**Project: CI/CD with GitHub Actions**

**Project Requirement:**



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

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

**Existing process challenges and possible solutions:**

Currently we follow the “git flow” approach for branching and it's giving us a lot of trouble and we can’t seem to achieve continent integration.

We want to switch to mainline based approach with feature branches as our branching strategy.

Our developers will not be permitted to commit to the mainline and they must create a pull request instead. A Pull request should be automatically validated by an automated CI process

**Project Plan:**

* Create two branches (main & develop or feature), main branch will have the actual code for production deployment and feature/dev branch will have all the CI preview code.
* Ensure Main branch is restricted from direct merges. Developers should create a PR then wait for reviewers to review the code and merge it with main branch.
* Create two github work flows, one is for CI (Preview environment) and another one is for CD (Prod environment)
* Ensure CI workflow is triggered automatically when there is a merge request
* Ensure production deployment job is triggered when there is a Tag creation.
* Create a managed K8S cluster in GKE with min 3 worker nodes.
* Install Sonar QUBE in K8S for code review.
* Install Git Client on local machines to commit code changes to Github repository.
* Develop all manifest files that we require to deploy microservice based application into K8S.
* Develop Github actions workflow yaml files and ensure it performs all below actions. And also ensure we use Kustomize configuration management tool to create all resources in K8S.

1) Code Review

2) Unit Tests

3) Dependency Vulnerability checks.

4) Packaging the artifacts

5) Building and Pushing container images

6) Creating a new namespace in our existing kubernetes cluster.

7) Deploy the application to create a preview

8) Delete the namespace after 30 minutes.

9) Notify developers in case of any failure in any of the above steps.

10) Send a consolidated report to my developers ( Number of Vulnerabilities in dependencies, Number of Issues by Sonarqube, URL of the app preview, URL of the sonarqube project )

11) only changed files should be scanned by Sonar qube

12) write a script that can check document and if there are any broken URLs in document CI job should be failed.

**Project Schedule:**

**Day 1 Tasks:**

Discuss about project requirement and implementation plan

Complete lab setup ( GKE cluster setup, tools installations, connectivity establishment from github workflow to GKE etc)

**Day 2 Tasks:**

Complete all the CI steps and CI job is successful

**Day 3 Tasks:**

Complete all the CD steps and ensure prod deployment is successful when there is a new Tag creation.

**Day 4 Task:**

Validate application and configurations.

Deliver the project.

Document

**Day 5 Task:**

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

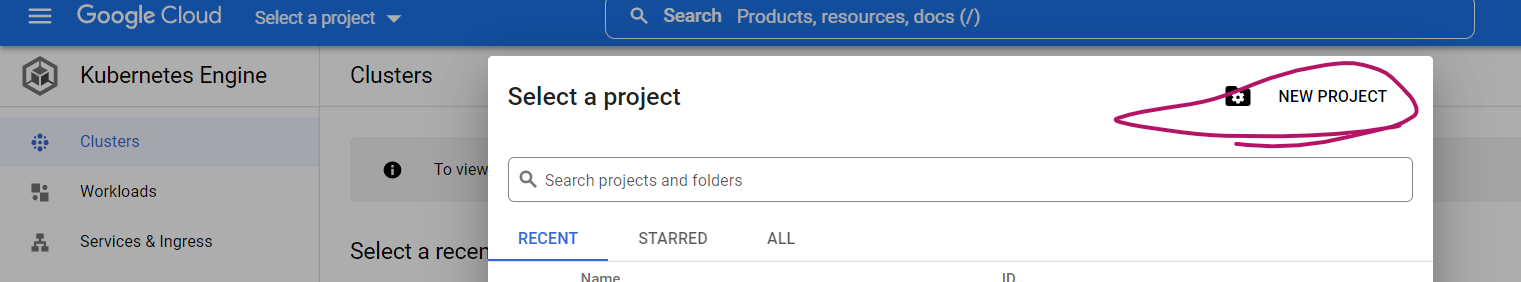
Close the project

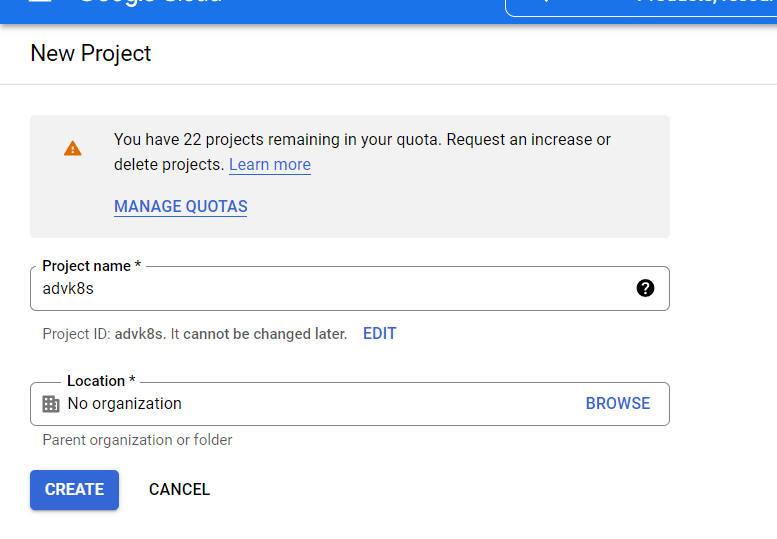
**GKE Work:**

Login into gcloud console

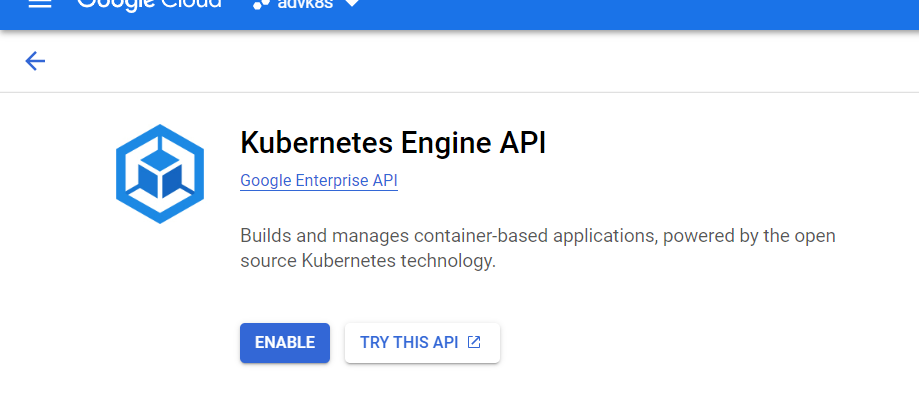
<https://console.cloud.google.com/kubernetes/list/overview?project=tidal-copilot-359606>

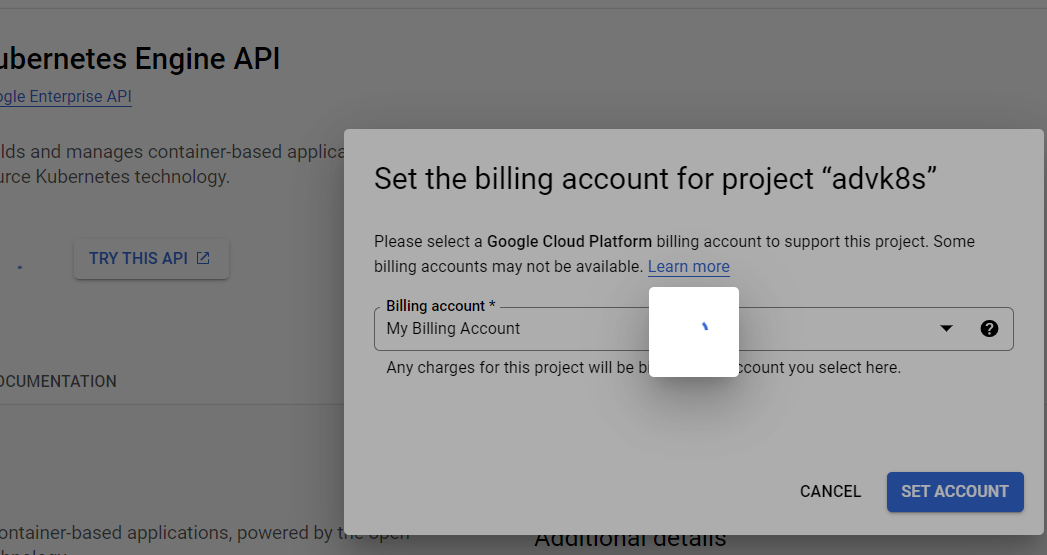
Create a Project with any name



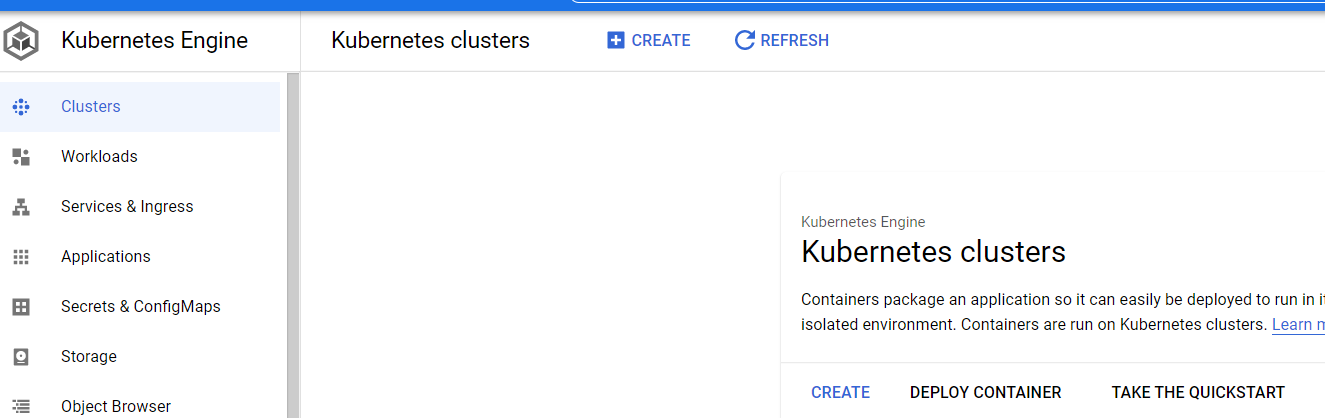


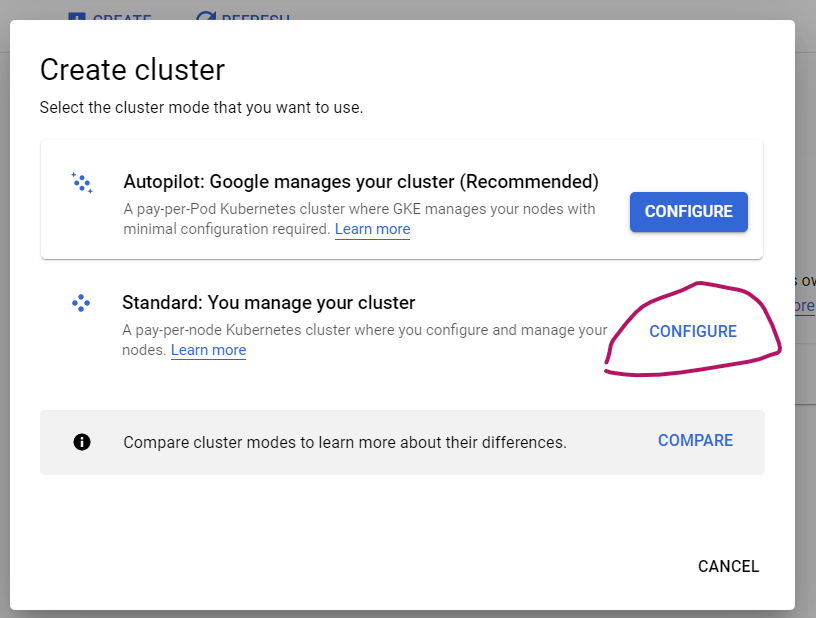
Enable Kubernetes engine api

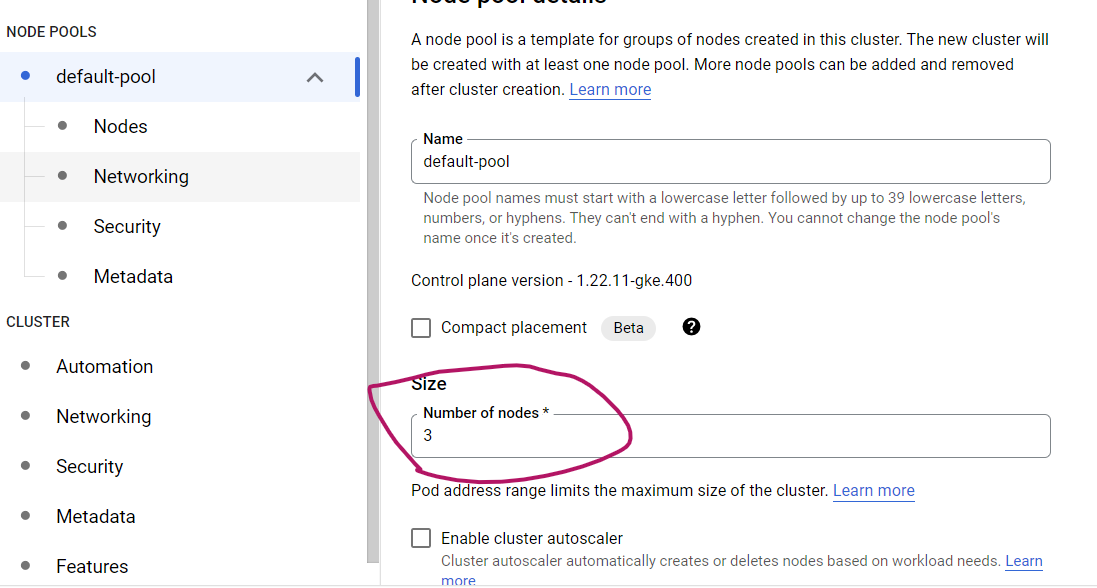


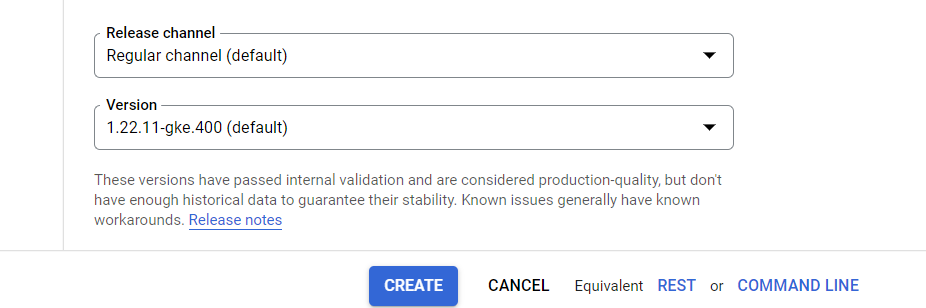


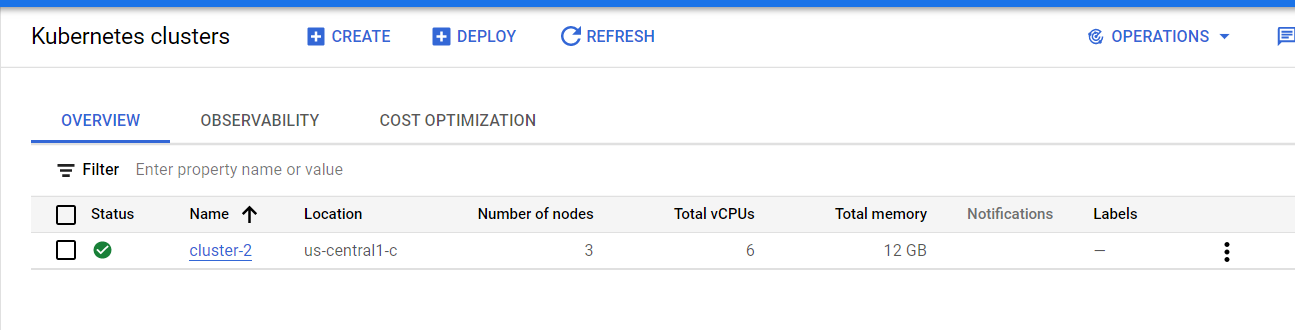
Click on Create GKE cluster



****

****

****

****

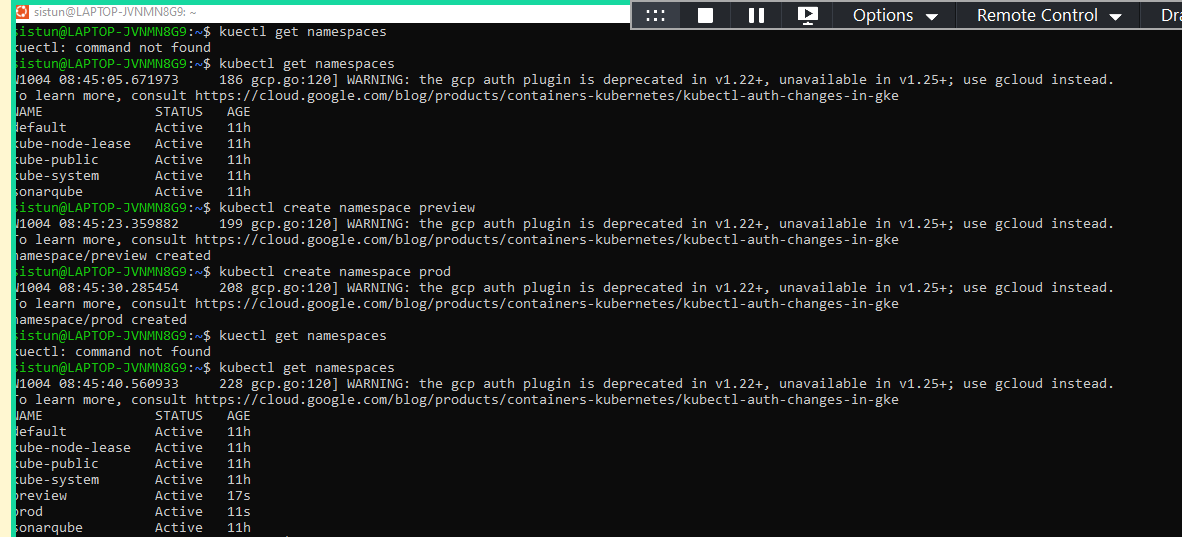
Once GKE cluster is up, you can create below namespaces. One is for deploying CI changes and another one is for deploying CD changes

kubectl get namespaces

kubectl create namespace preview

kubectl create namespace prod

kubectl get namespaces

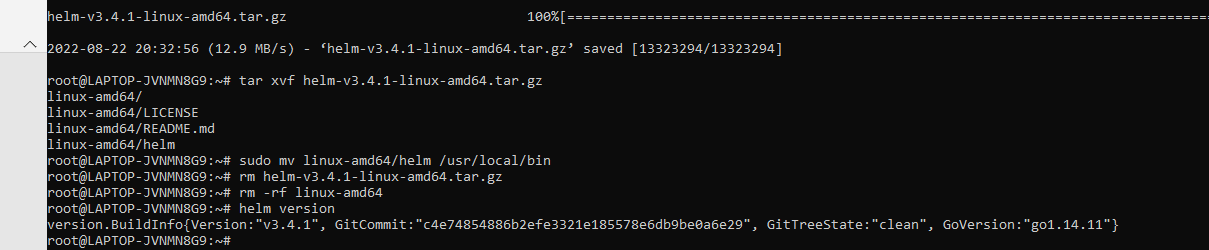


**Install SonarQube on GKE:**

We are using HELM packer manger to setup SonarQube on GKE

Note: Install HELM tool on your local Ubuntu machine before you use helm commands to install sonarqube on GKE

