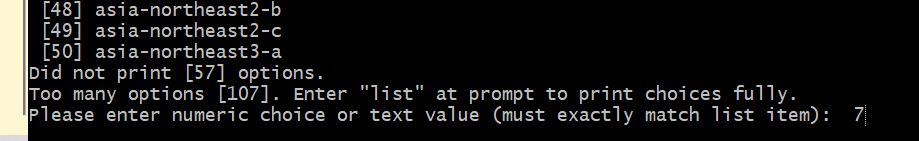
****

Copy the code from here to Jenkins cli

****

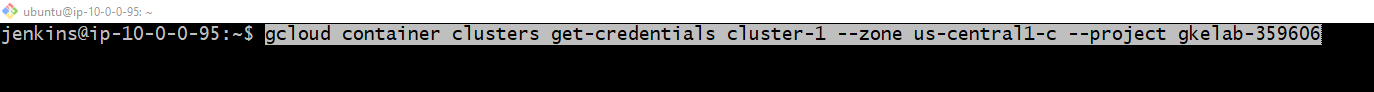




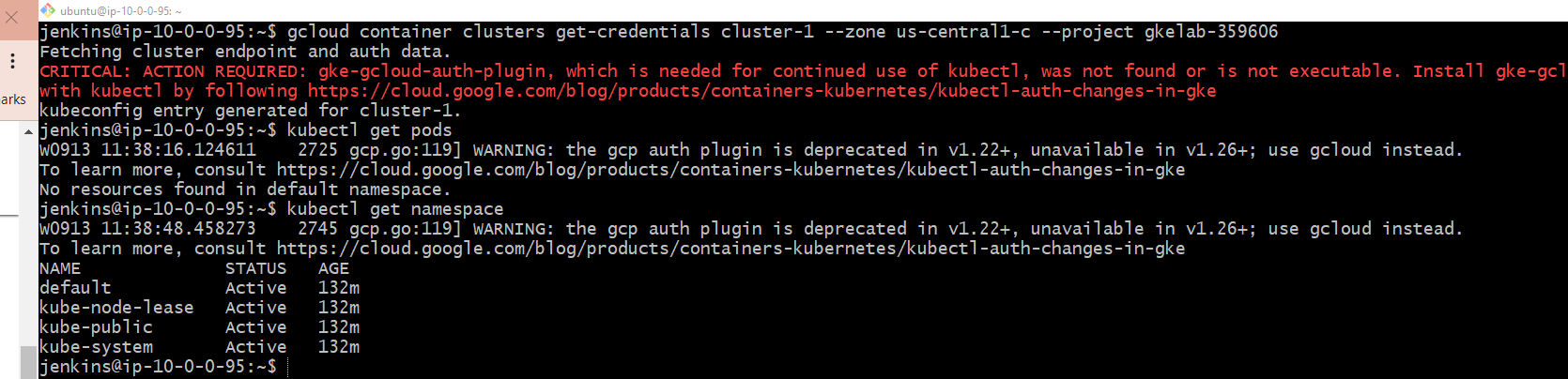


Connect to active GKE cluster with below command.

gcloud container clusters get-credentials cluster-1 --zone us-central1-c --project gkelab-359606

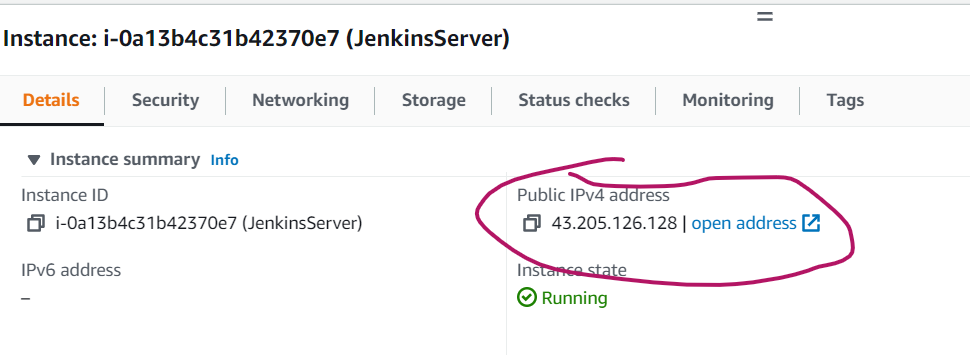


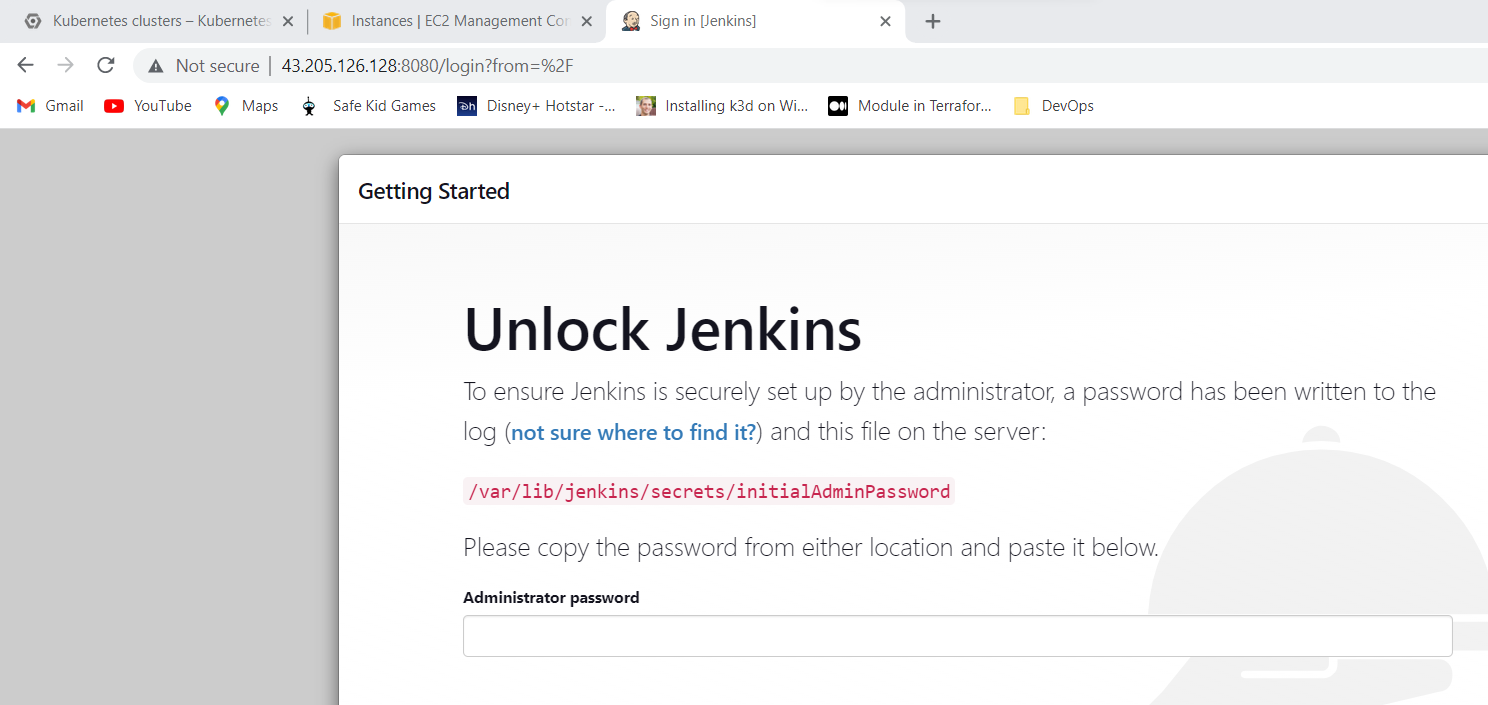
Now my Jenkins server is able to talk with my GKE cluster and I have validated it by running some kubectl commands

****

**Now lets login into Jenkins server GUI and setup Jenkins job.**

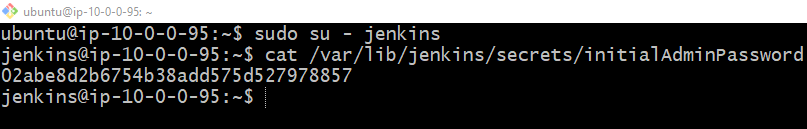
**Take the public IP of Jenkins server and access it from browser with port 8080**