<https://phoenixnap.com/kb/install-helm>



https://docs.sonarqube.org/latest/setup/sonarqube-on-kubernetes/

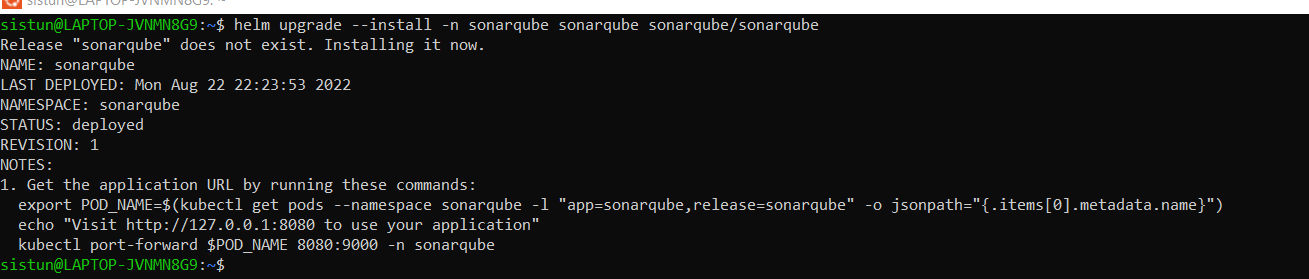
To install the Helm Chart from our Helm Repository, you can use the following commands:

helm repo add sonarqube https://SonarSource.github.io/helm-chart-sonarqube

helm repo update

kubectl create namespace sonarqube

helm upgrade --install -n sonarqube sonarqube sonarqube/sonarqube



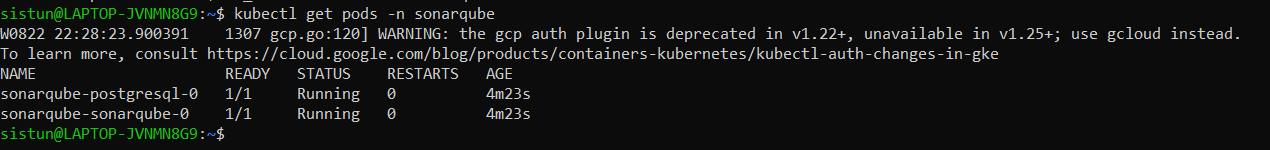
NOTES:

1. Get the application URL by running these commands:

export POD\_NAME=$(kubectl get pods --namespace sonarqube -l "app=sonarqube,release=sonarqube" -o jsonpath="{.items[0].metadata.name}")

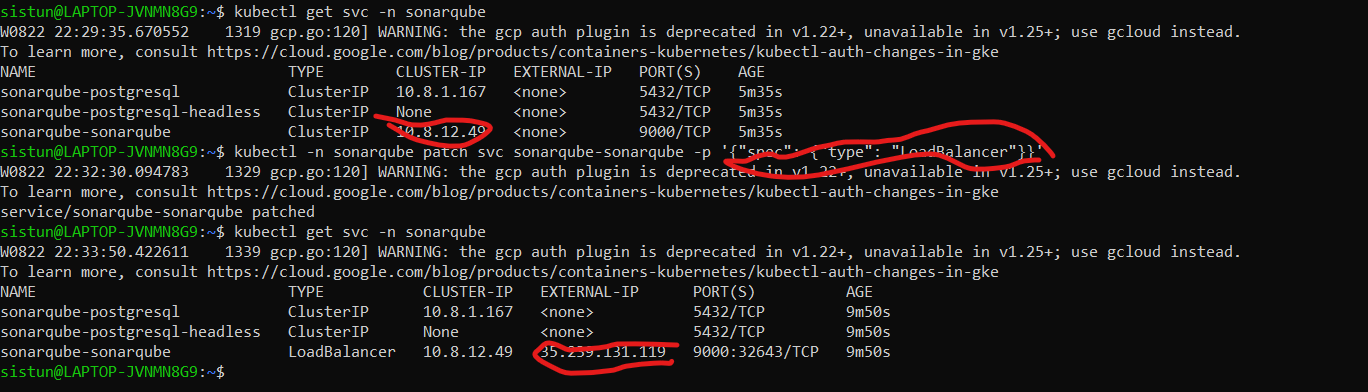
echo "Visit http://127.0.0.1:8080 to use your application"

kubectl port-forward $POD\_NAME 8080:9000 -n sonarqube



By default sonarqube created as a cluster IP and we can change it to load balancer type by using patch command so that we can use sonarqube for GitHub Actions CI\CD communication.

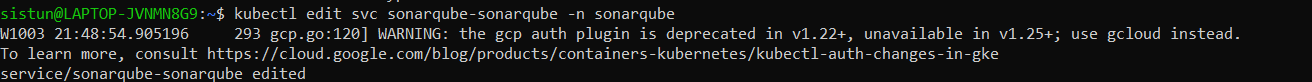
kubectl -n sonarqube patch svc sonarqube-sonarqube -p '{"spec": {"type": "LoadBalancer"}}'

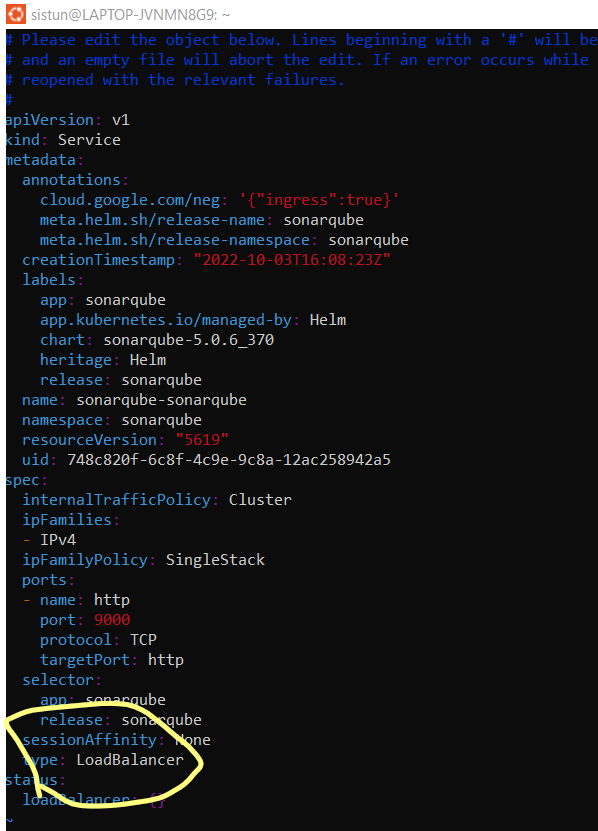


*Optional: You can also change cluster type by editing below service*

kubectl get svc -n sonarqube

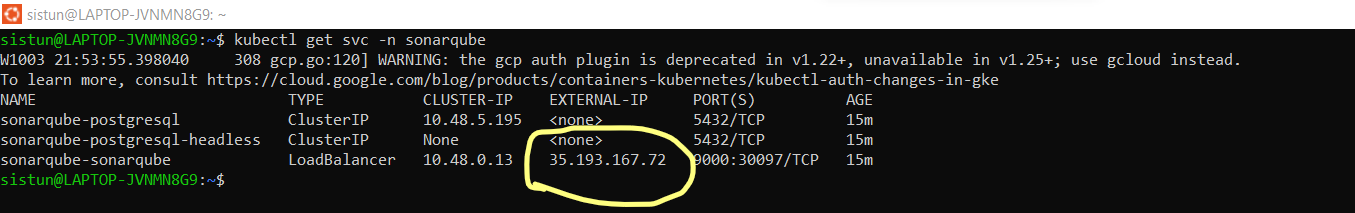
kubectl edit svc sonarqube-sonarqube -n sonarqube



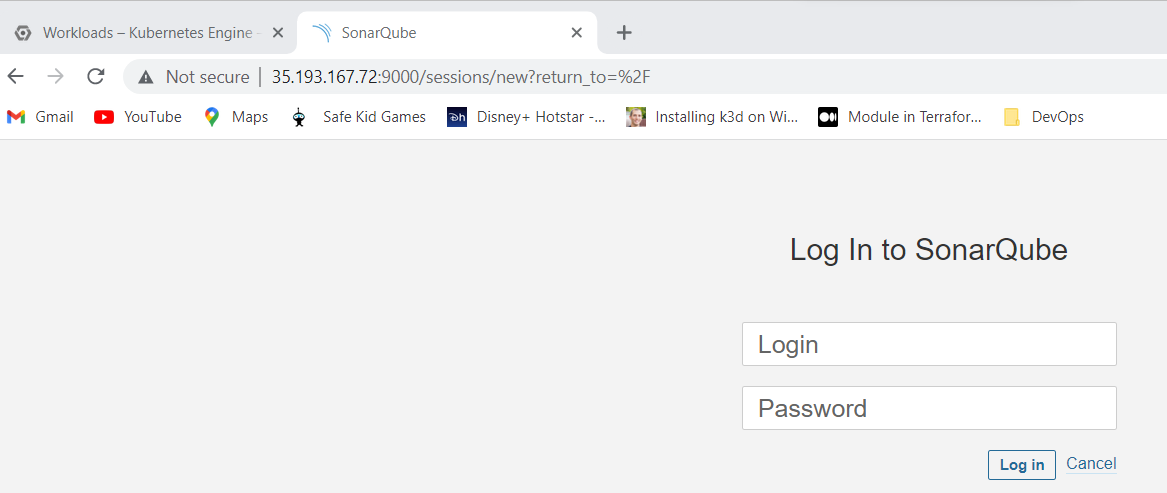


You can see sonarqube external IP now

kubectl get svc -n sonarqube

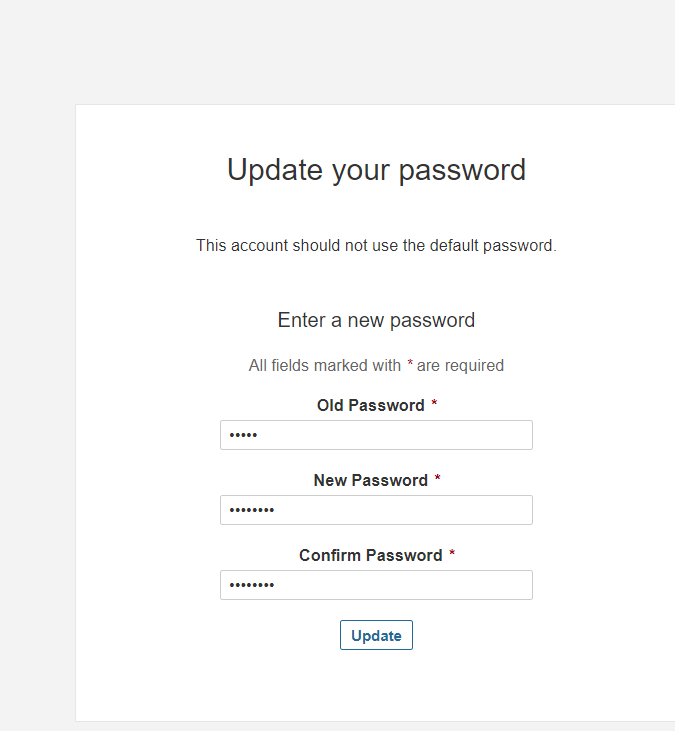


Access URL <http://35.193.167.72:9000>

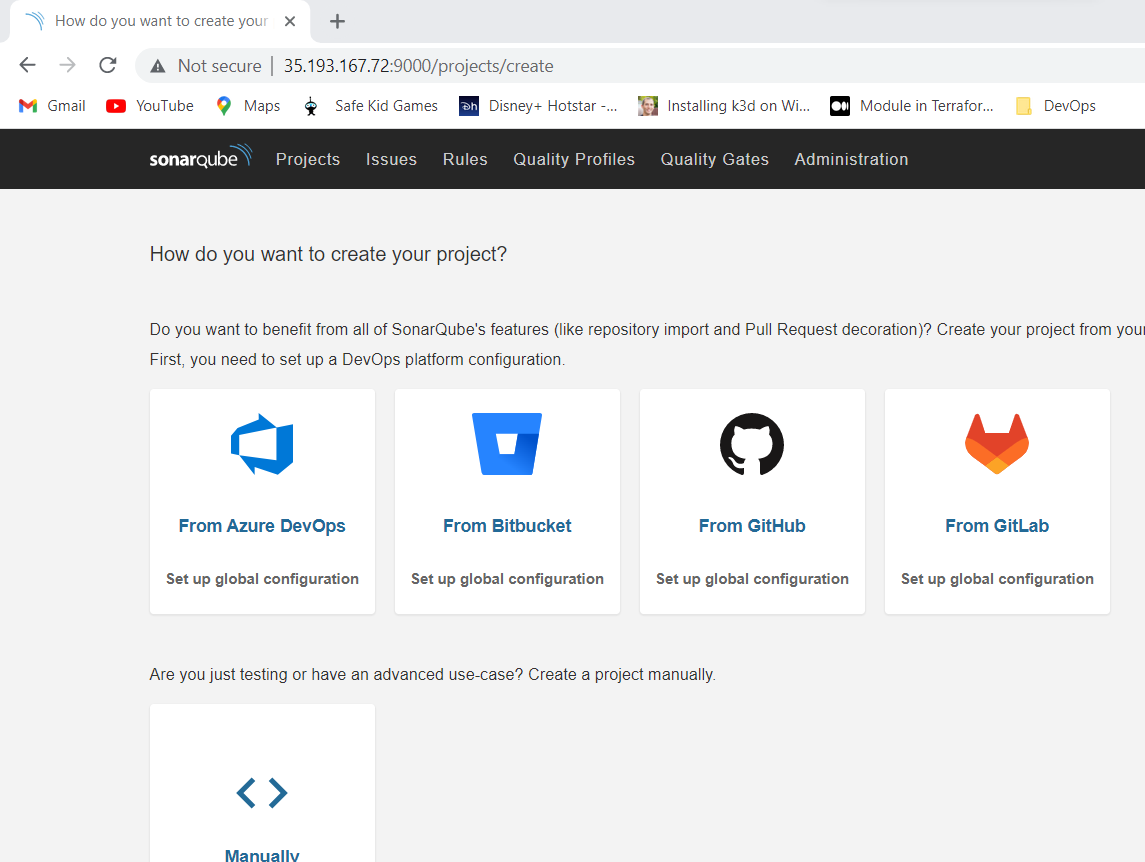


**UN:admin**

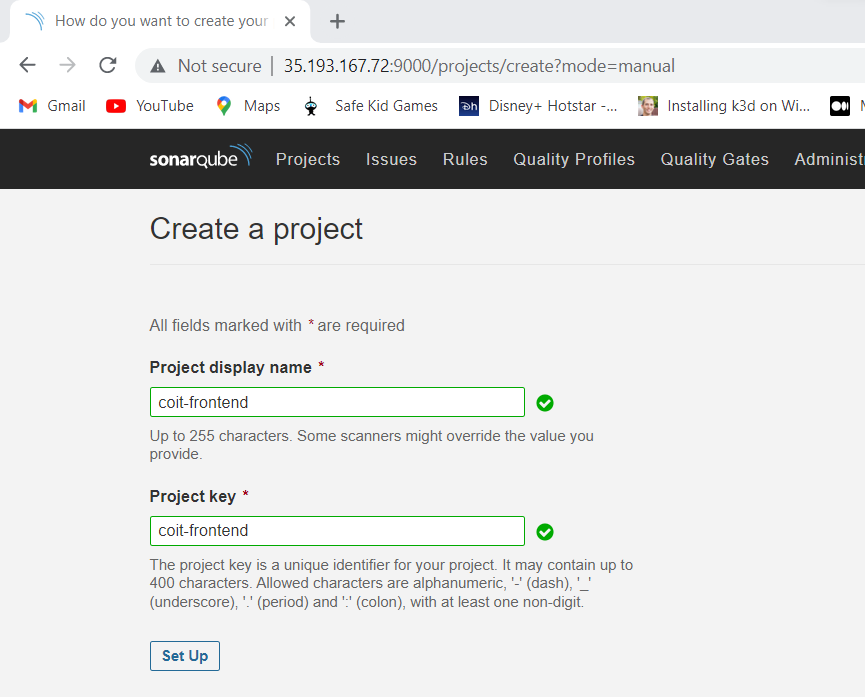
**Password: admin**

****

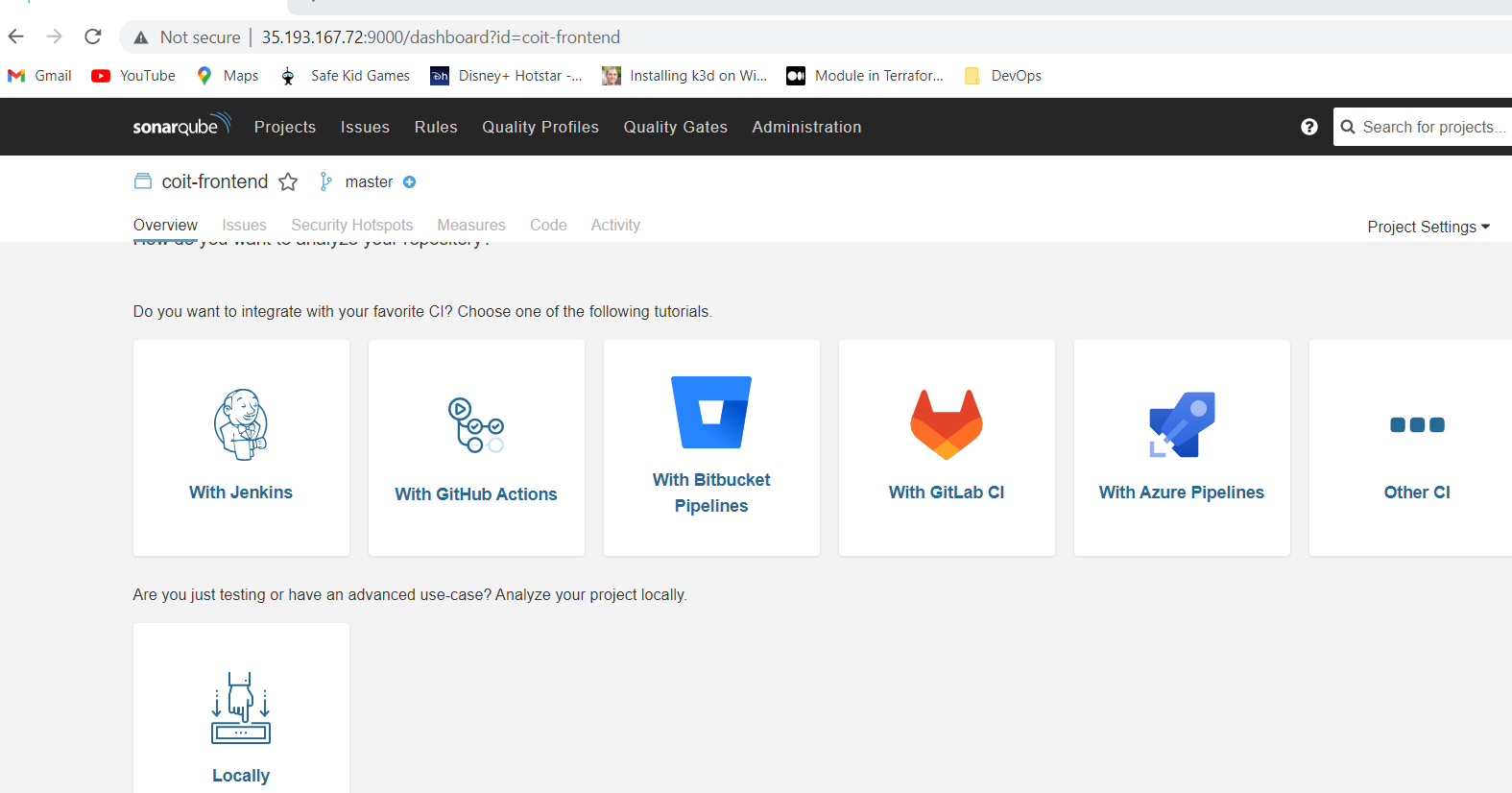
**Select Manually**

****

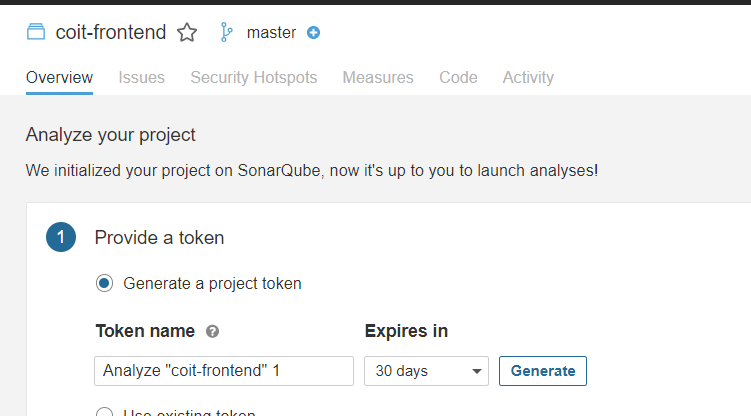
**Give project name & key (it can be any name)**

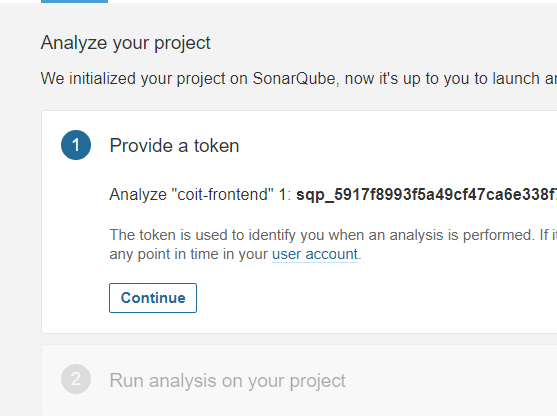
****

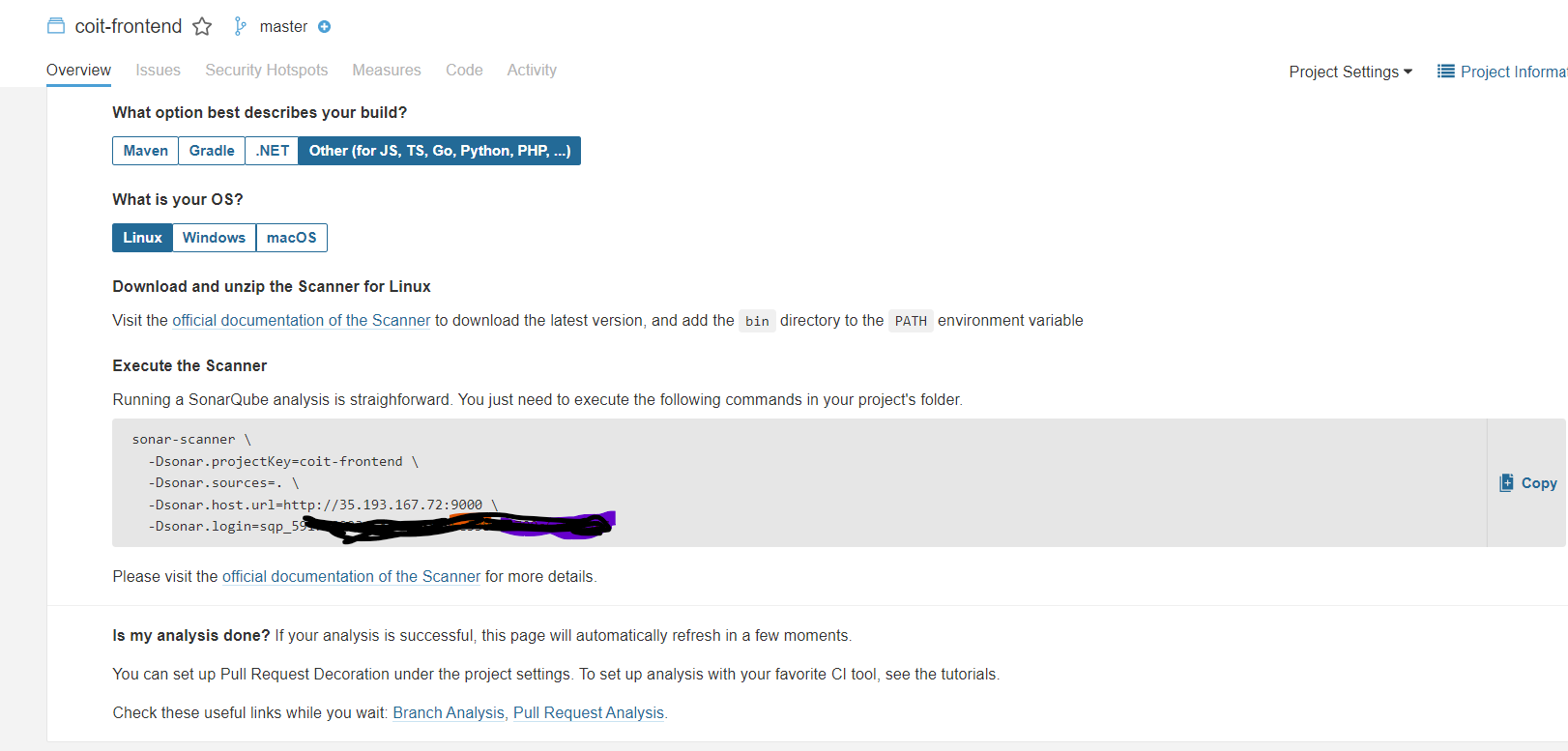
**Select Locally:**

****

**Generate token**

****





Save below values safely as we will use these values in our CI process for code review later

sonar-scanner \

-Dsonar.projectKey=coit-frontend \

-Dsonar.sources=. \

-Dsonar.host.url=http://35.193.167.72:9000 \

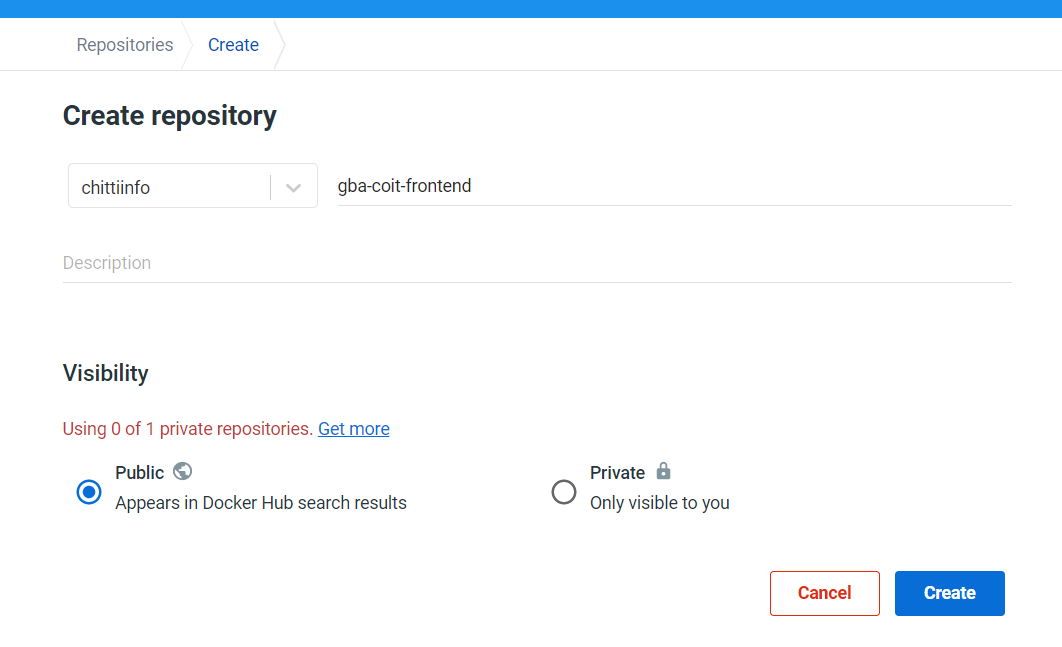
-Dsonar.login=

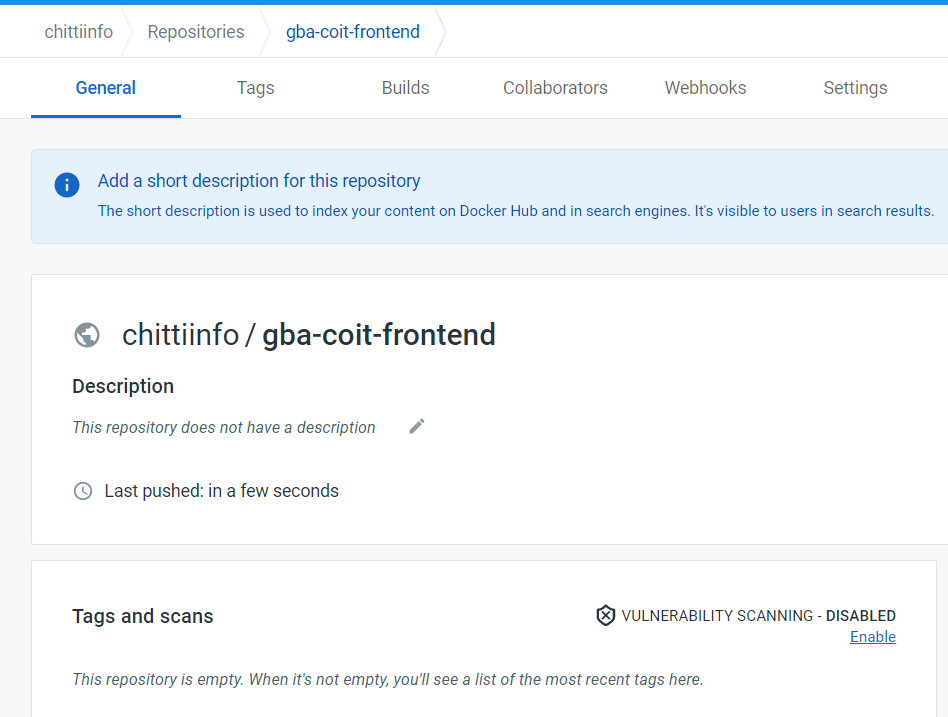
**Docker Work:**

Login into your docker hub account

https://hub.docker.com/

Create a repository

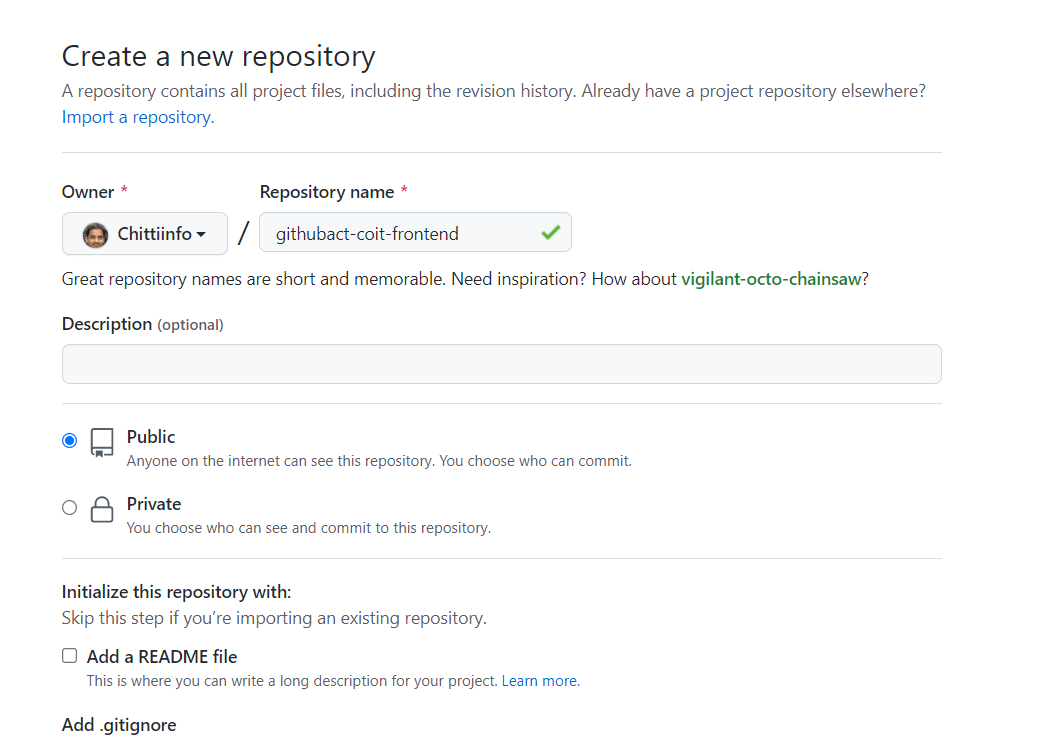


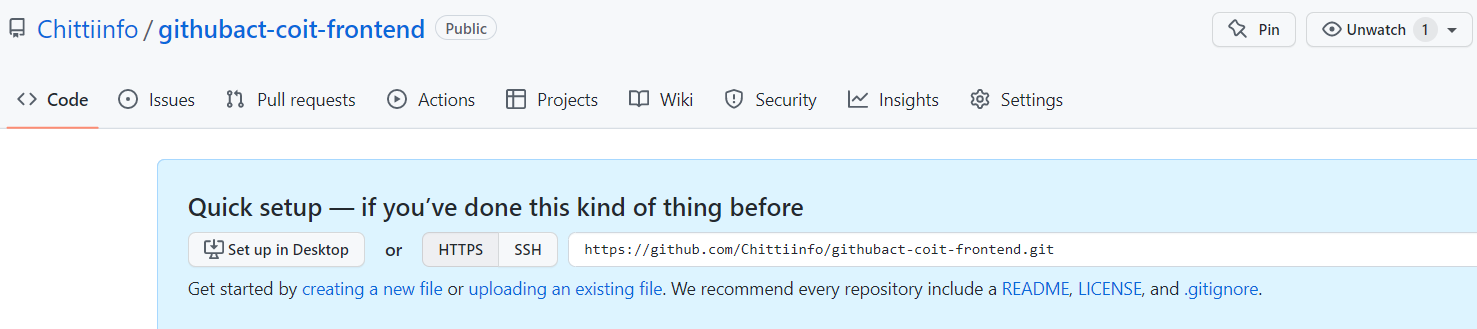


Make a note of this repository name as we will be pushing build docker images to this repository from CI workflow later.