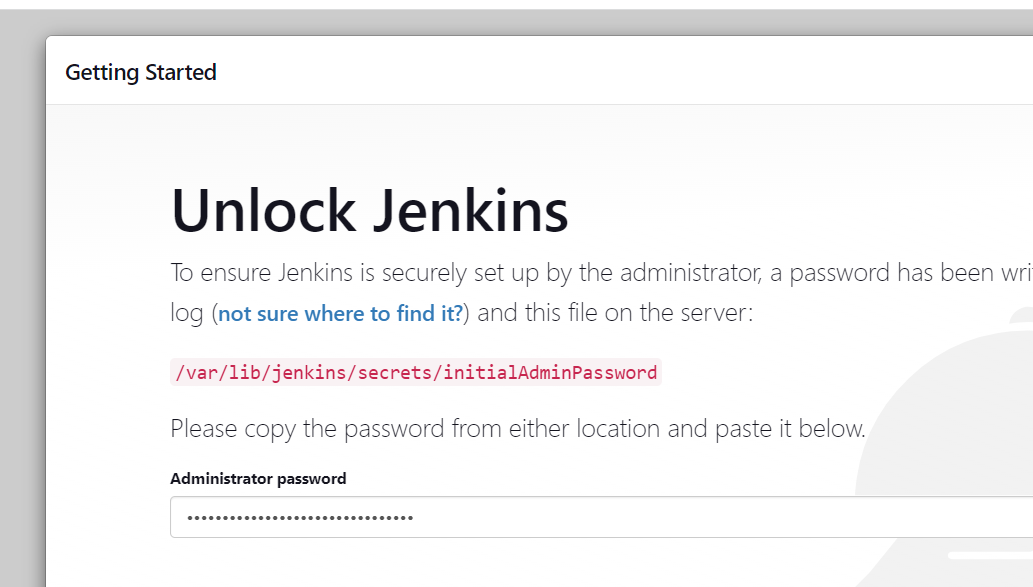
****

****

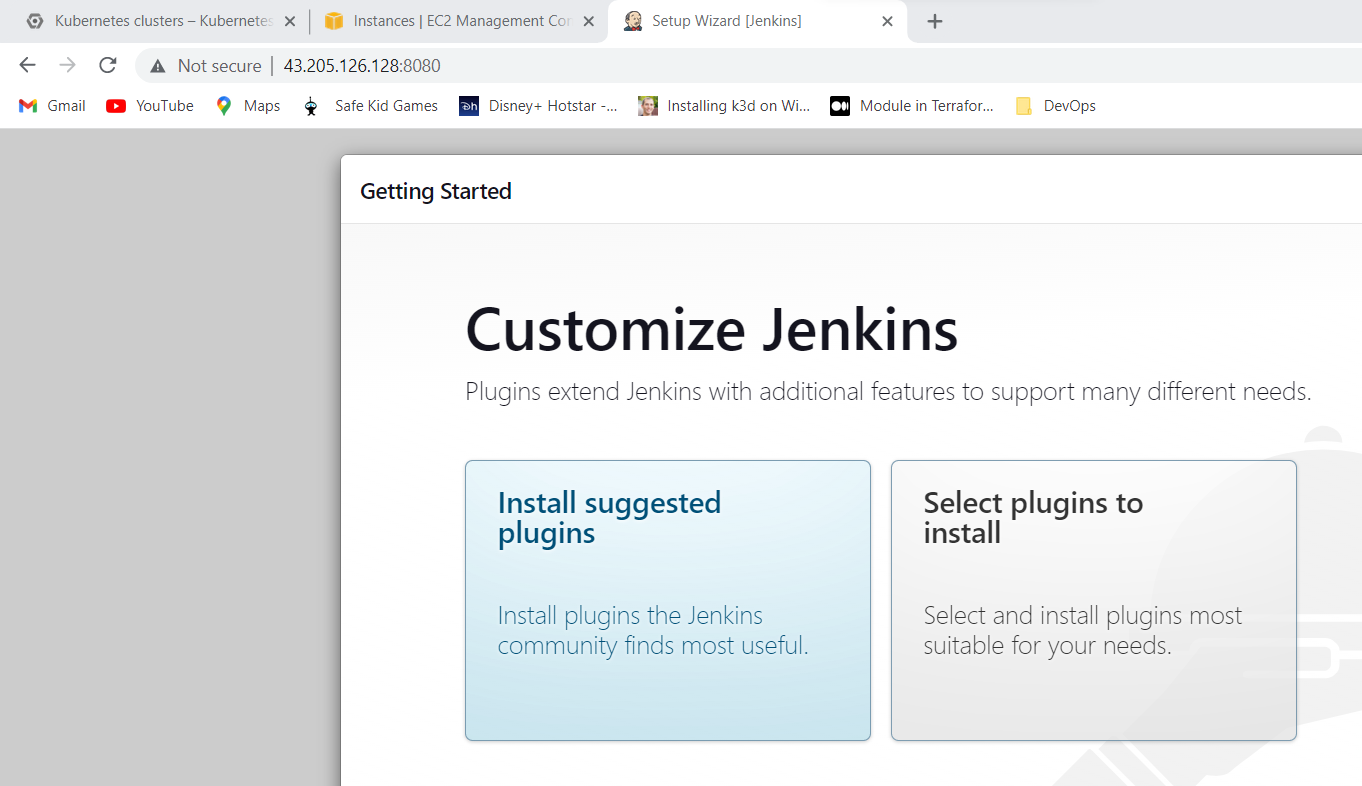
**You can get intial admin password from below file in Jenkins server**

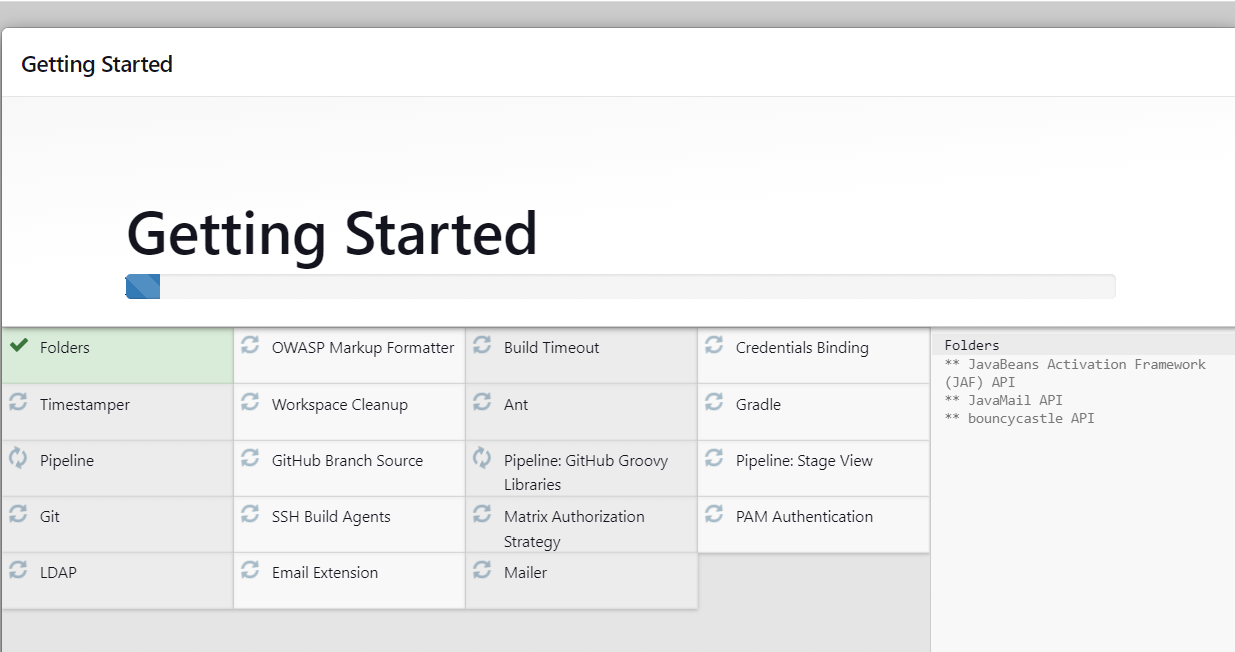
cat /var/lib/jenkins/secrets/initialAdminPassword