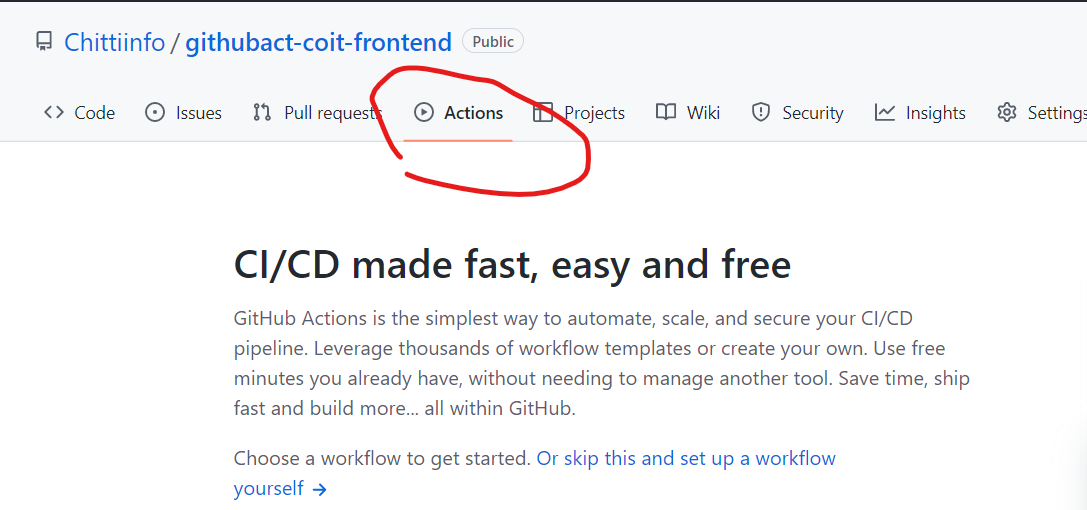
**GitHub Work:**

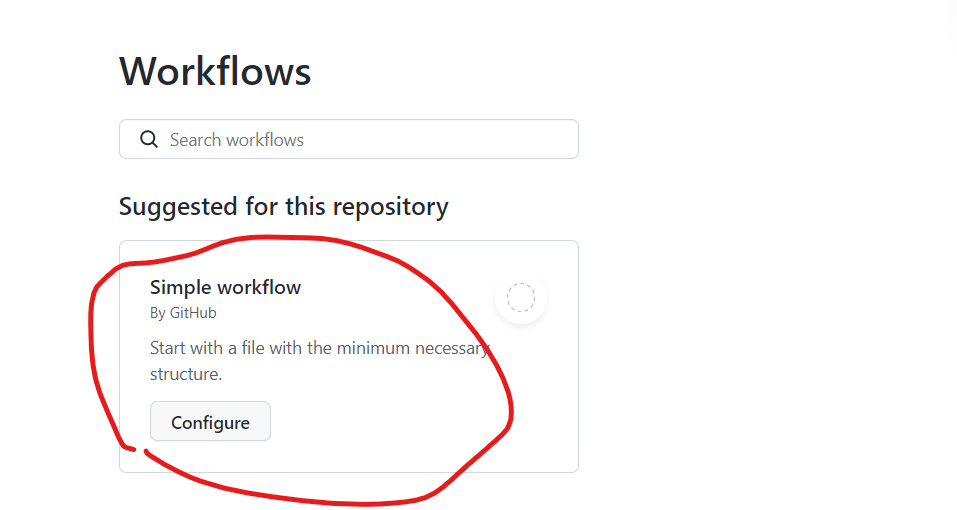
Create a github repository that we are going to use for storing source code files. Below I created GitHub a repository and the URL is <https://github.com/Chittiinfo/githubact-coit-frontend.git>

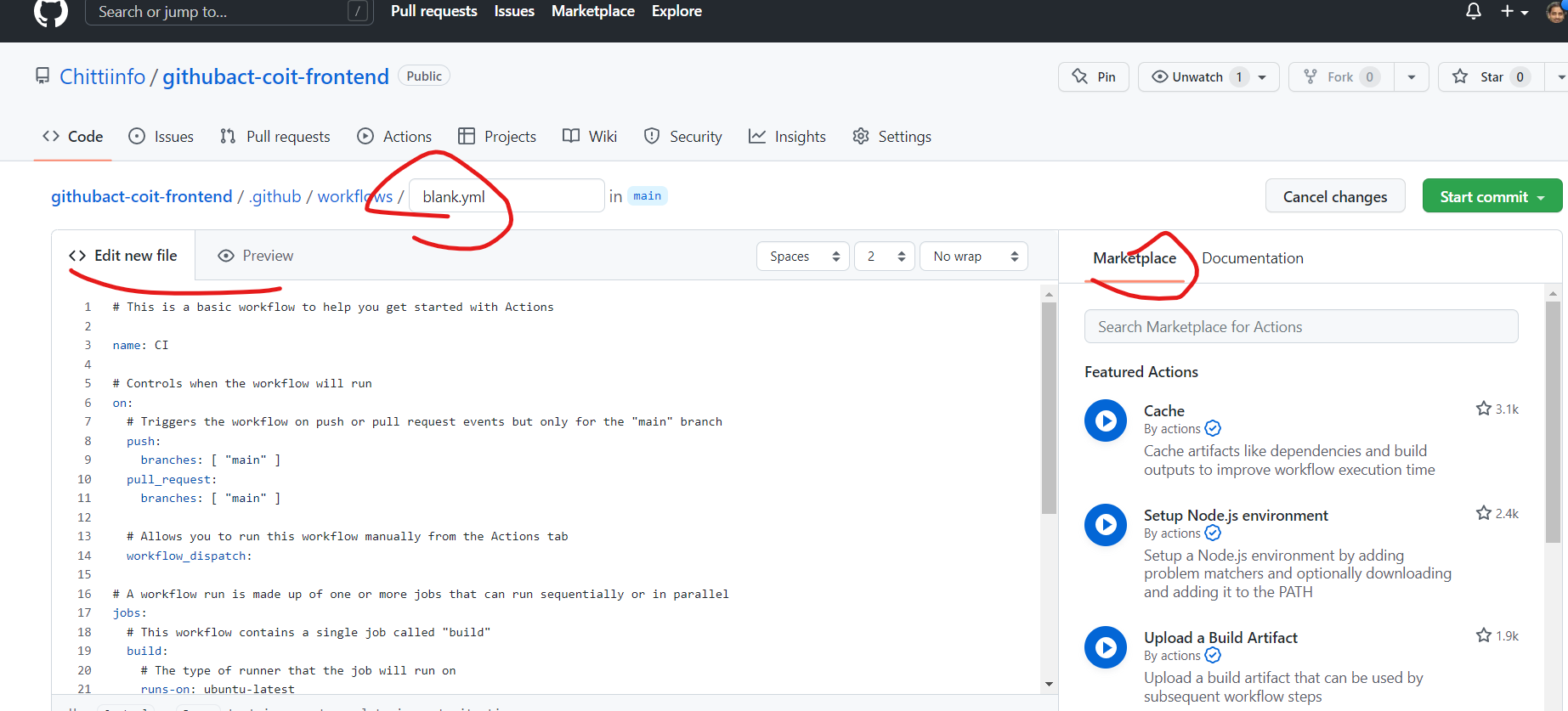




If you want you can create/generate CI/CD workflow template from directly github account or you can use existing workflow files from demo project <https://github.com/Coit-IO/coit-simple-microservice>







**Git Work:**

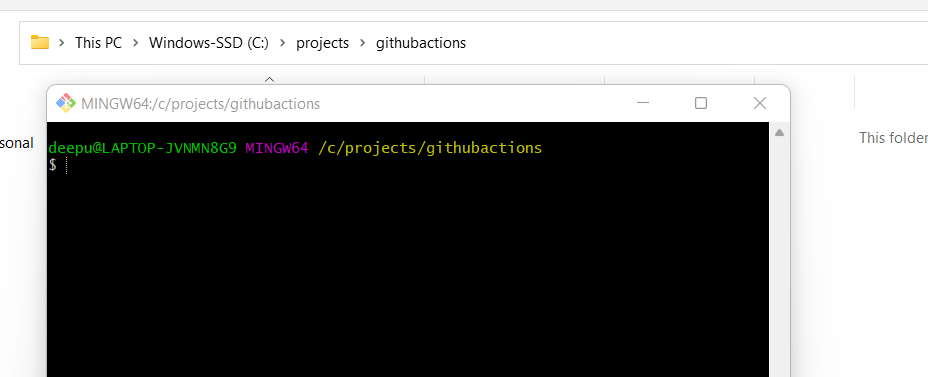
**NOTE:** You can pull the orginal code from<https://github.com/Coit-IO/coit-simple-microservice> and make the necessary modifications or else you can download code from my github repository that’s already modified as per this project requirement

<https://github.com/Chittiinfo/githubact-coit-frontend.git>

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

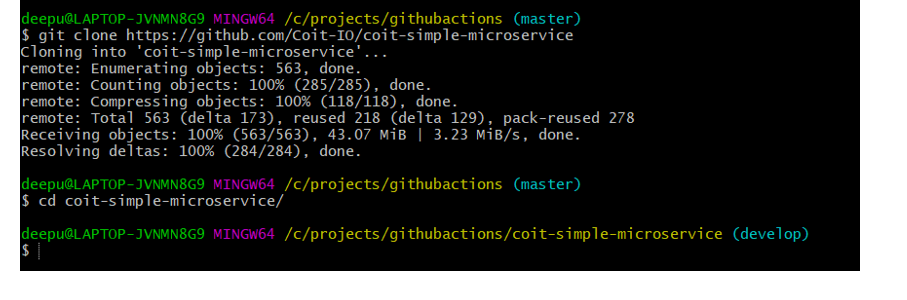
**Git Setup: Follow below link to install Git client on your machine.**

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

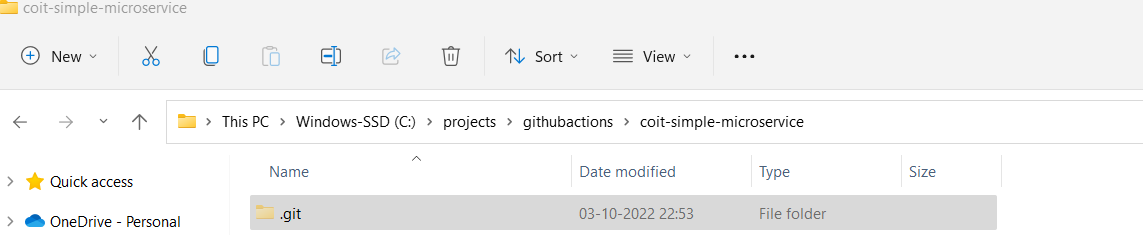


git clone <https://github.com/Coit-IO/coit-simple-microservice>

cd coit-simple-microservice/



Delete .git folder if you want to track only your commits and ignore all previous commit history.

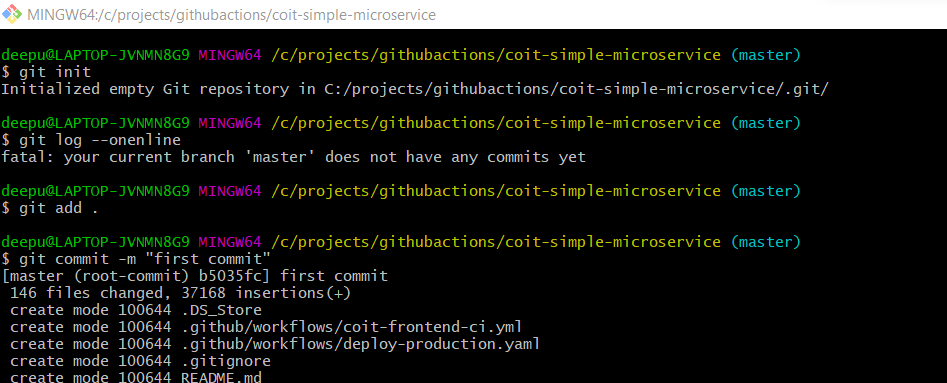


git init

git log --oneline

git add .

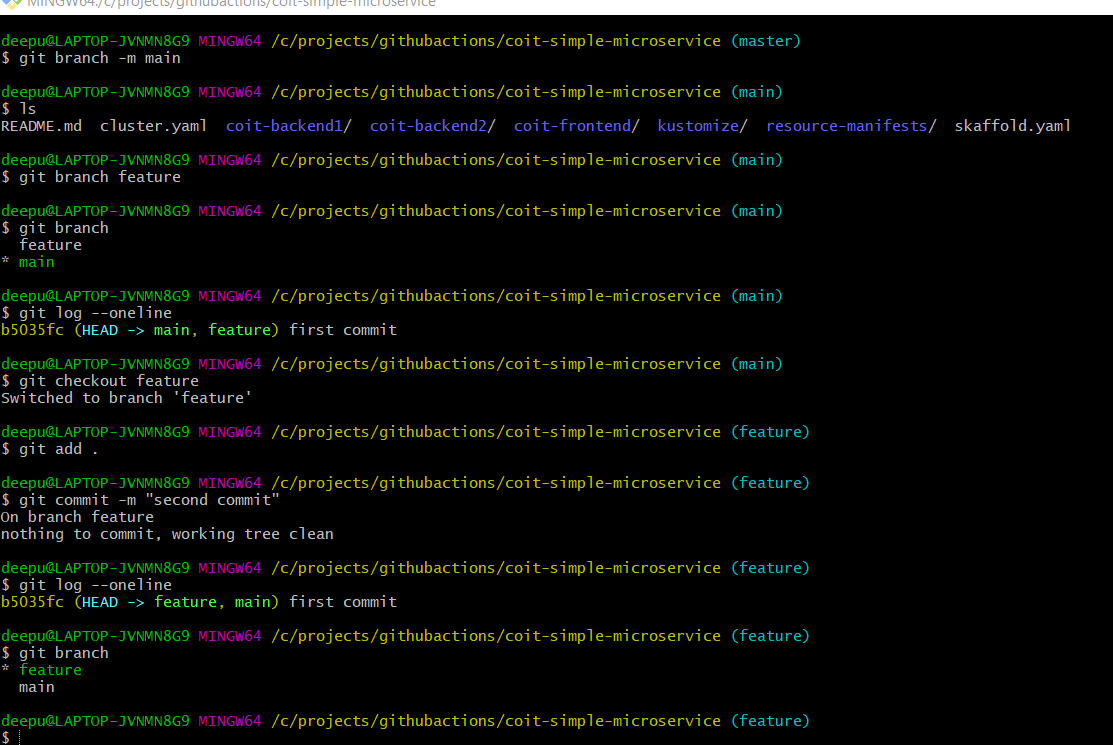
git commit -m "first commit"



git branch -m main - To rename current branch to main

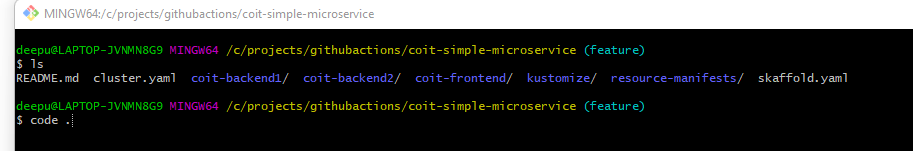
git branch feature

git checkout feature

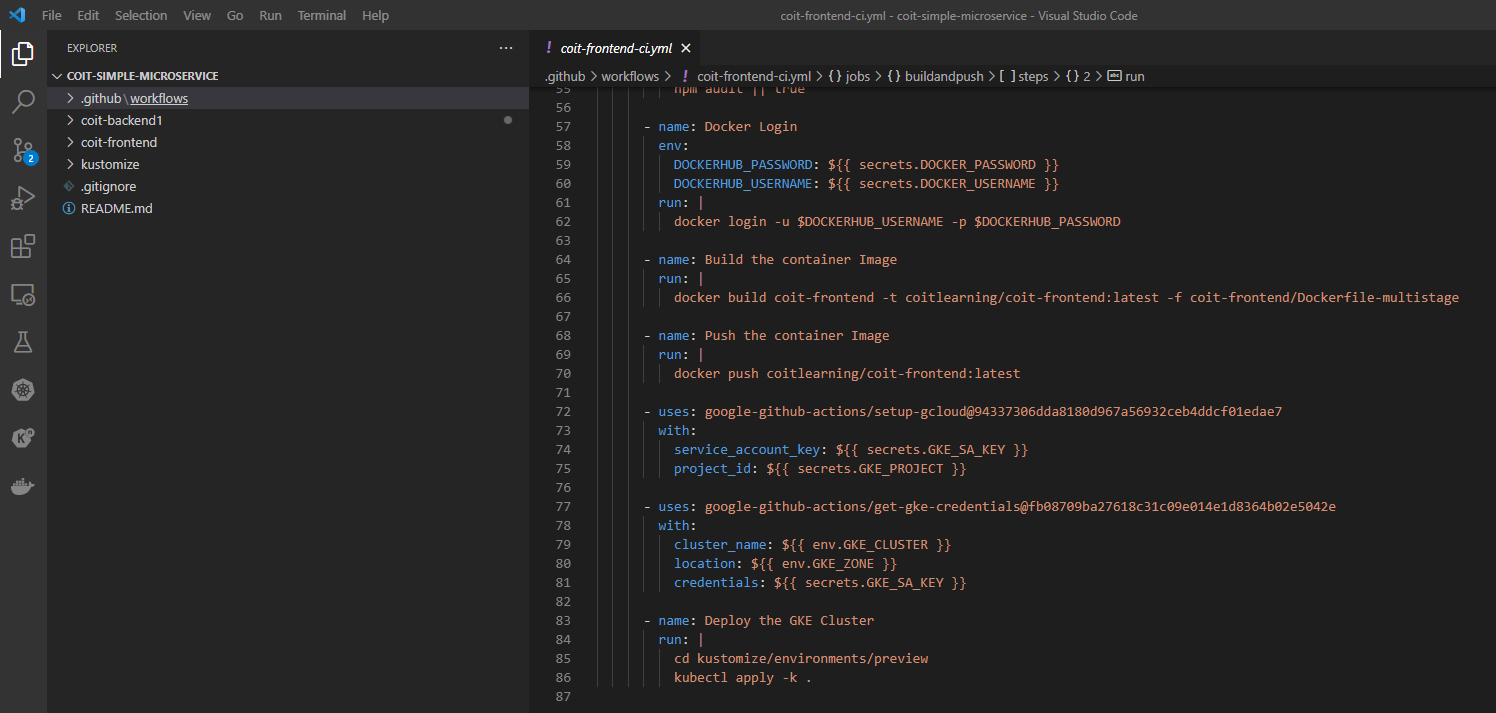


Once downloaded the code from <https://github.com/Coit-IO/coit-simple-microservice> repository delete the unnecessary files and make sure we have only files which are related to Coit Front End application as we just want to deploy front end application into k8s in this project. Once cleaned up unnecessary files/folders then commit changes in local git repository.

Use code . command to open code directly in VSC.



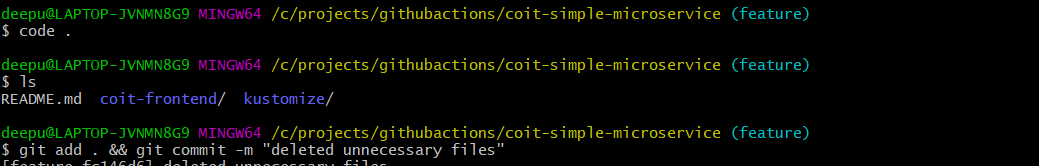
Ensure all the manifest files are modified as per our project requirement and place them in respective folders.



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

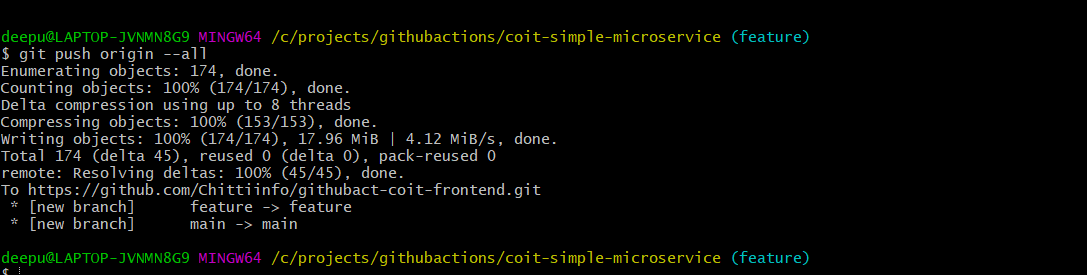
git remote remove origin

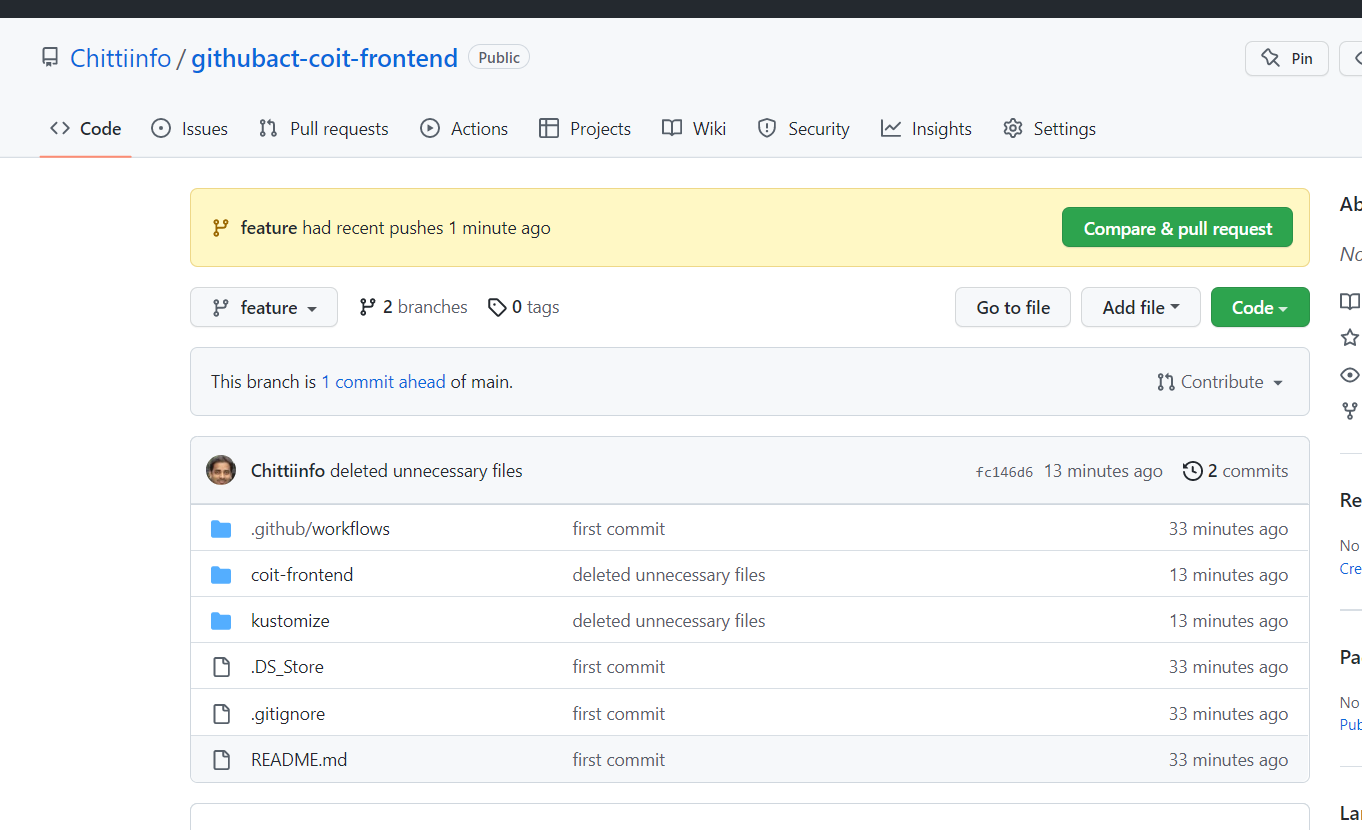
**git config --global user.name "Nag"**

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

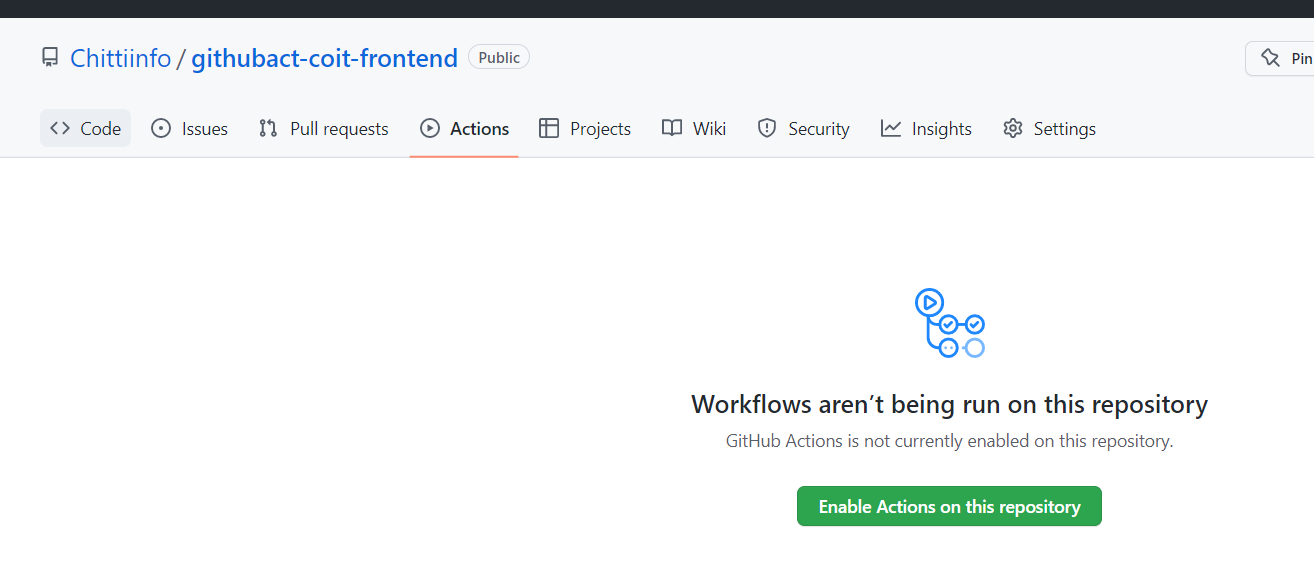
**git remote add origin** https://github.com/Chittiinfo/githubact-coit-frontend.git