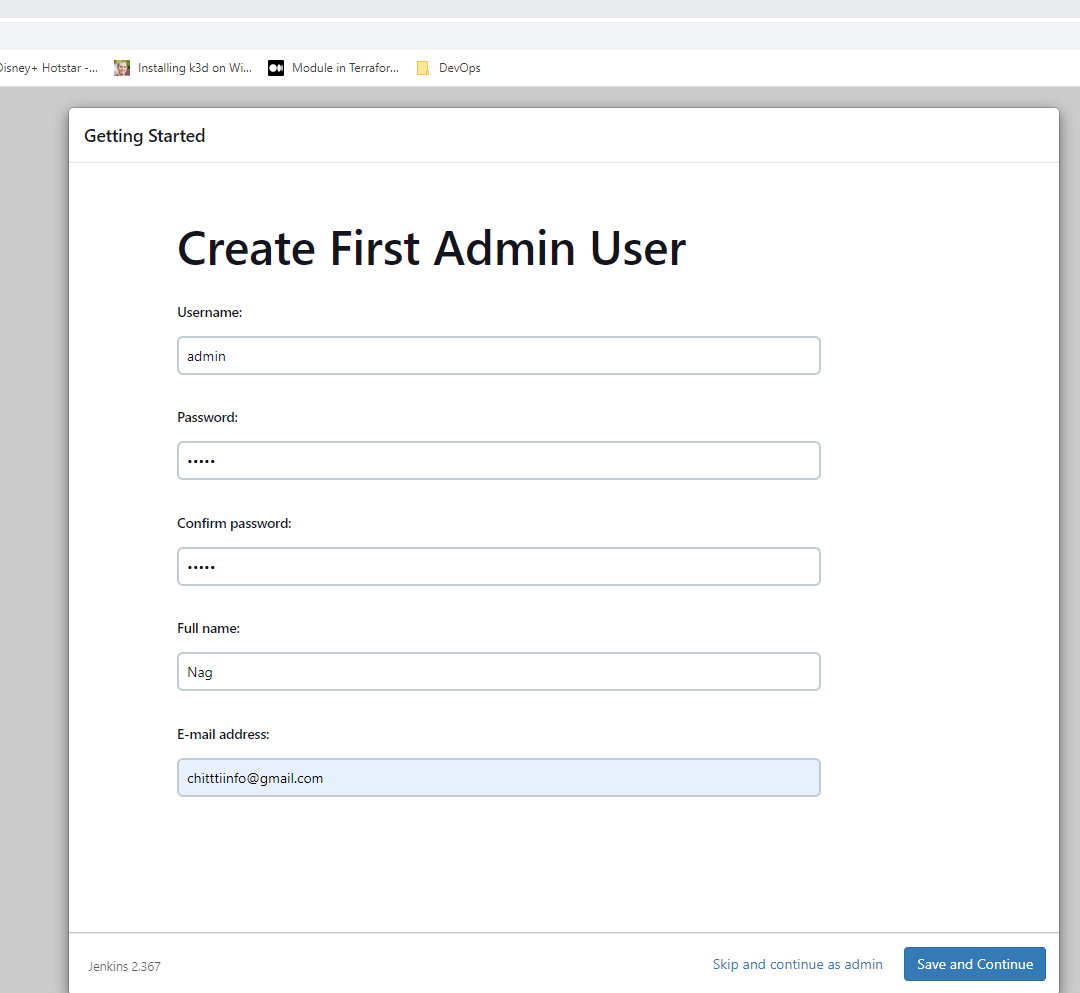
****

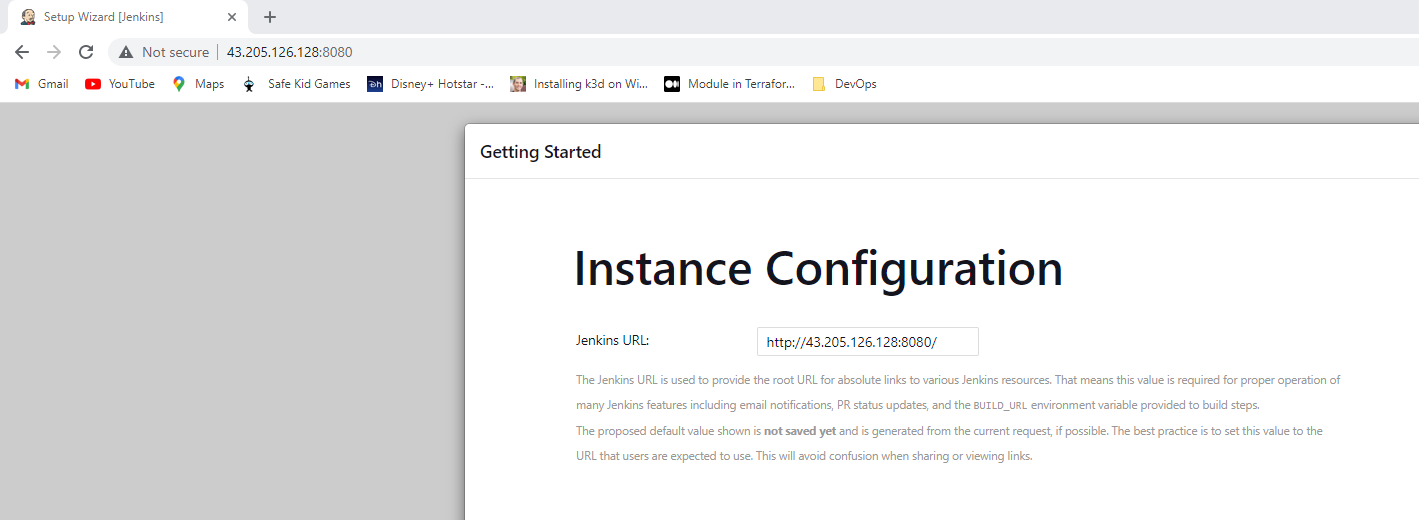
****

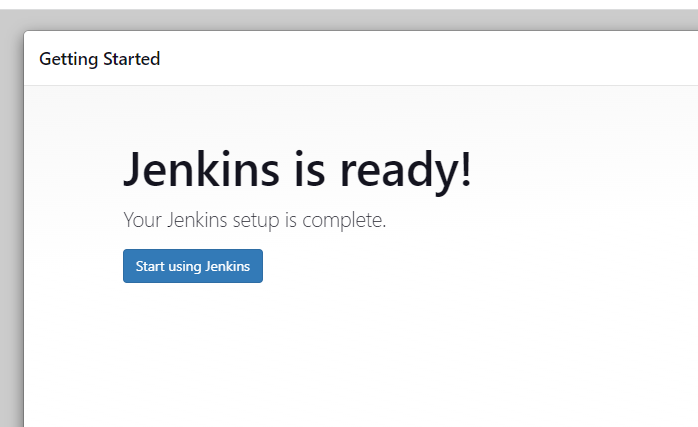
Install suggested plugins

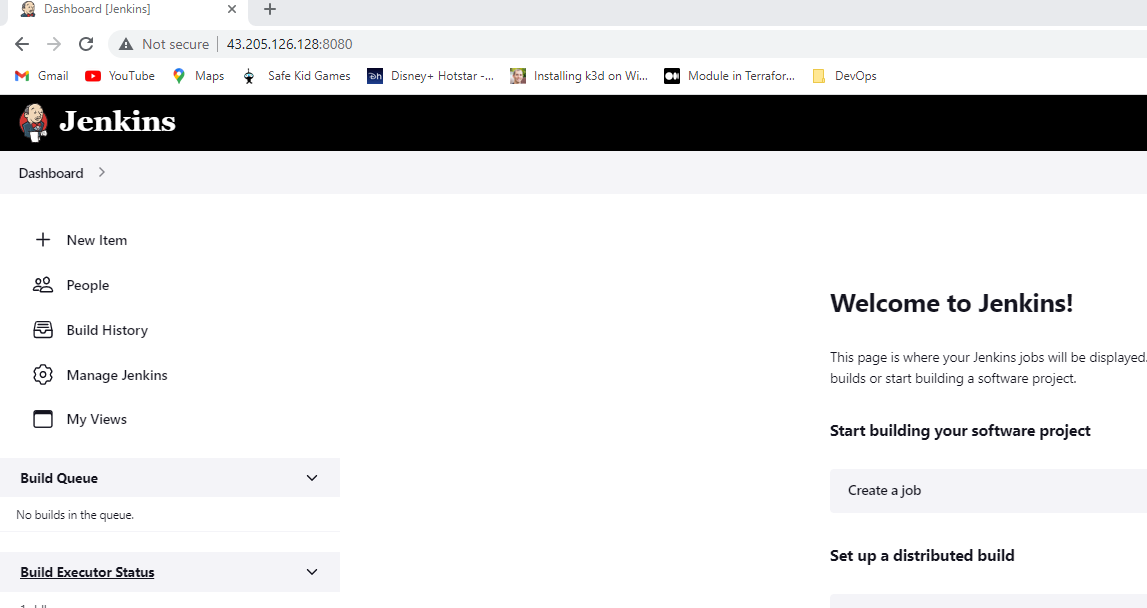
****

****

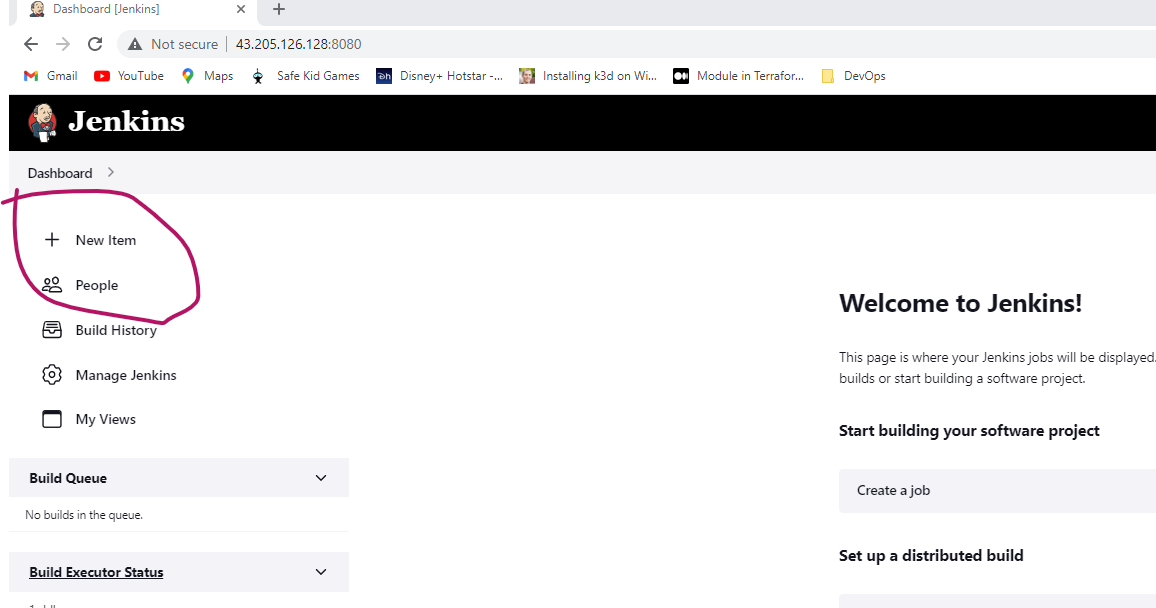
****

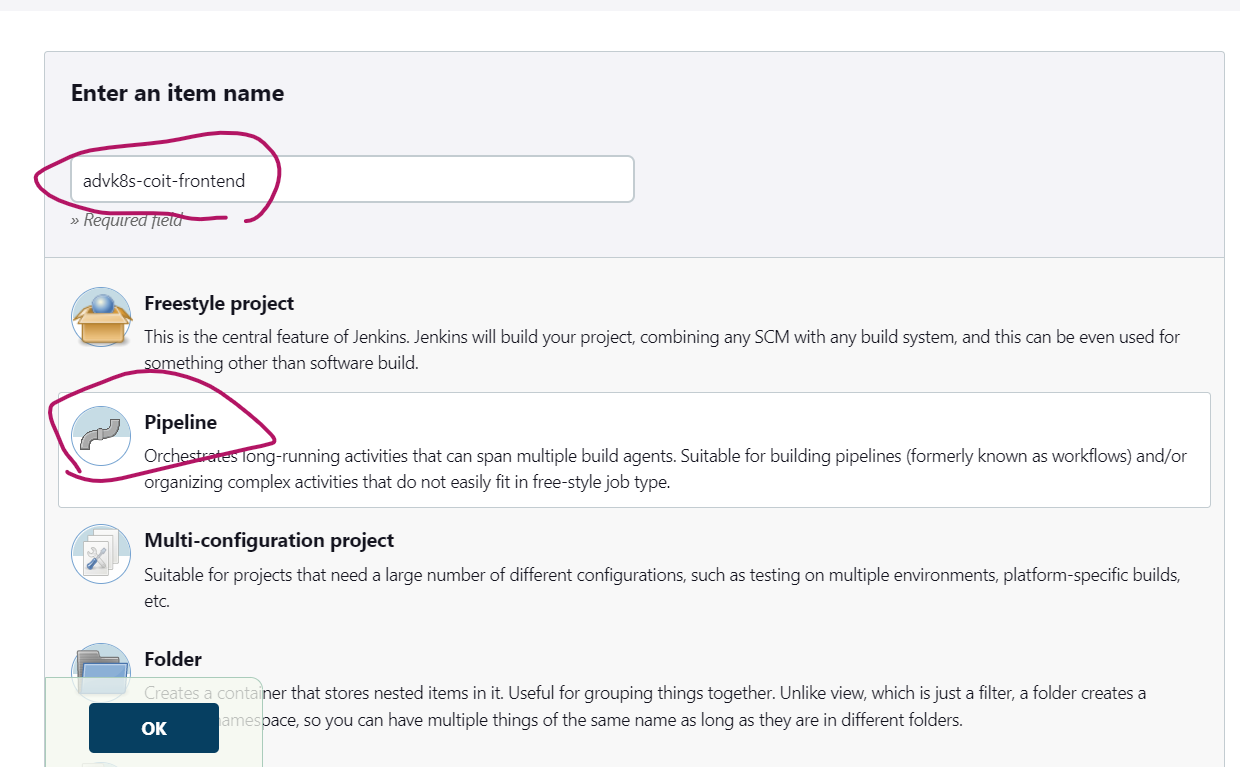
****

****

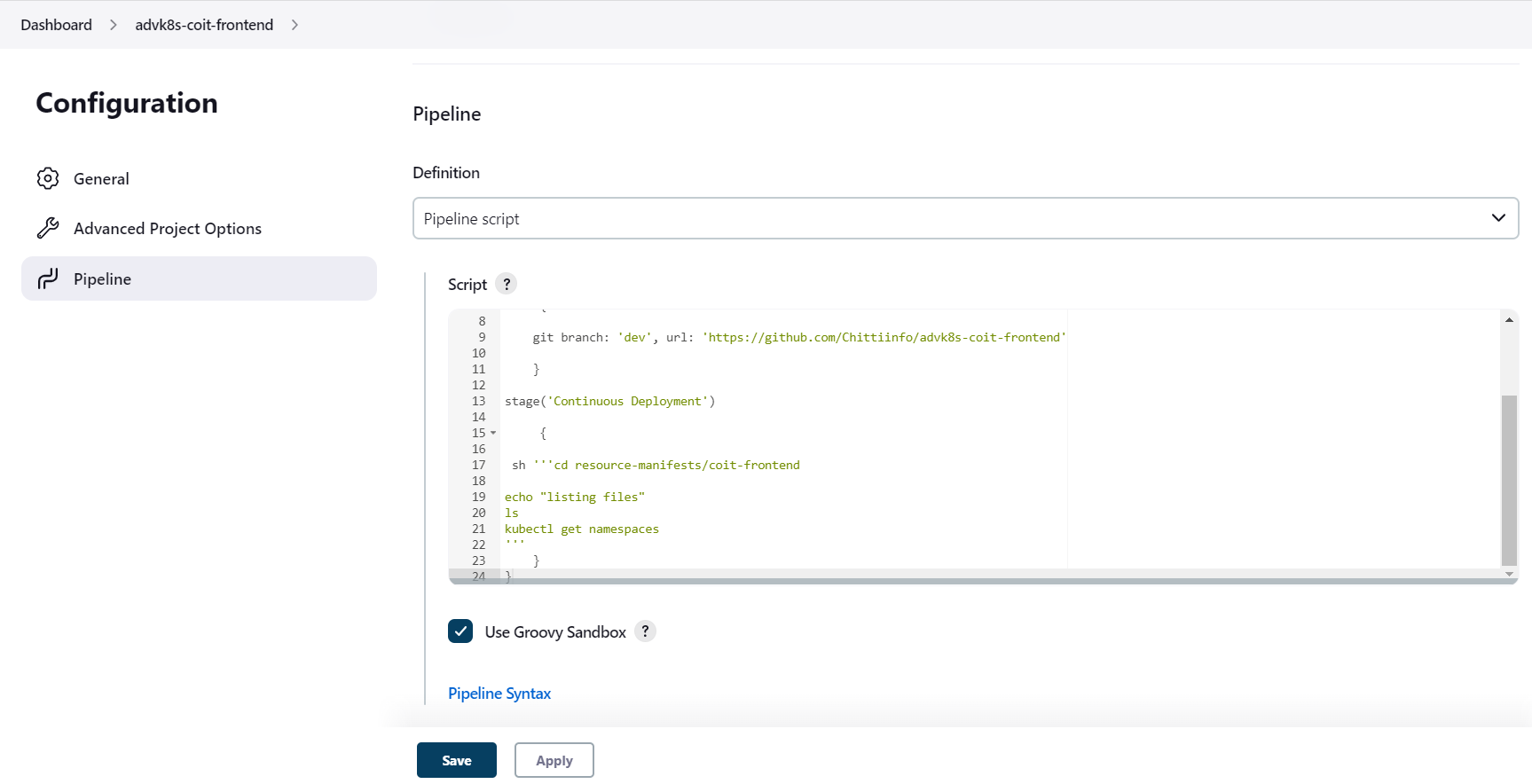
****

**Build Jenkins job:**

****

****

Validate github downloading and GKE cluster connection by using below groovy code.