**git push origin --all or** git push --set-upstream origin feature **---- To push the code to remote repository.**

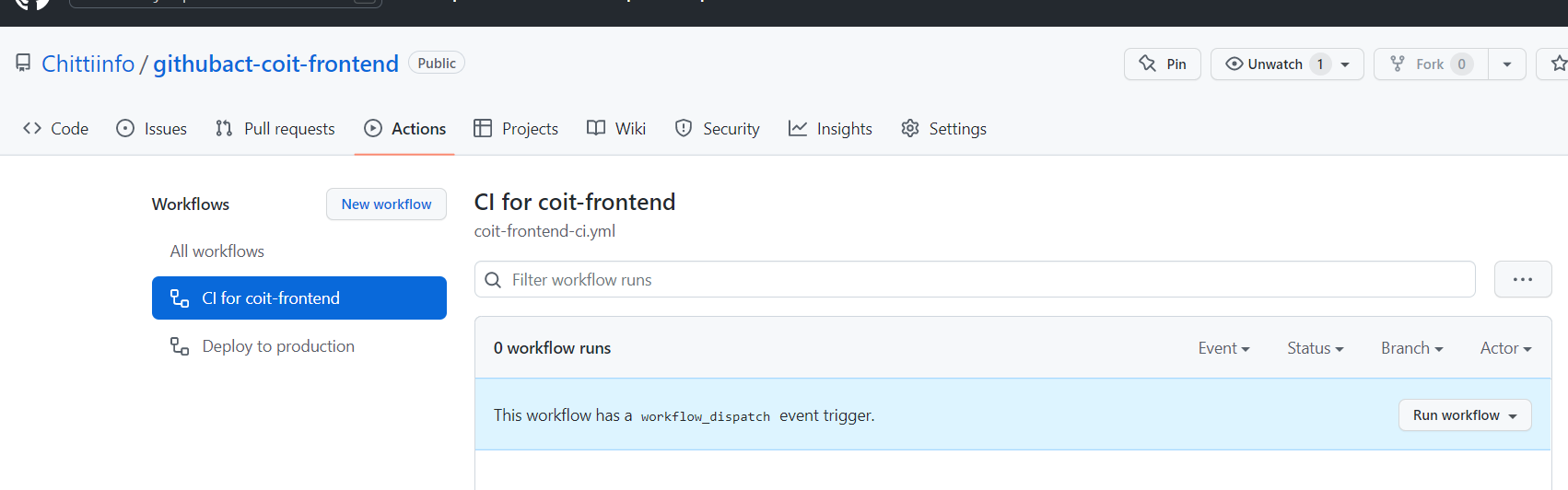
****

****

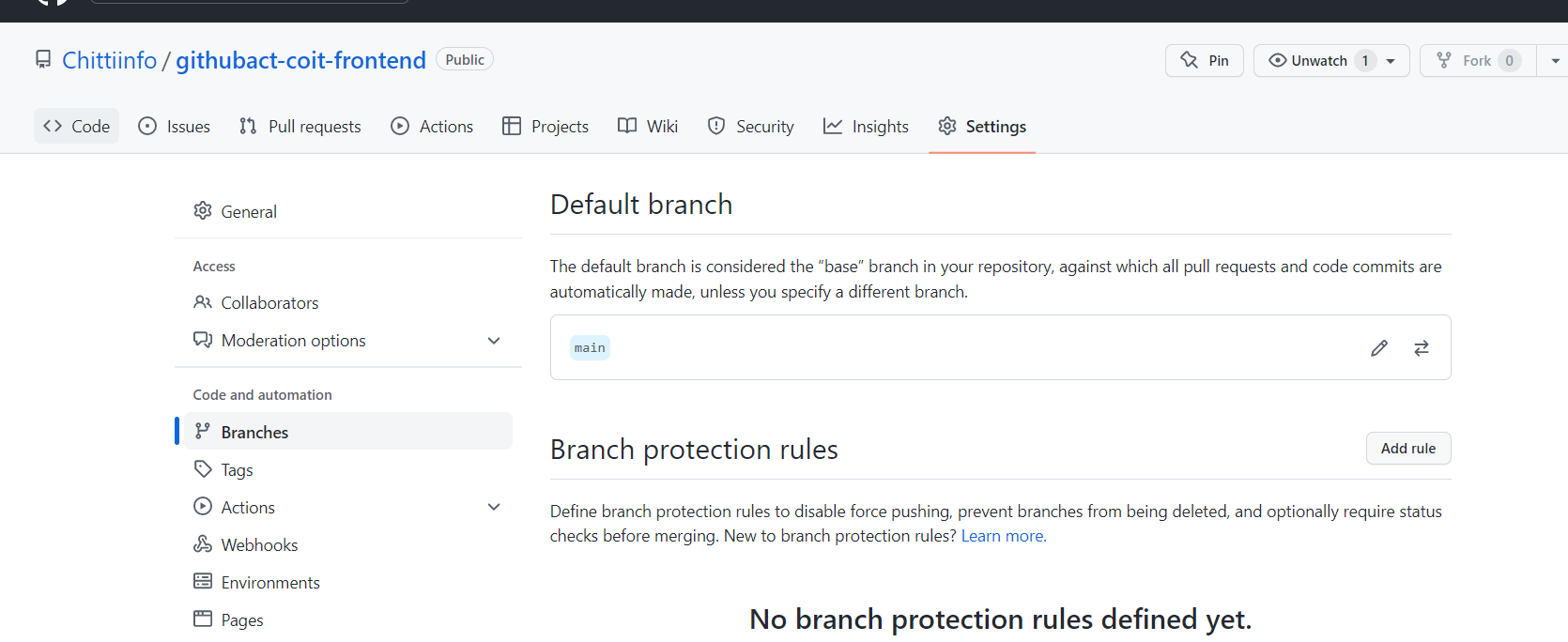
**Enable actions on this repository**

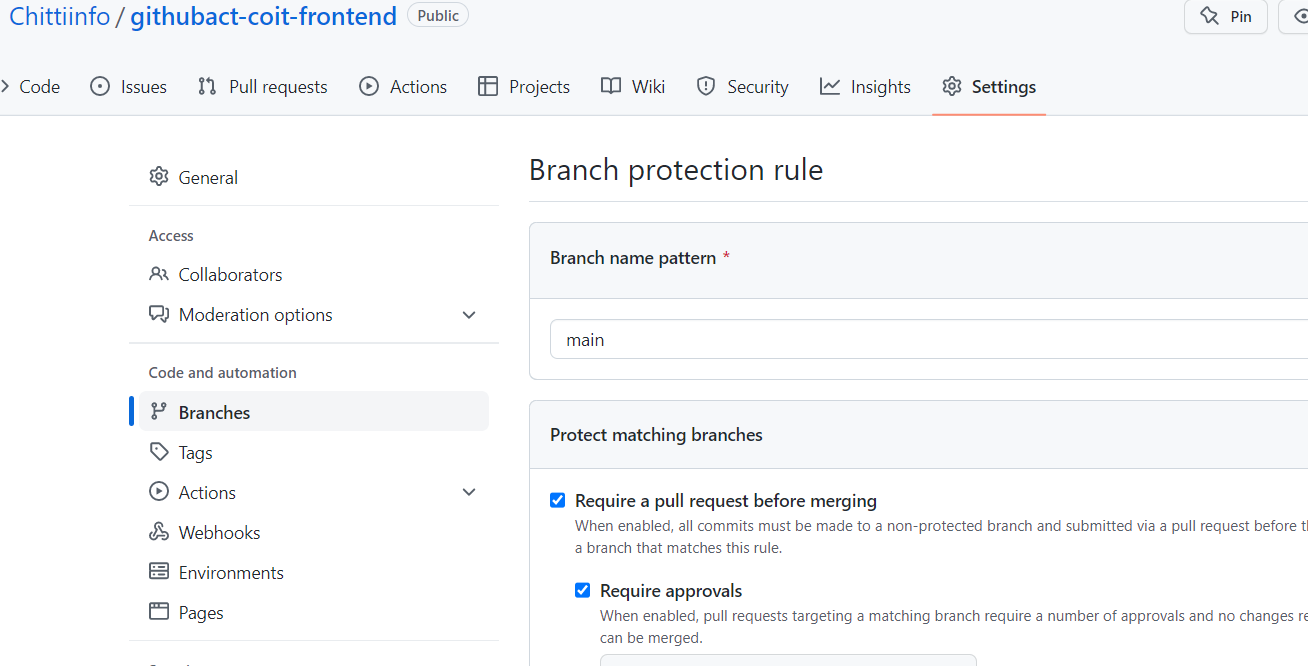
****

**You can notice below workflows that we included in the code**

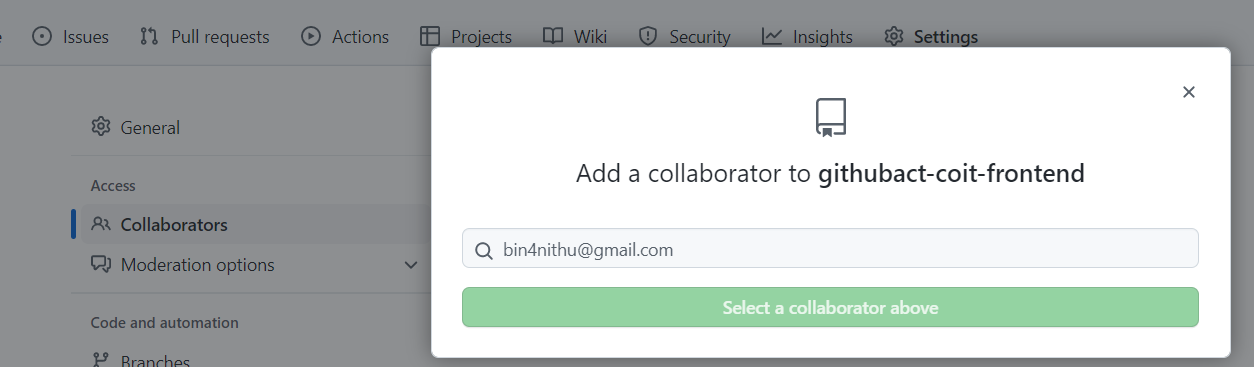
****

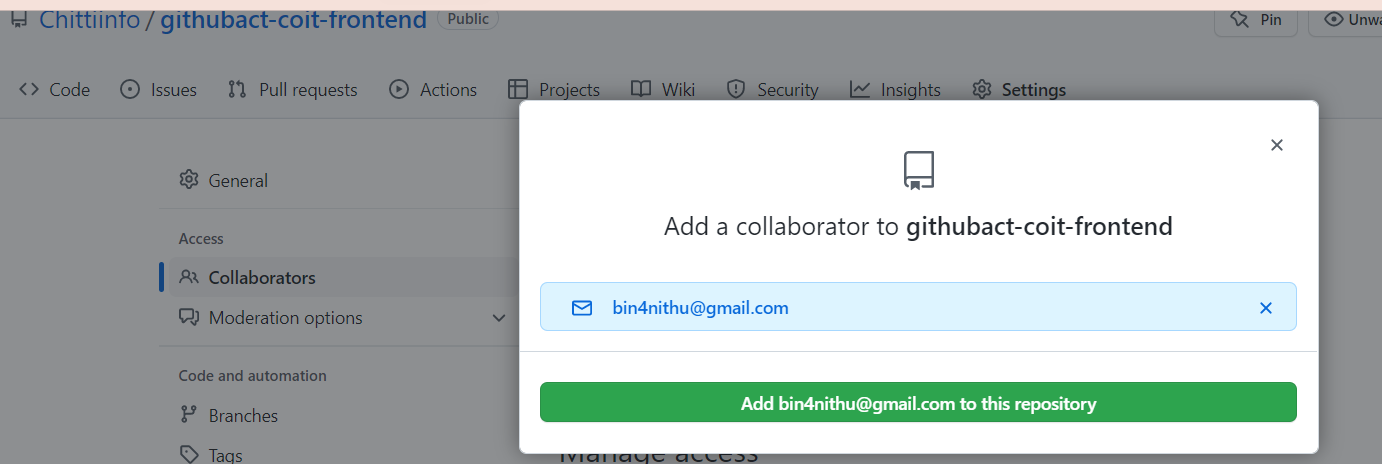
**Add branch protection rules: As we don’t allow developers to commit codes into main branch directly we need to create below branch protection rule on main branch**

****

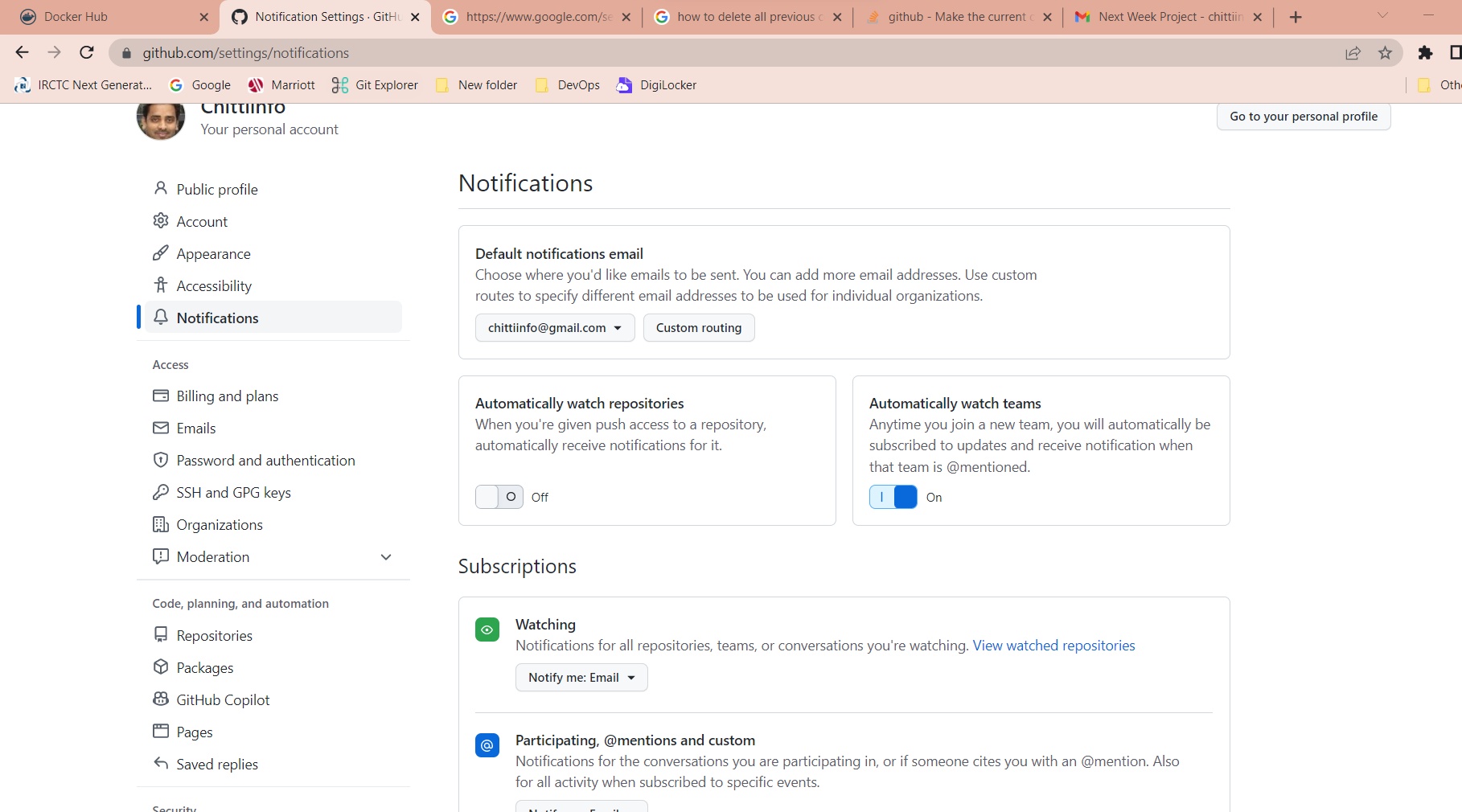
****

**We can add required collaborators to repository.**

****

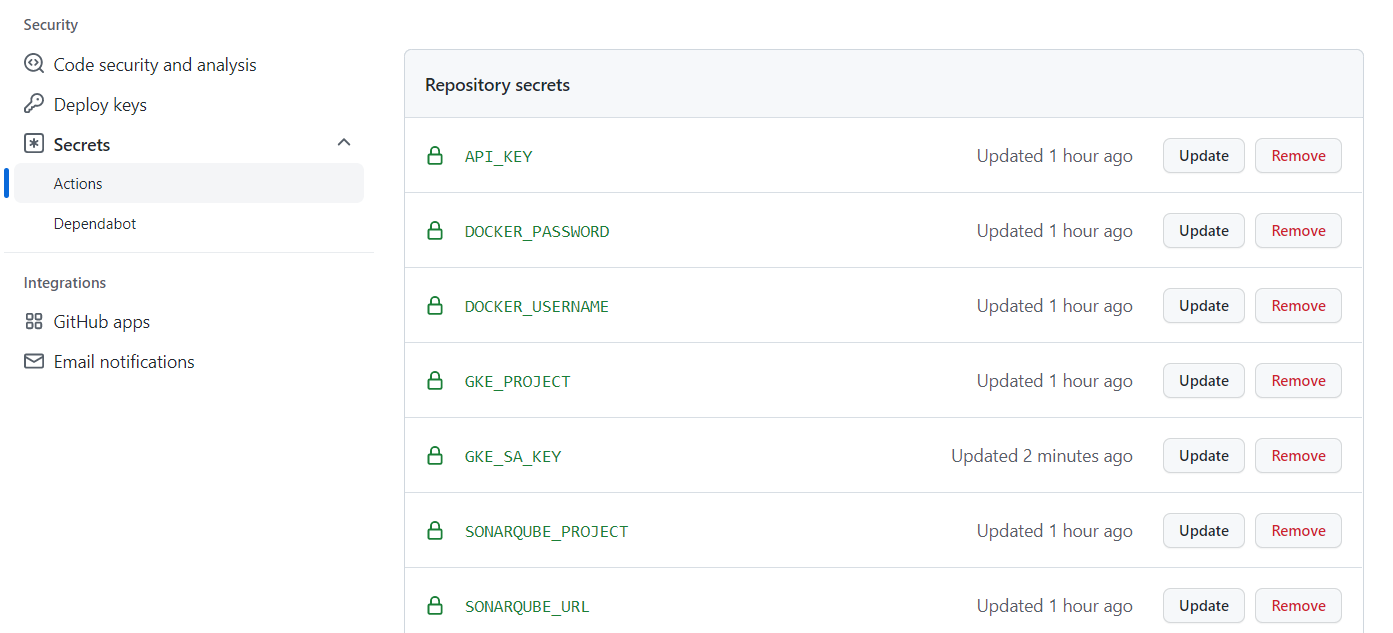
****

**You can manage all mail notifications from here**

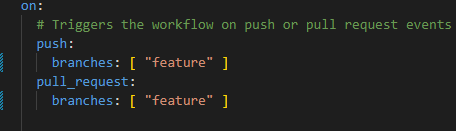
****

**Developing workflow for CI:**

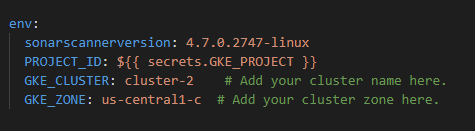
* **Store all the required secretes in Github Secrets section**

****

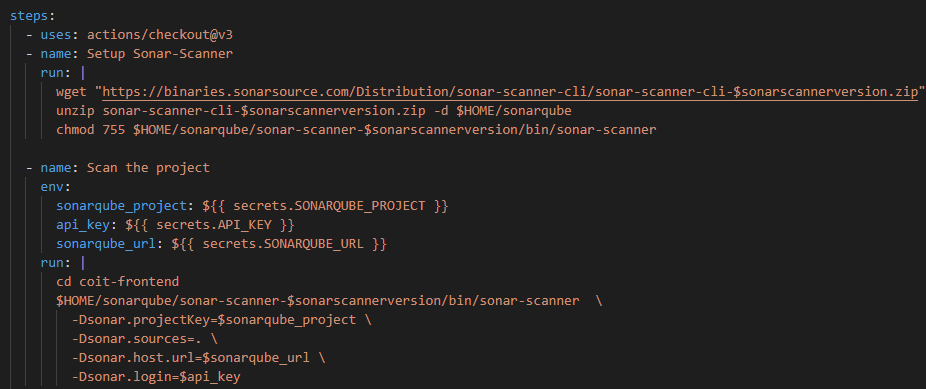
* **Change push\pull\_request branch name to “feature”**

****

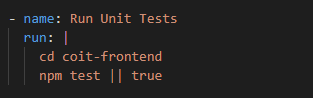
* **Define required environment variables.**

****

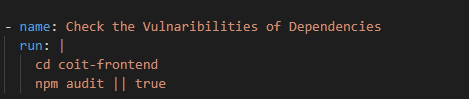
* **Install Sonar scanner client on runner then scan the source code and upload results in Sonar Server**

****

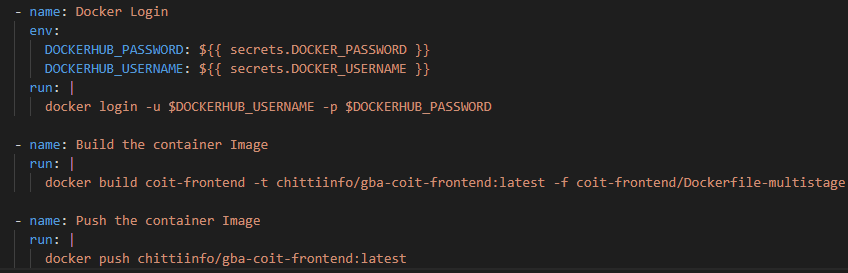
* **Run unit test**

****

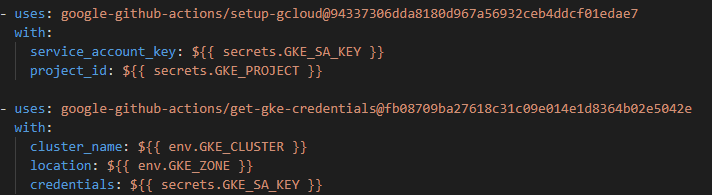
* **Check Vulnerabilities of Dependencies**

****

* **Login to your docker account and build & push image to your docker repository.**

****

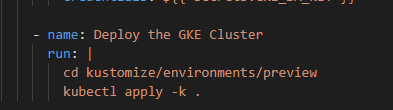
* **Connect to gcloud and GKE**

****

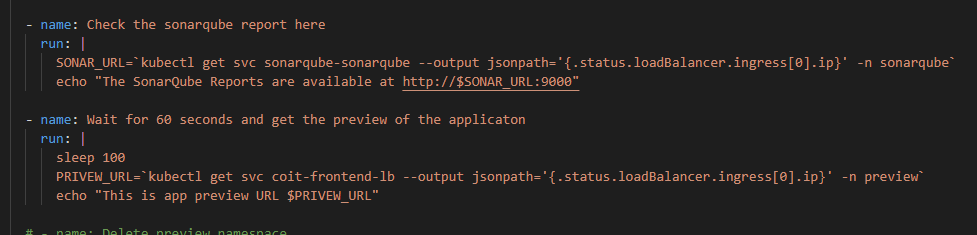
**Follow below guide to Create a service account that we can use in our CI to authenticate with GKE**

[**https://docs.github.com/en/actions/deployment/deploying-to-your-cloud-provider/deploying-to-google-kubernetes-engine**](https://docs.github.com/en/actions/deployment/deploying-to-your-cloud-provider/deploying-to-google-kubernetes-engine)

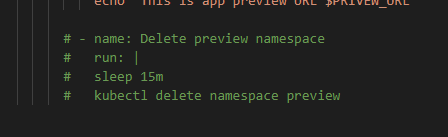
* **Deploy application in GKE cluster with kustomize.**