**Sample Code:**

node('built-in')

{

stage('Continuous Download')

{

git branch: 'dev', url: 'https://github.com/Chittiinfo/advk8s-coit-frontend'

}

stage('Continuous Deployment')

{

sh '''cd resource-manifests/coit-frontend

echo "listing files"

ls

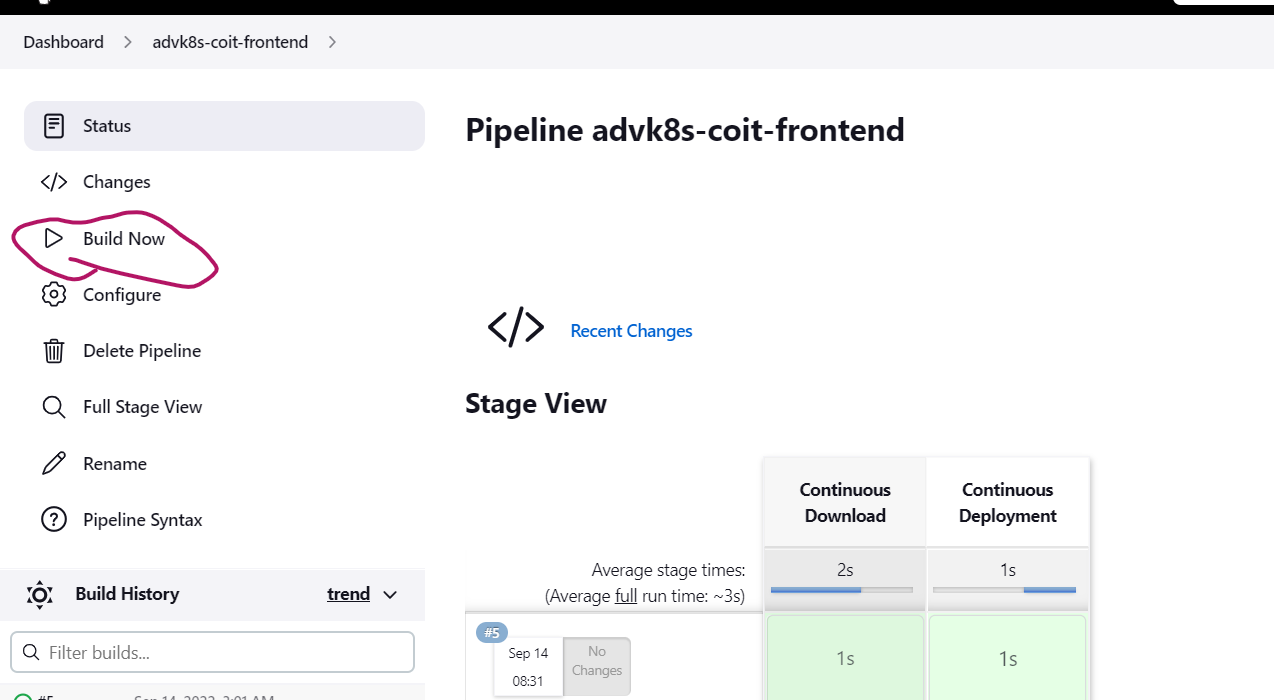
kubectl get namespaces

'''

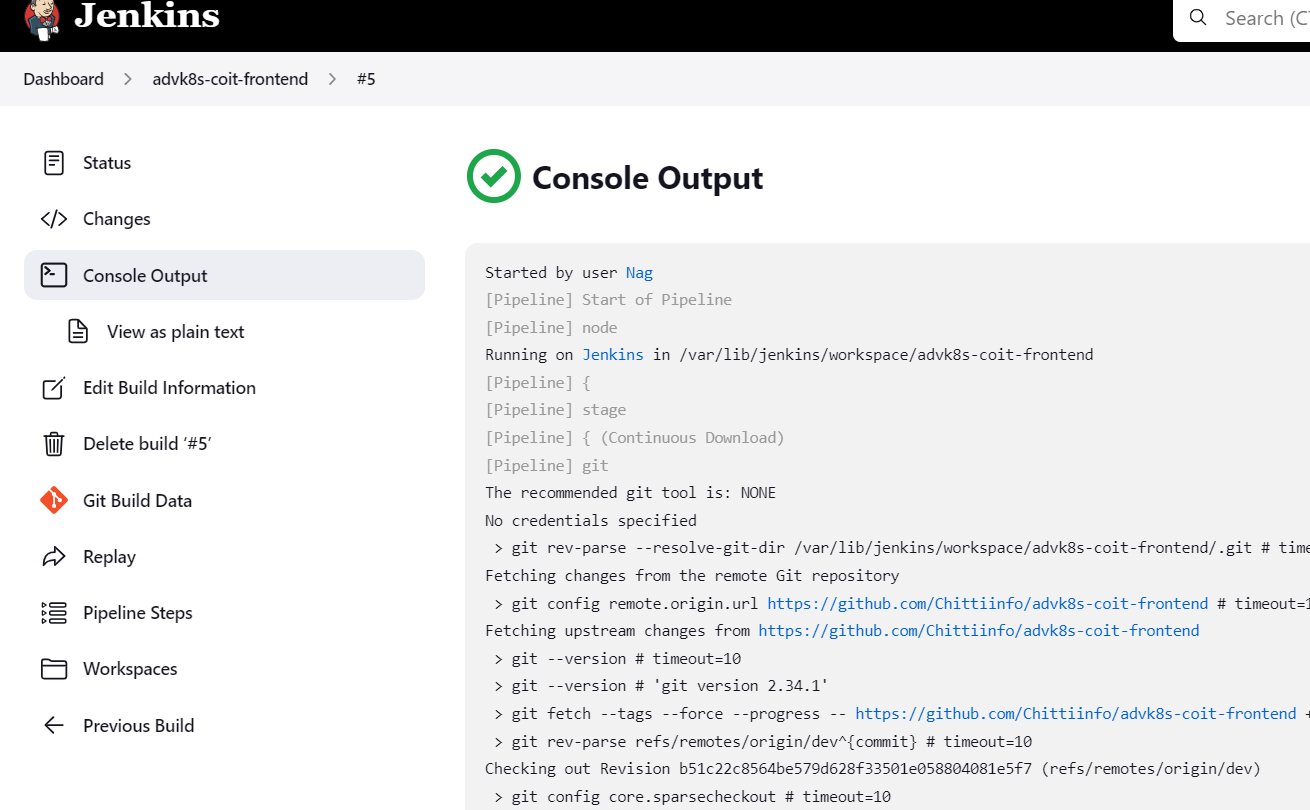
}

}

Once above code is saved in your Jenkins job then just run it.



You can check results from Console Output



**Building Pipeline to deploy application yaml files:**

**Sequence\order to deploy yml files into GKE:**

Roles

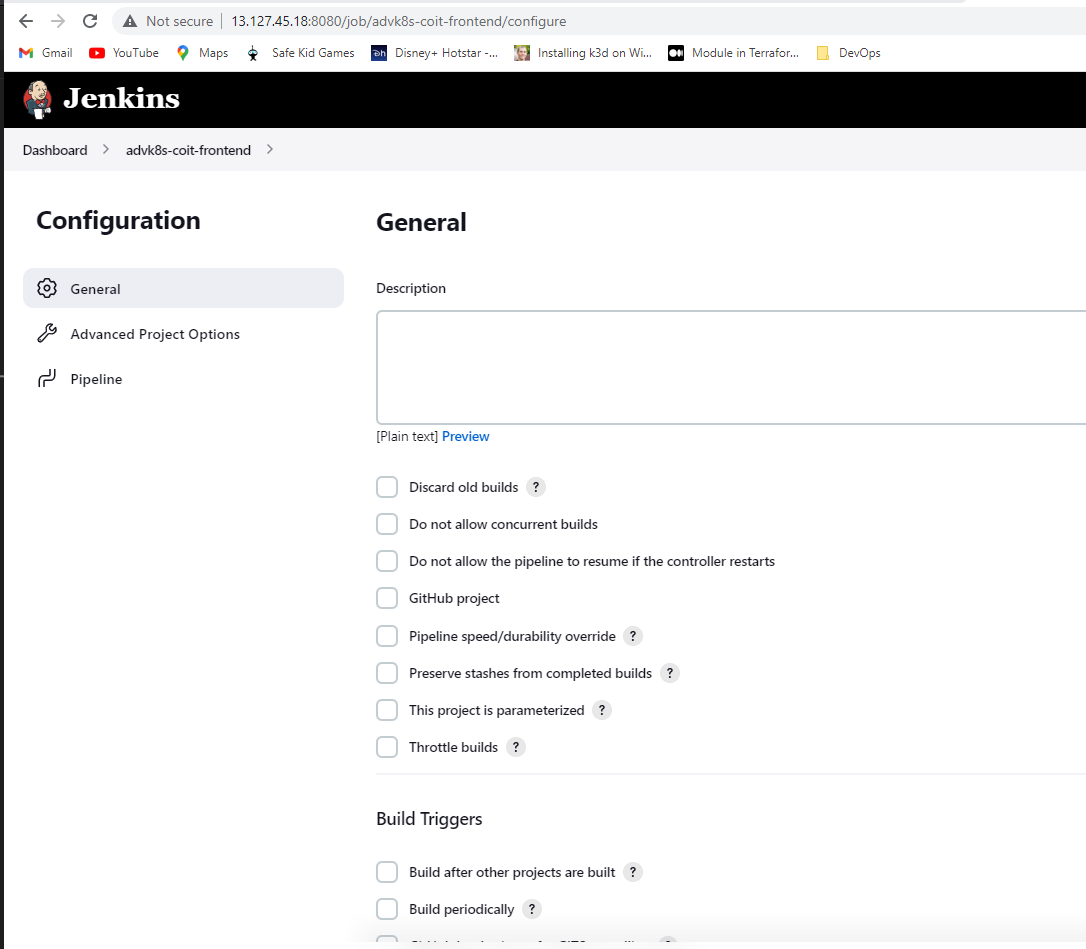
PVC

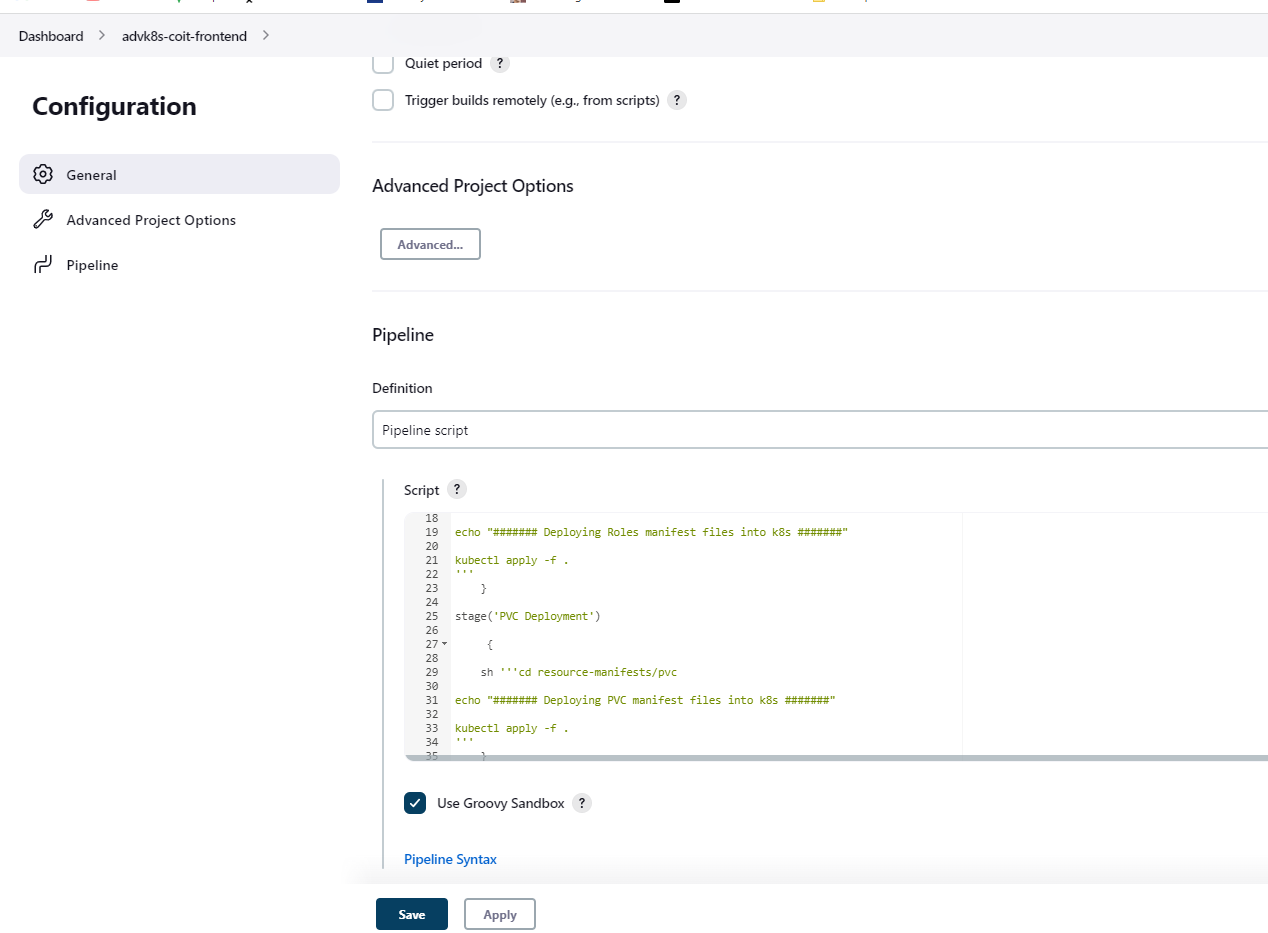
Issuers

Configmap

Coit-frontend

Ingress

****

****

**Groovy code:**

node('built-in')

{

stage('Continuous Download')

{

git branch: 'dev', url: 'https://github.com/Chittiinfo/advk8s-coit-frontend'

}

stage('Roles Deployment')

{

sh '''cd resource-manifests/roles

echo "####### Deploying Roles manifest files into k8s #######"

kubectl apply -f .

'''

}

stage('PVC Deployment')

{

sh '''cd resource-manifests/pvc

echo "####### Deploying PVC manifest files into k8s #######"

kubectl apply -f .

'''

}

stage('issuers Deployment')

{

sh '''cd resource-manifests/issuers

echo "####### Deploying issuers manifest files into k8s #######"

kubectl apply -f .

'''

}

stage('configmap Deployment')

{

sh '''cd resource-manifests/Configmap

echo "####### Deploying Config manifest files into k8s #######"

kubectl apply -f .

'''

}

stage('coit-frontend Deployment')

{

sh '''cd resource-manifests/coit-frontend

echo "####### Deploying application deployement,service and HPA manifest files with Kustomize #######"

kubectl apply -k .