****

* **To show sonar qube URL and Preview app url**

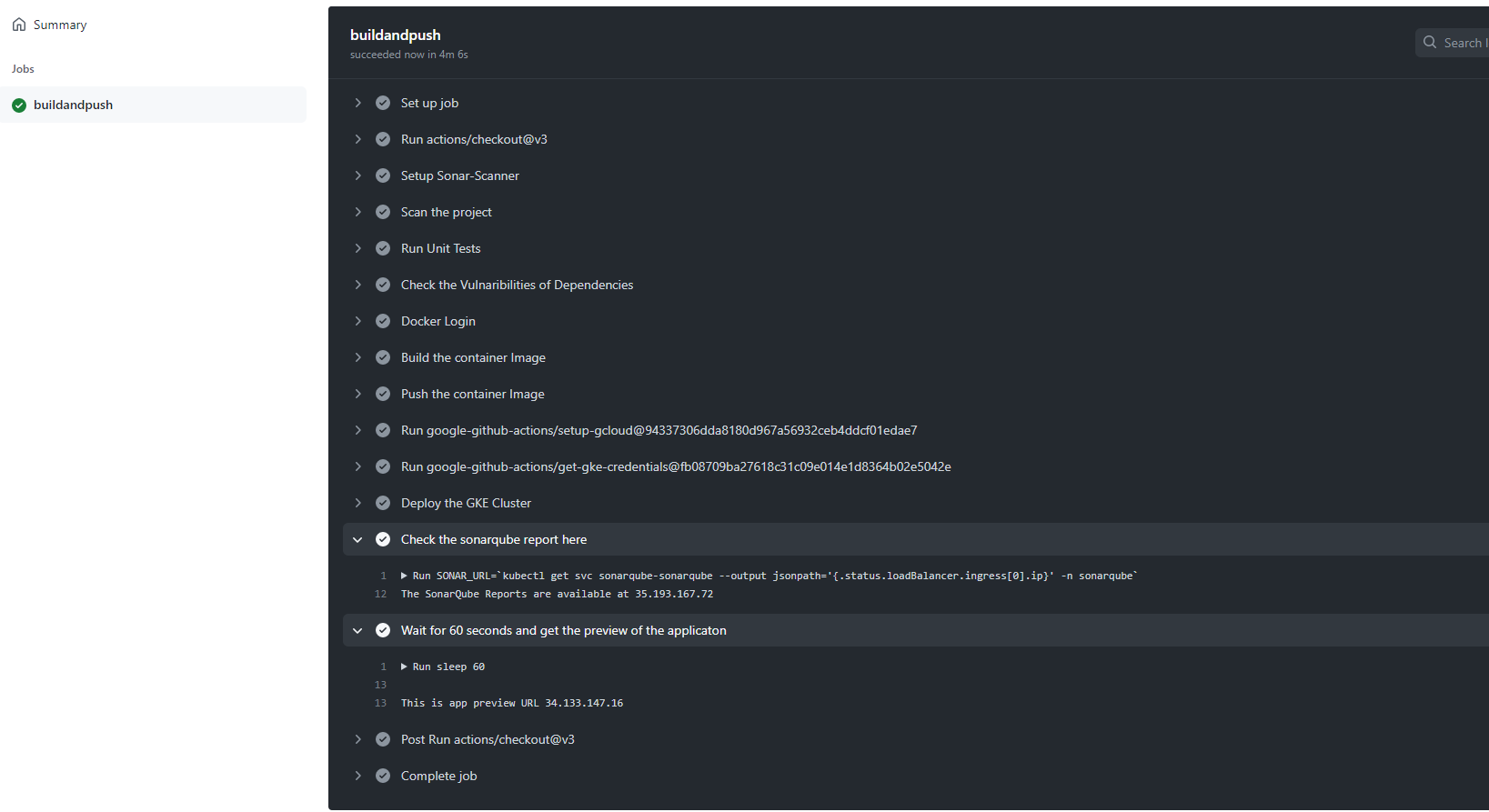
****

* **Delete name space after 15 or 30 min.**

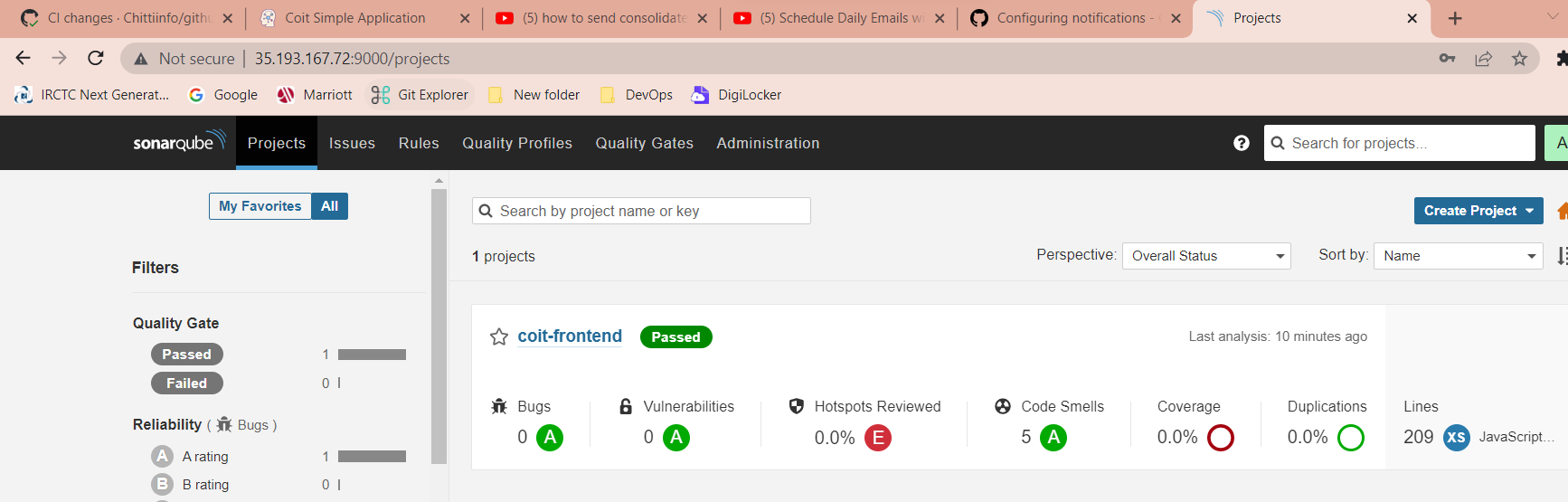
****

**Validation:**

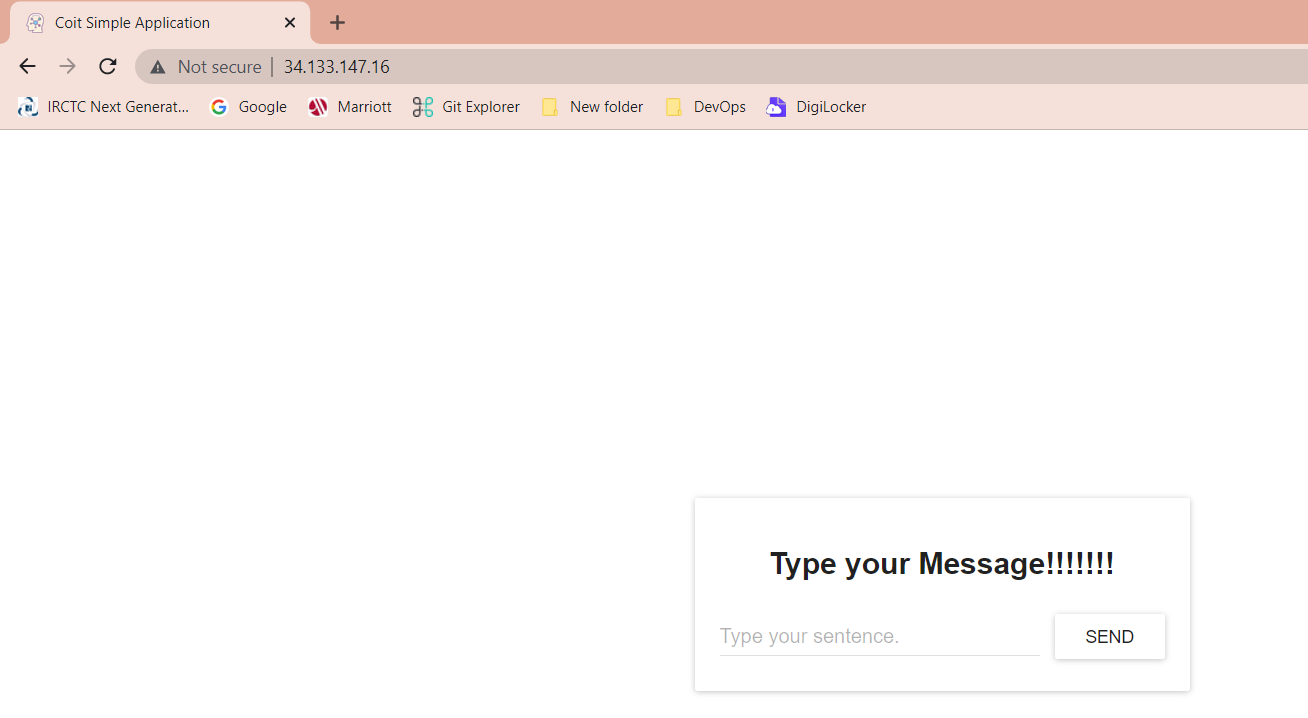
**CI job is competed successfully**

****

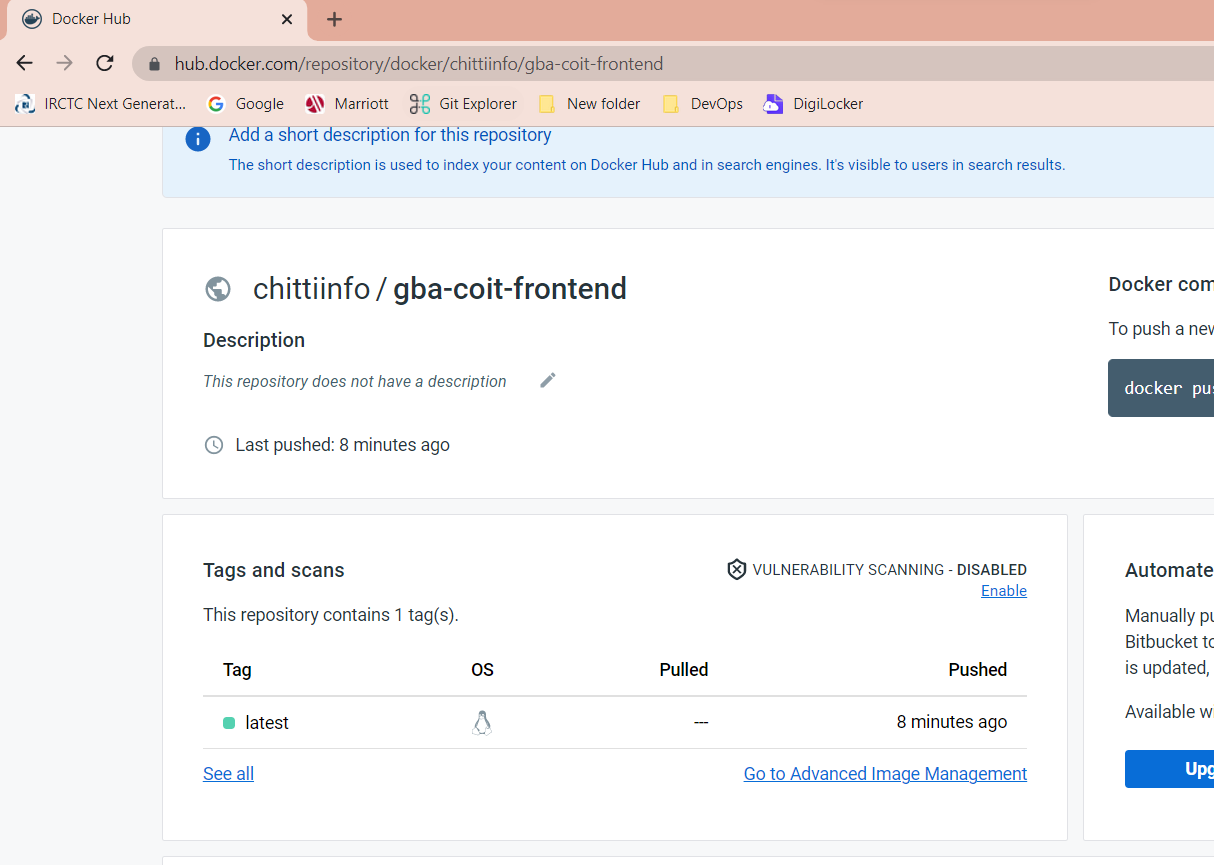
**Code scan results are reported at Sonarqube**

****

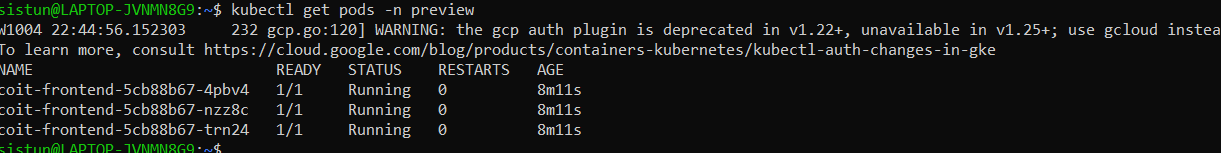
**Front end app is accessible from browser**

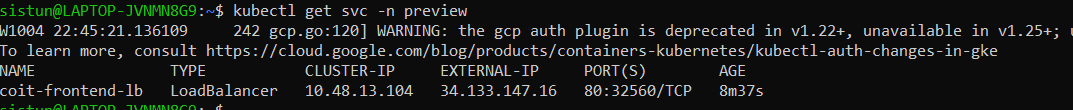
****

**Image is pushed to my docker hub repository**

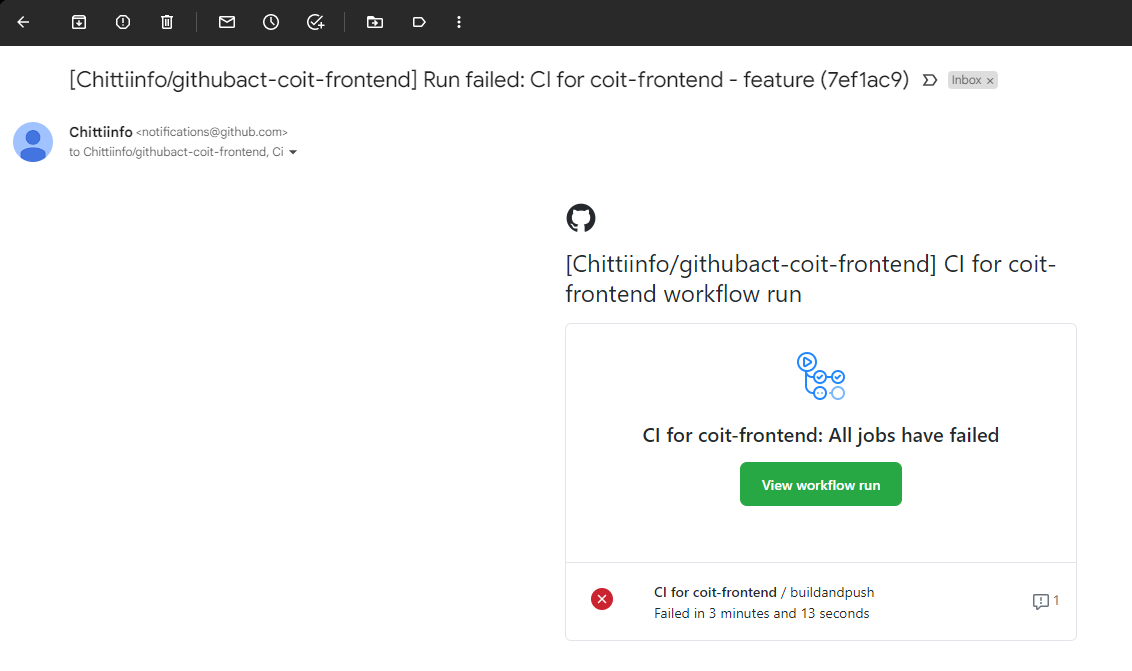
****

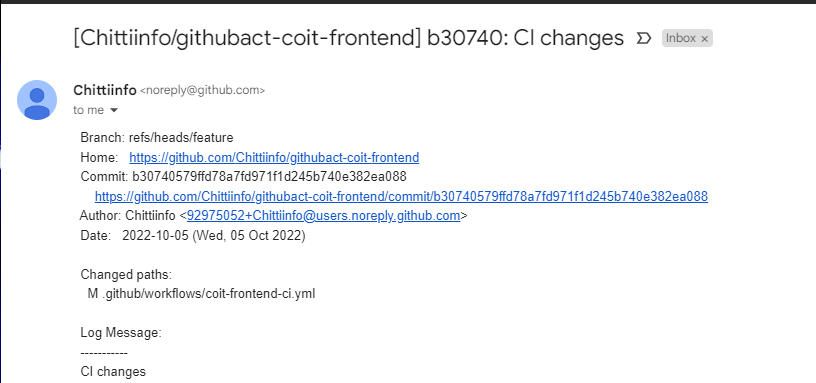
**Frontend app Pods & services are running fine**

****

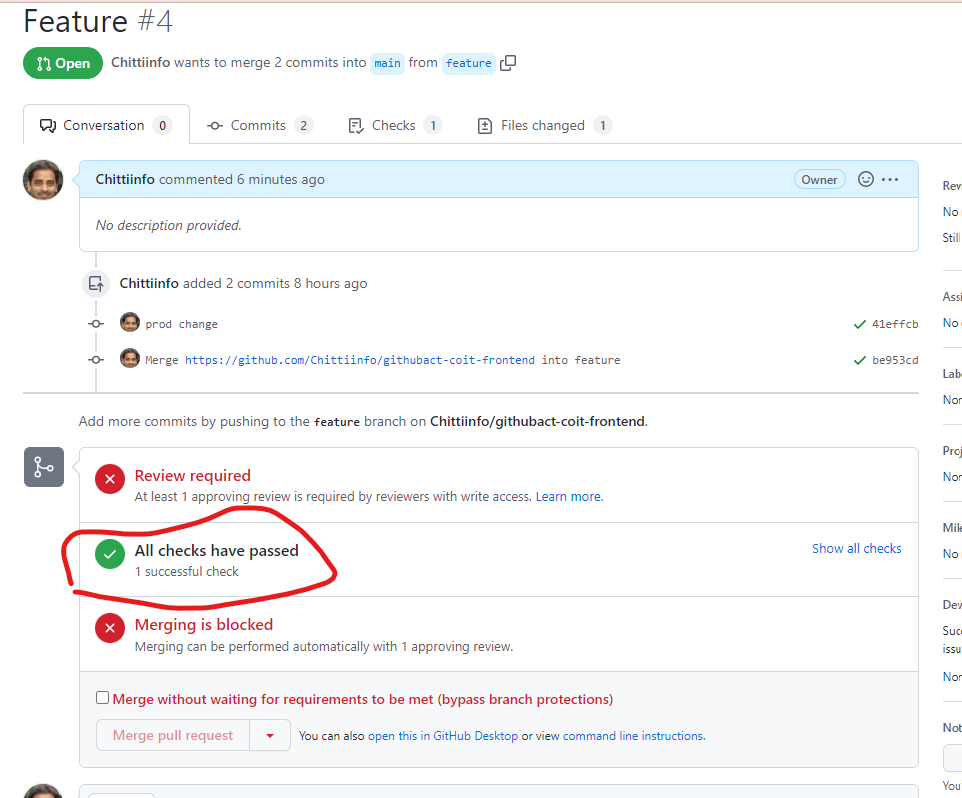
****

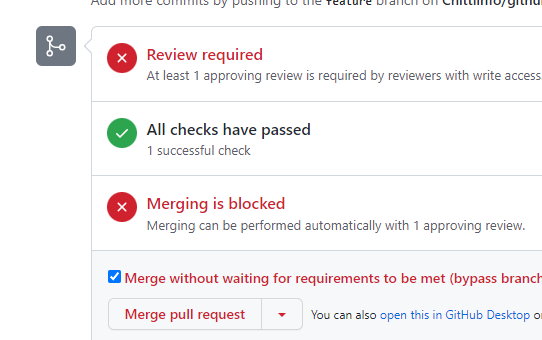
**If any failure in CI/CD process or commits/pushes I am receiving eamils.**

****

****

**Once CI workflow is successfully run and if all checks are passed then we can merge feature branch pull request to mainline branch.**