'''

}

stage('ingress Deployment')

{

sh '''cd resource-manifests/ingress

echo "####### Deploying application Ingress manifest file #######"

kubectl apply -f .

sleep 20

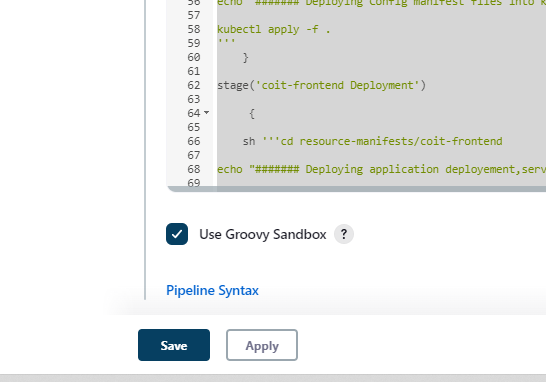
kubectl get service ingress-nginx-controller --namespace=ingress-nginx

'''

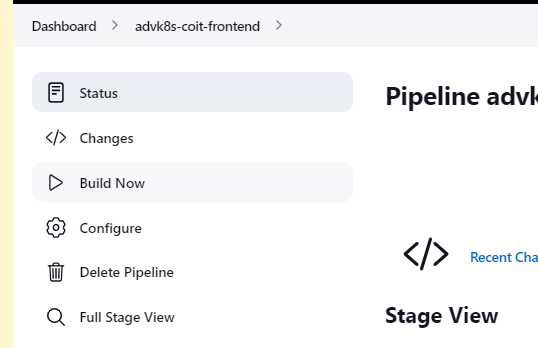
}

}

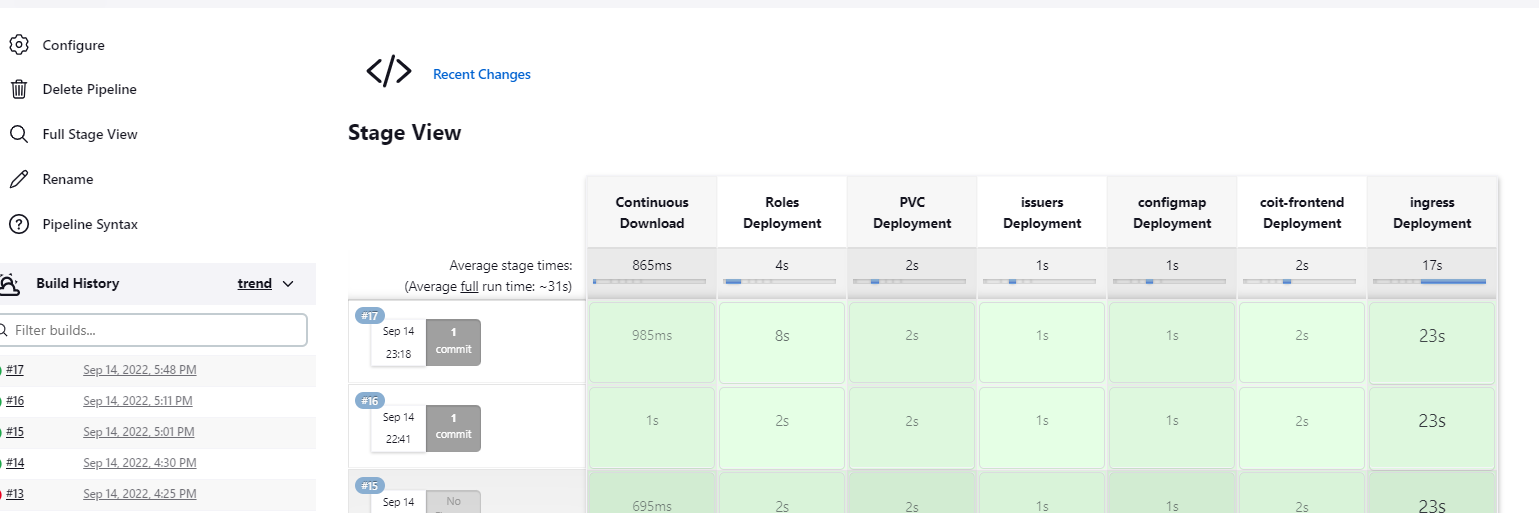
Once above code is copied, just apply and save.



Run the build now.



You can go to full stage view/console output to check the status of build and make sure its complete green\successfull.

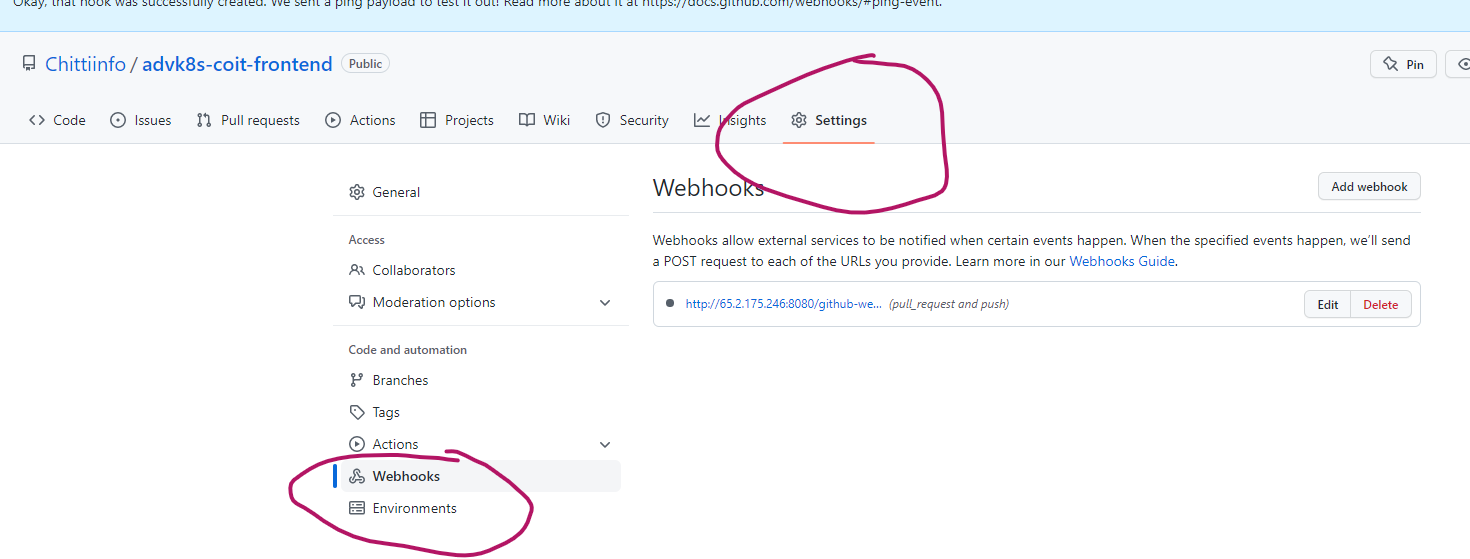


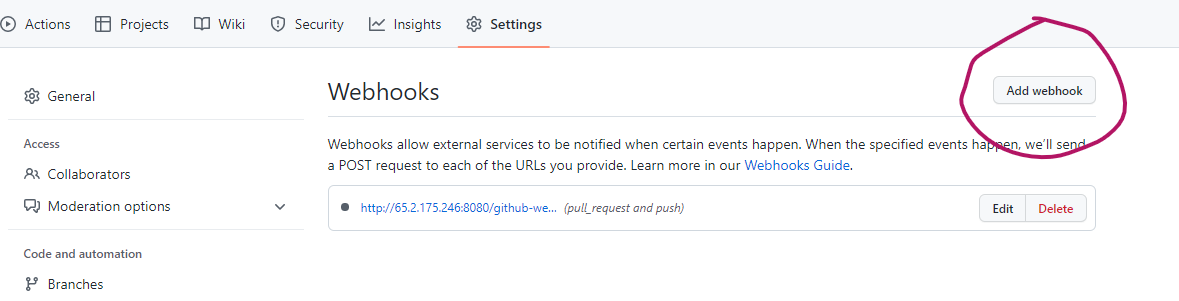


**How to trigger Jenkins Job automatically when there is a code commit in remote GitHub repository:**

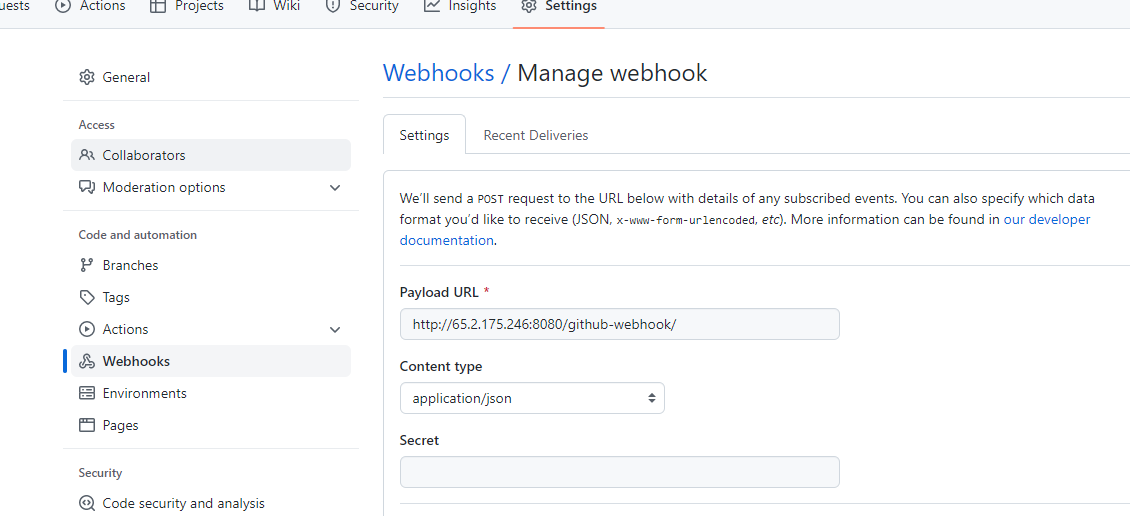
**Step 1: go to your GitHub repository and click on ‘Settings’.**

**Step 2: Click on Webhooks and then click on ‘Add webhook’.**

****

****

**Step 3: In the ‘Payload URL’ field, paste your Jenkins environment URL. At the end of this URL add /github-webhook/. In the ‘Content type’ select: ‘application/json’ and leave the ‘Secret’ field empty.**

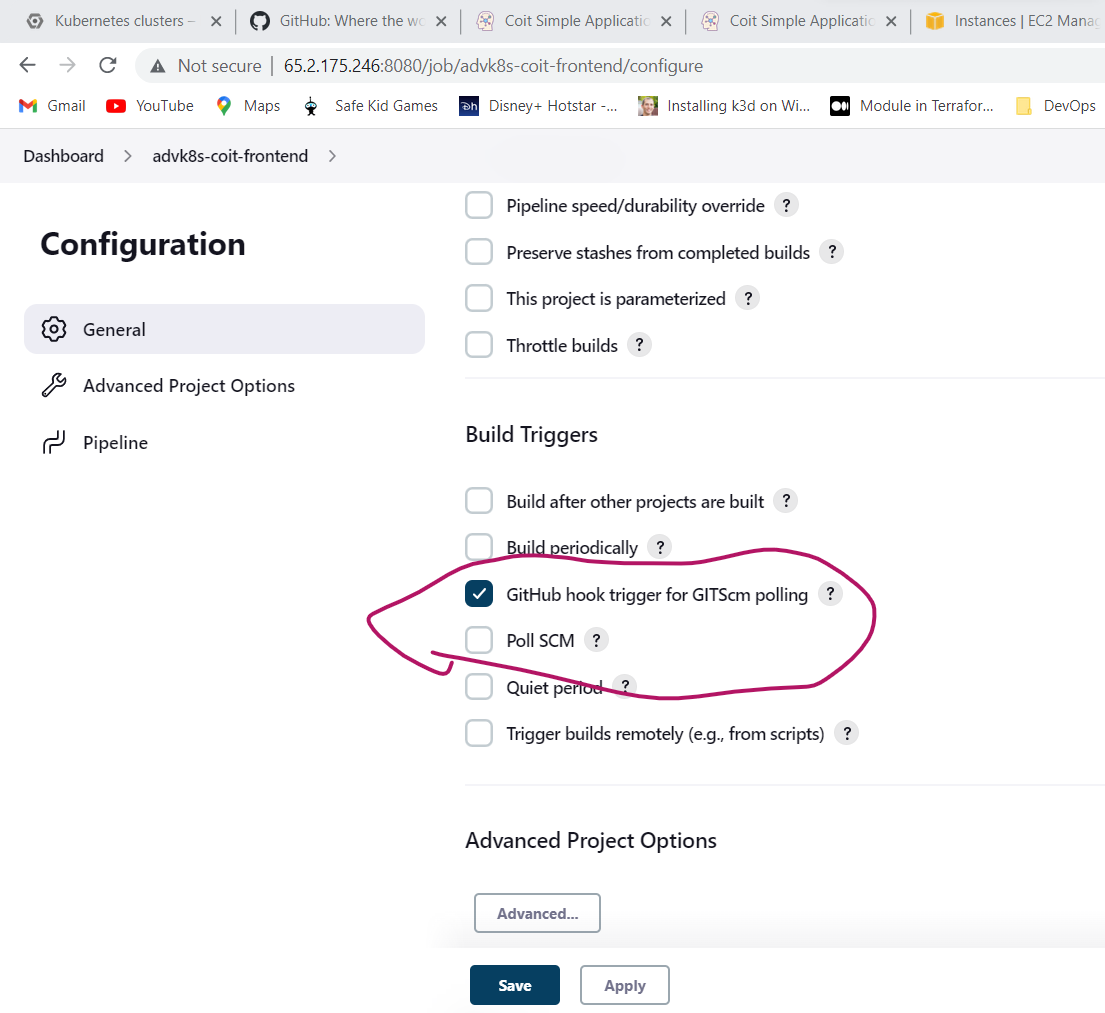
****

**Step 4: In the page ‘Which events would you like to trigger this webhook?’ choose ‘Let me select individual events.’ Then, check ‘Pull Requests’ and ‘Pushes’. At the end of this option, make sure that the ‘Active’ option is checked and click on ‘Add webhook’.**

****

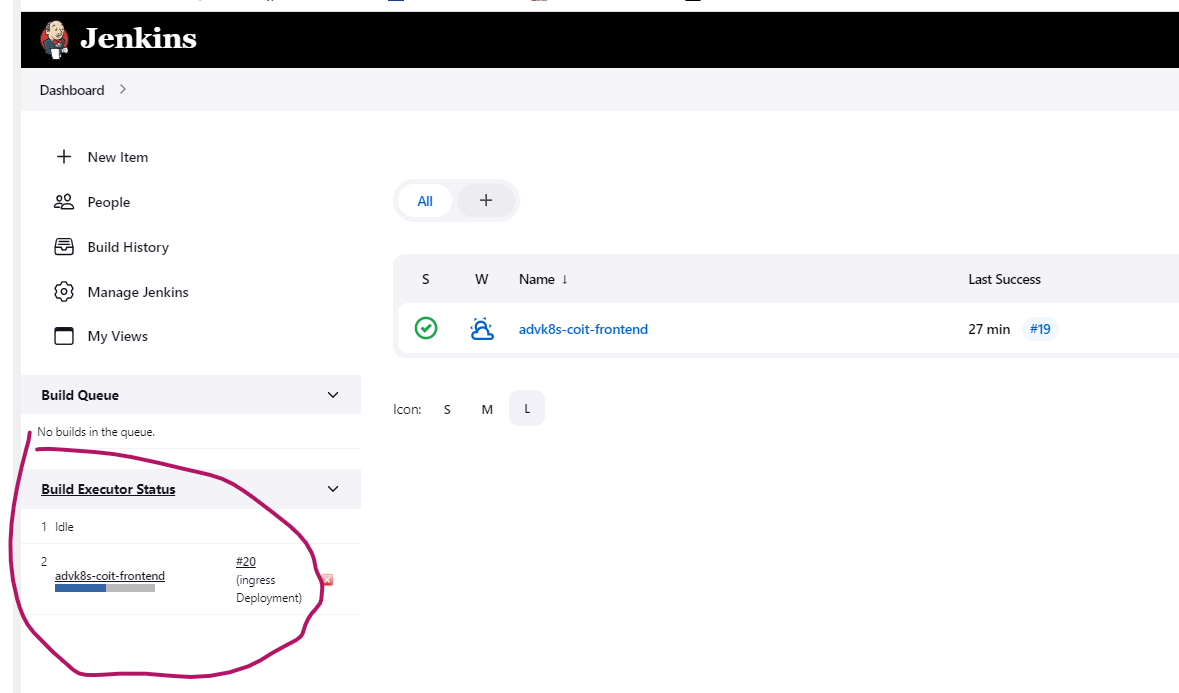
**We're done with the configuration on GitHub’s side! Now let's move on to Jenkins.**

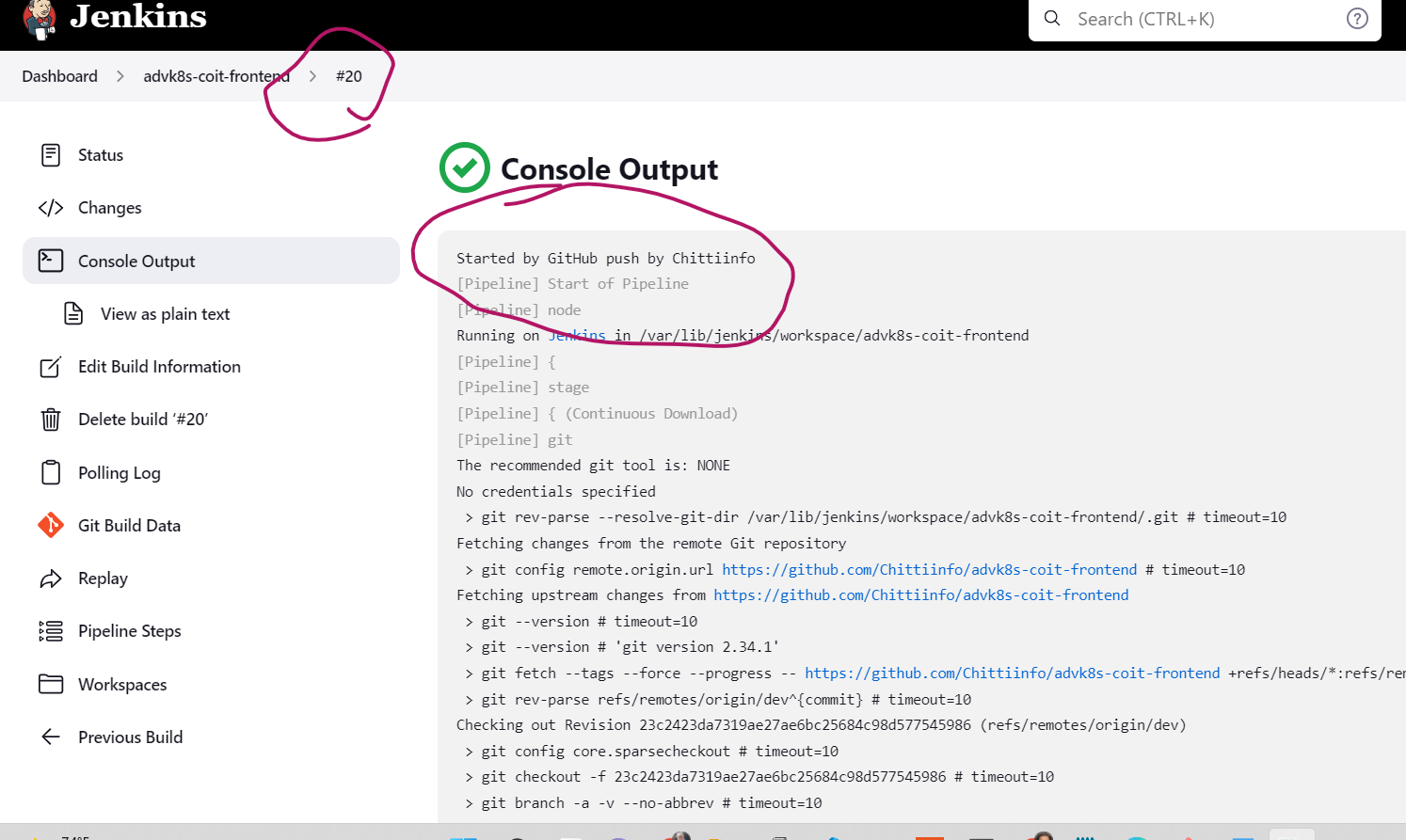
In the ‘Build Triggers’ tab enable the ‘GitHub hook trigger for GITScm polling’. Or, choose the trigger of your choice.

****

That's it! Your GitHub repository is integrated with your Jenkins project. With this Jenkins GitHub integration, you can now use any file found in the GitHub repository and trigger the Jenkins job to run with every code commit.

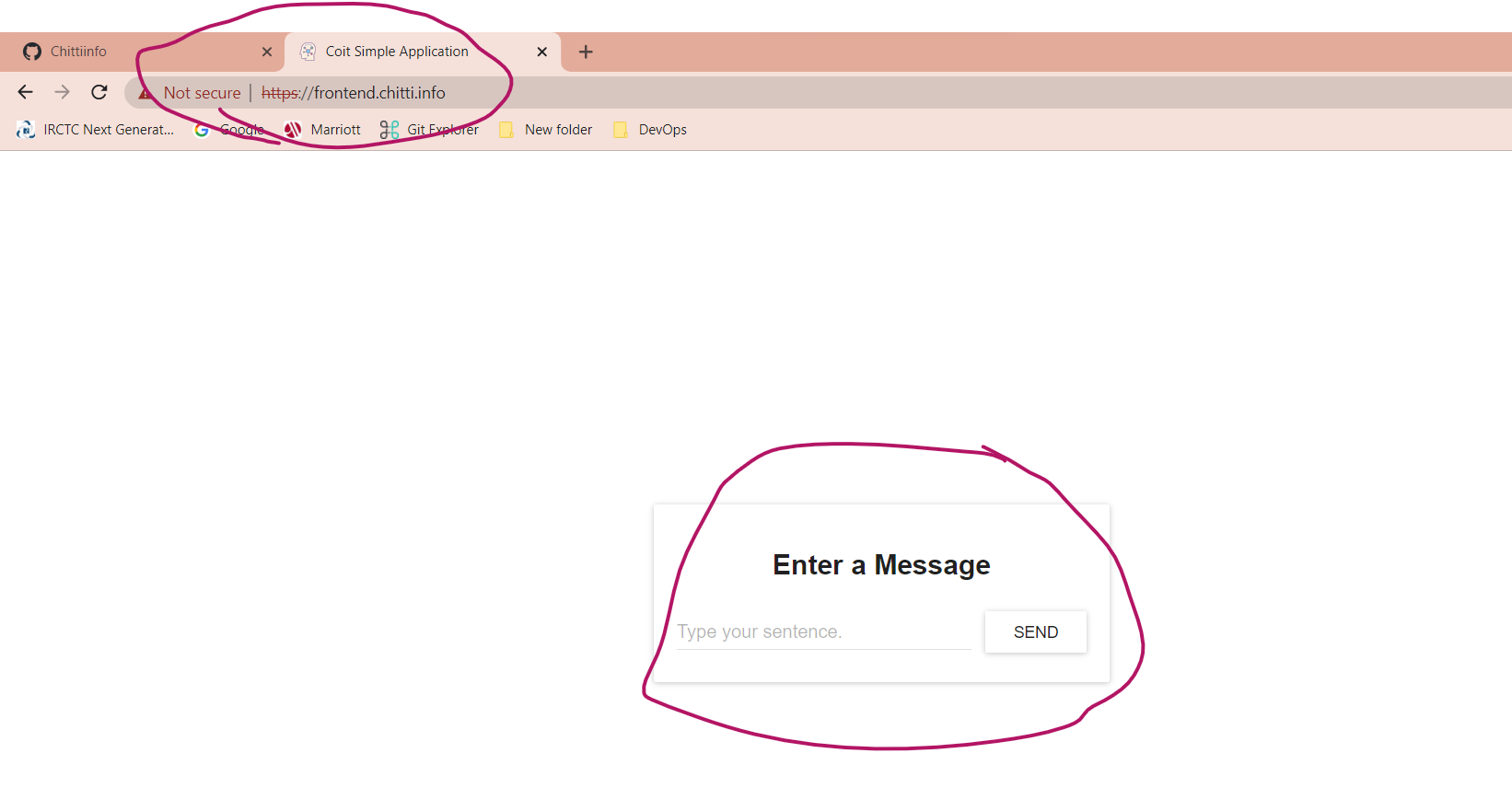
I have just made code changes in local git and committed code changes to GitHub. As soon as I committed code to my github below job (job no #20) got triggered automatically and my deployment is also successful.



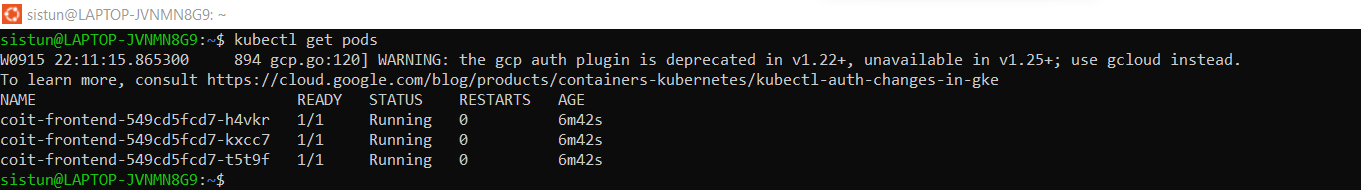


**Validation:**

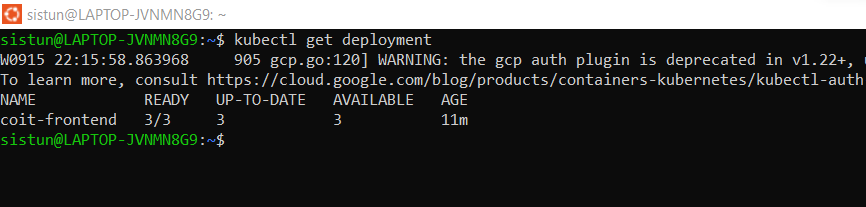
Access this site <http://frontend.chitti.info/> and it should be showing below page.



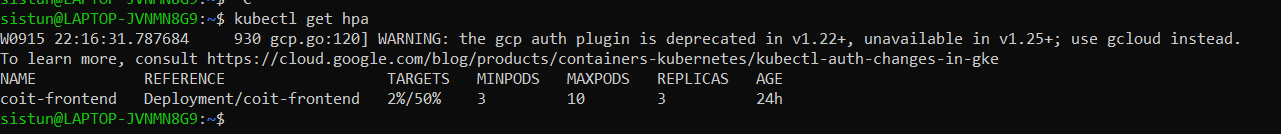
Min 3 pods are running as per our HPA configuration (min 3 max 10)



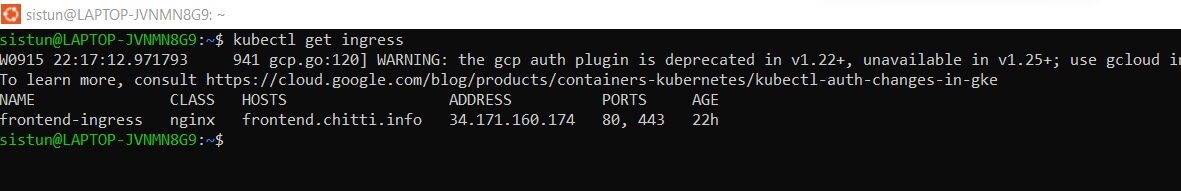
Deployment is running with 3 replicas



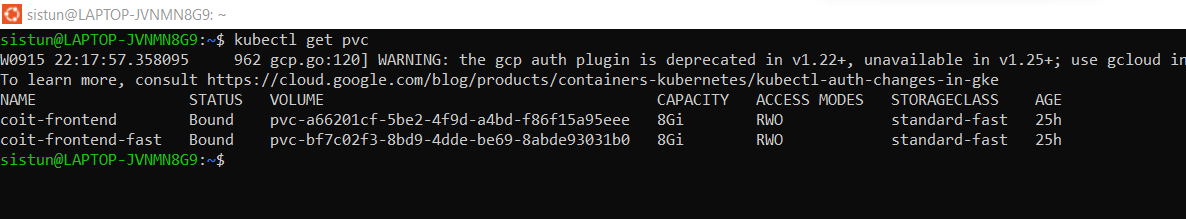
HPA is running fine



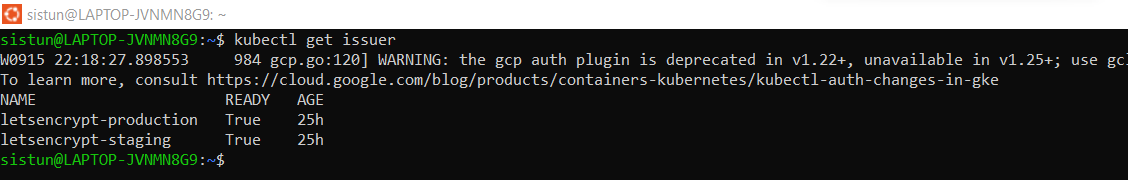
Ingress



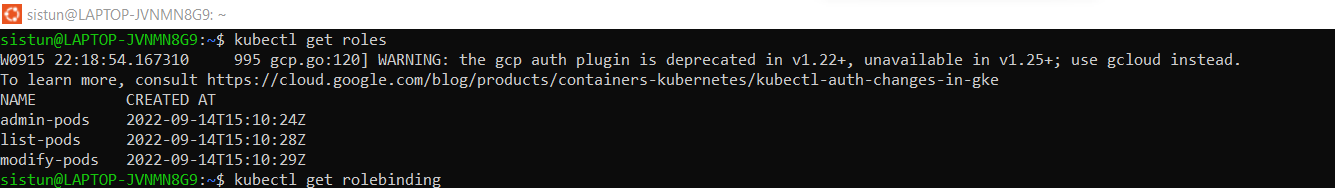
PVC

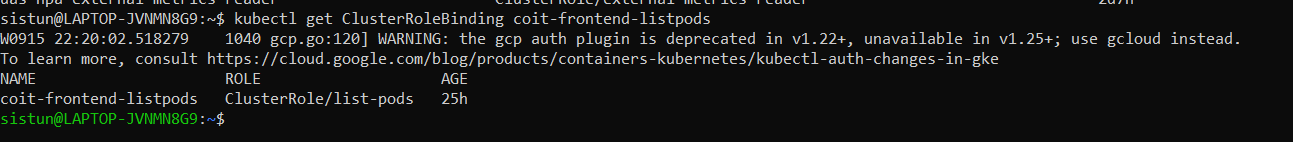


Issuer



Roles & ROlebindings





Configmap.