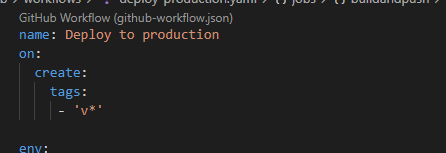
****

****

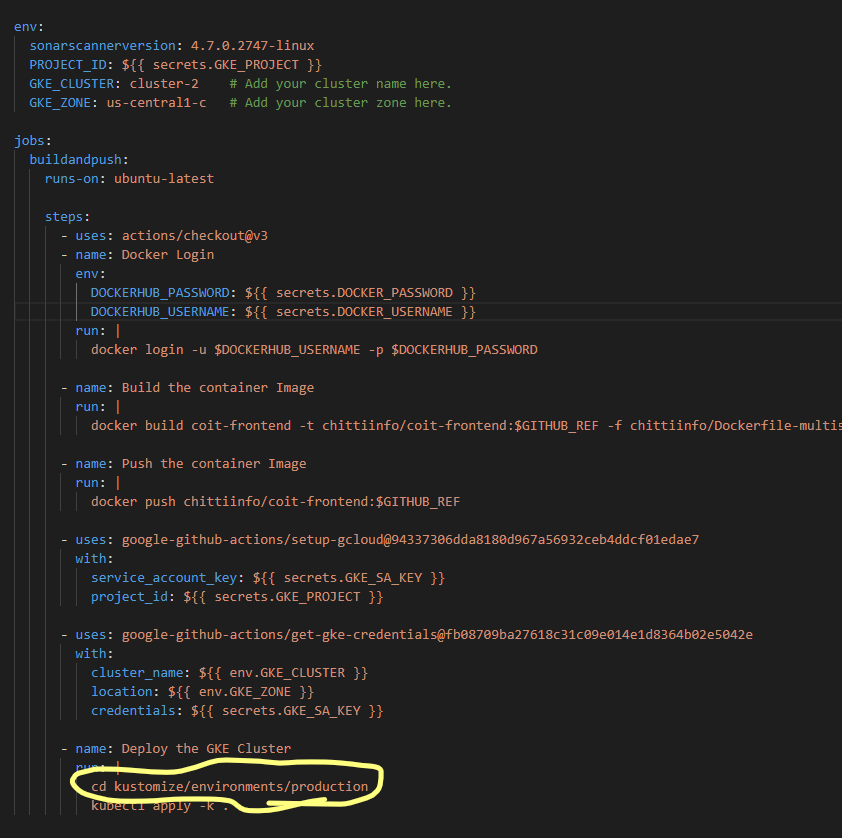
**Now both Feature branch and mainline branch have the same code and its stable. Now lets develop workflow yml file for Production enviorment. Mainline branch code will be deployed when there is a tag creation.**

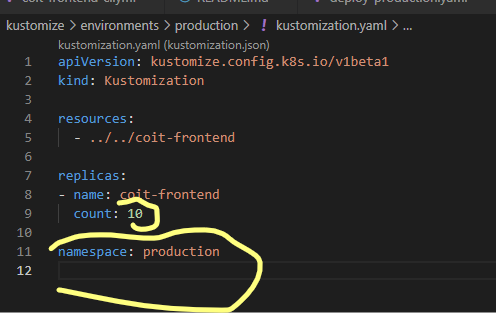
**Develop Production workflow yml file:**

As we want to trigger production workflow when there is tag creation we included V\* in yml file



As we are just deploying stable image to production environment all the remaining code will be same as preview environment except few changes. K8s namespace will be production and need to create 10 replicas with kustomization.

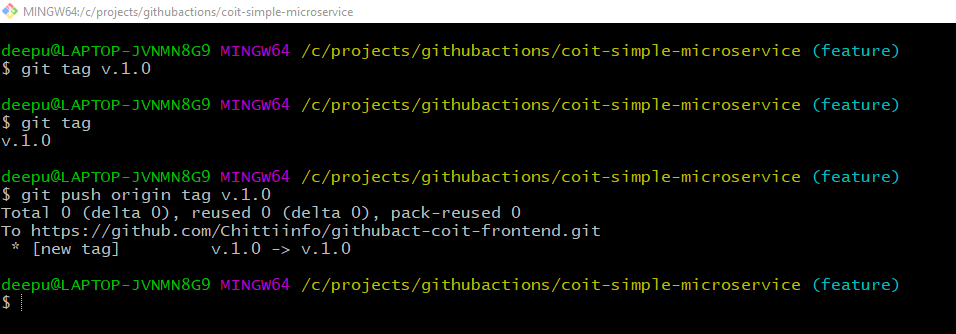




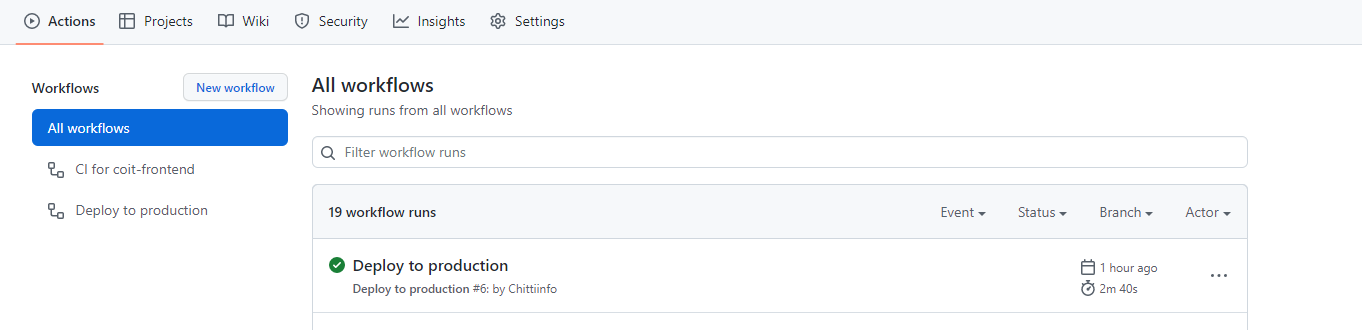
Commit latest changes to remote repositor then run below command to create a tag on local and remote repository.

Git tag v.1.0

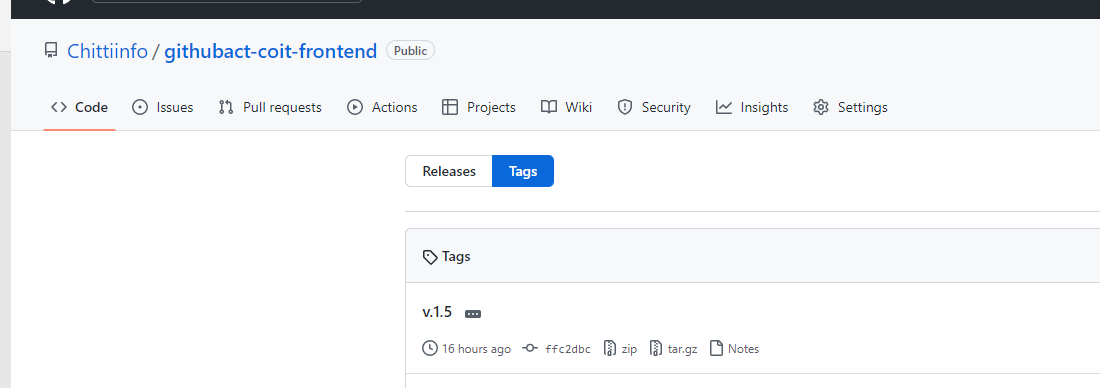
git push origin tag v.1.0



As soon as above tag is pushed to remote repository it triggered production workflow and its successful.

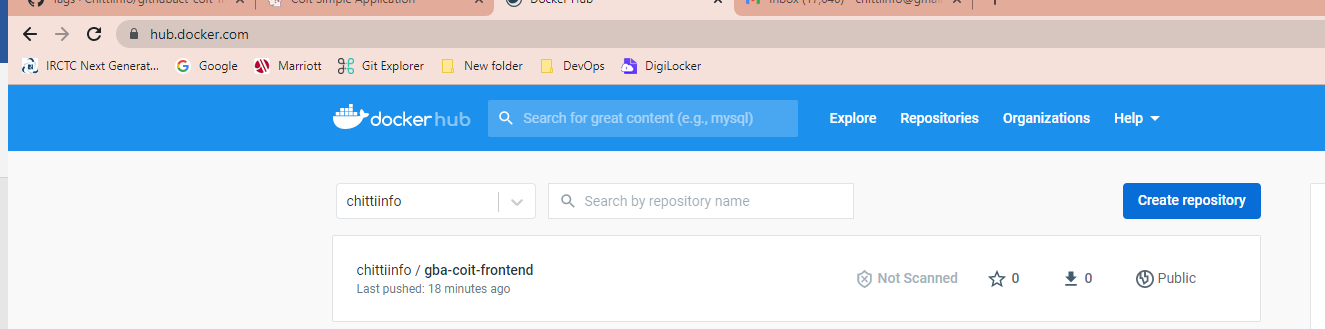


Even tag is also created in my remote repository.

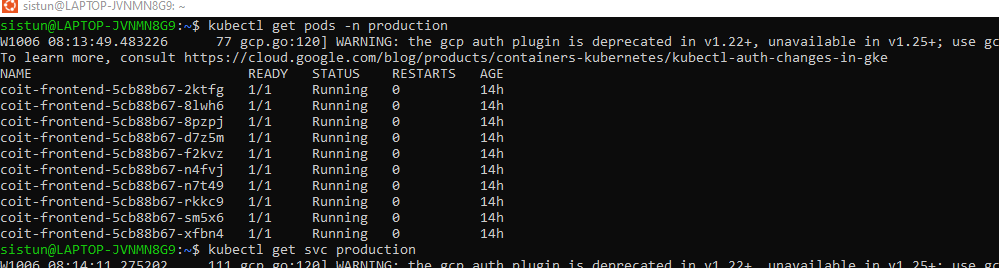


**Validation of production environments:**

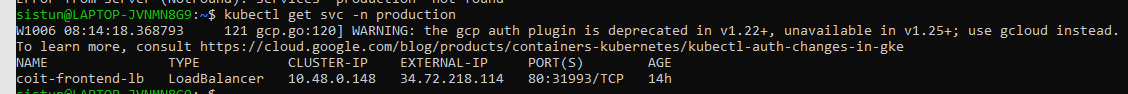
**Production workflow built and pushed image to docker hub.**

****

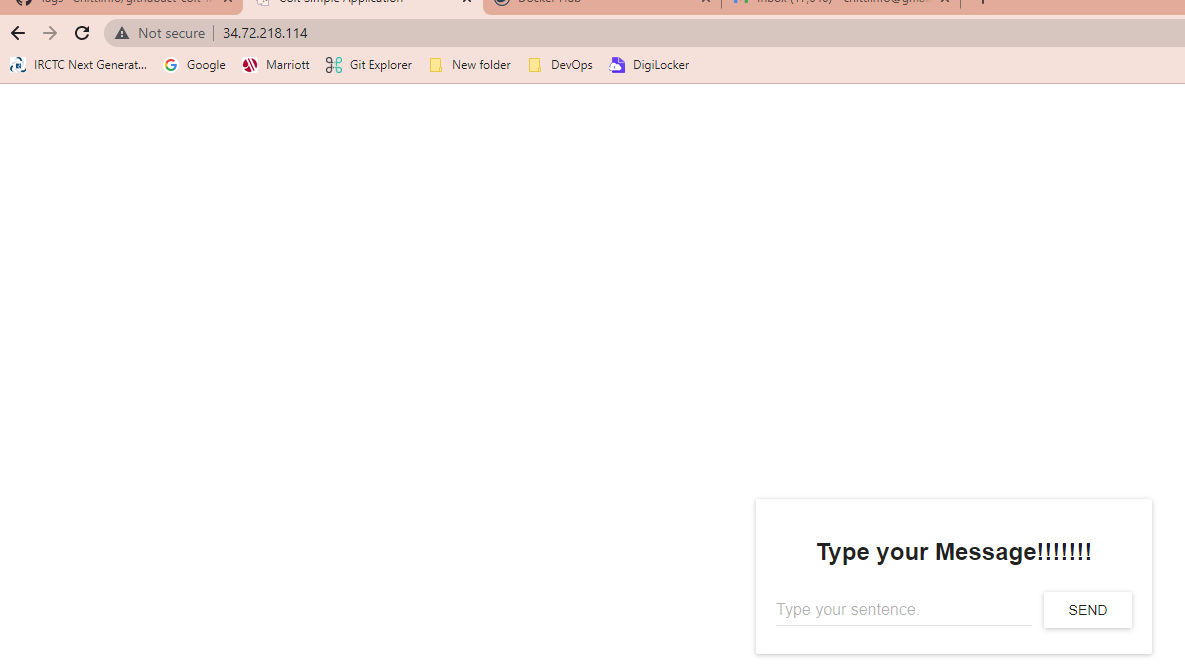
**It created total 10 coit-frontend pod replicas**



It also created a service with external IP.

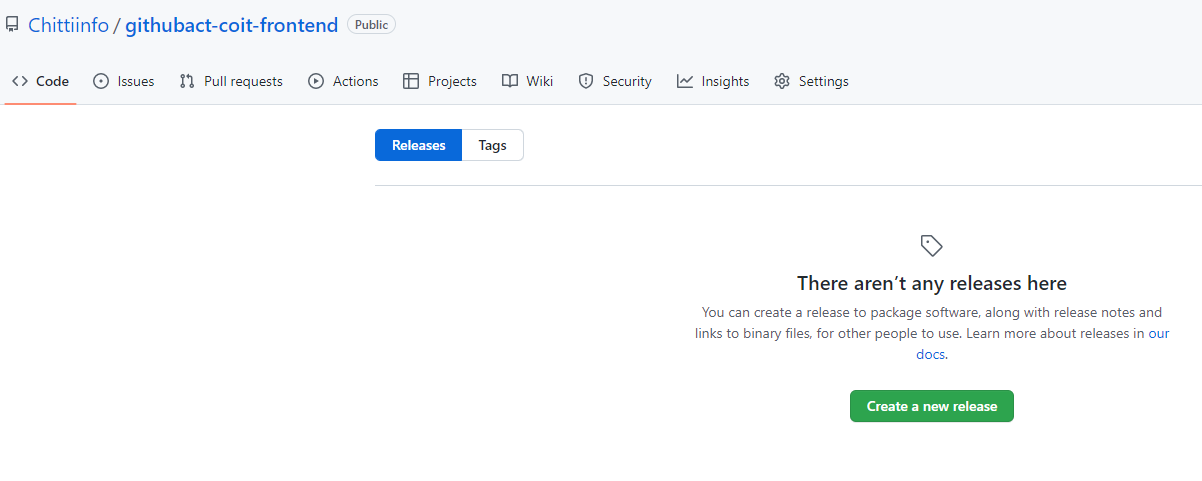


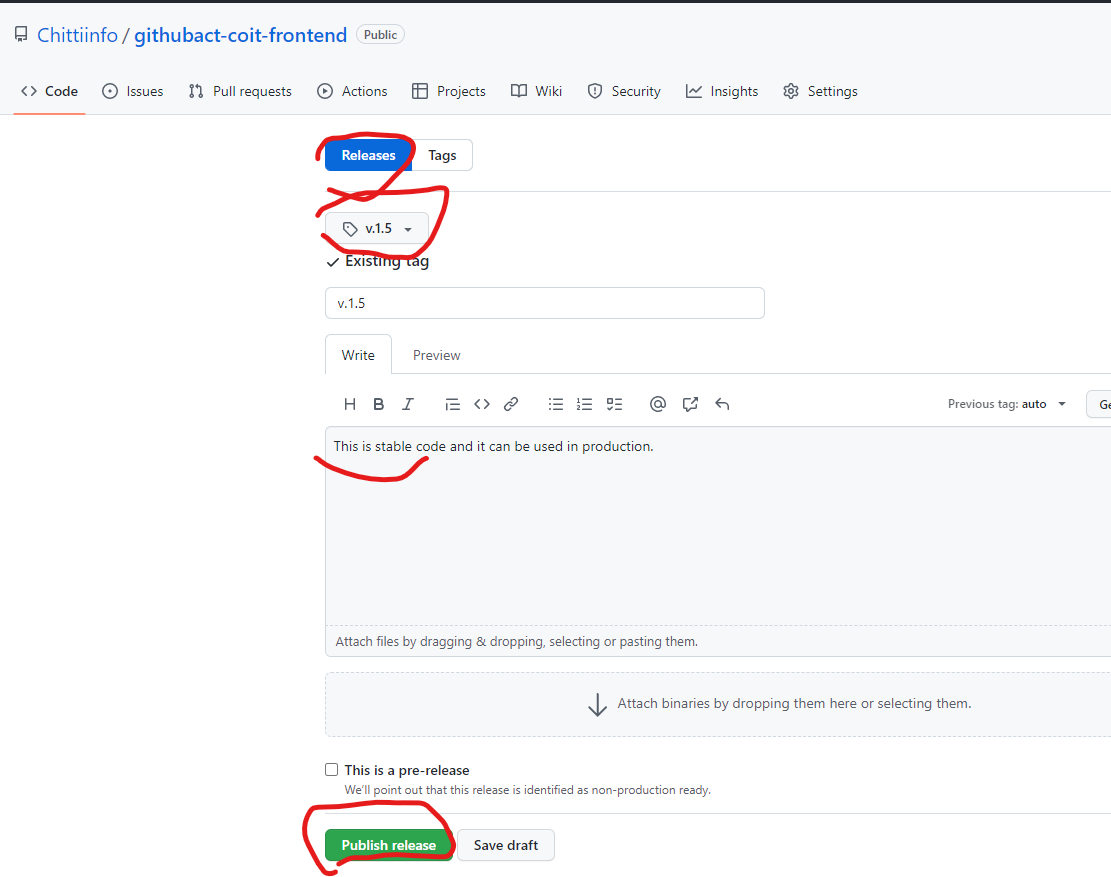
Production environment front app is accessible with service IP.

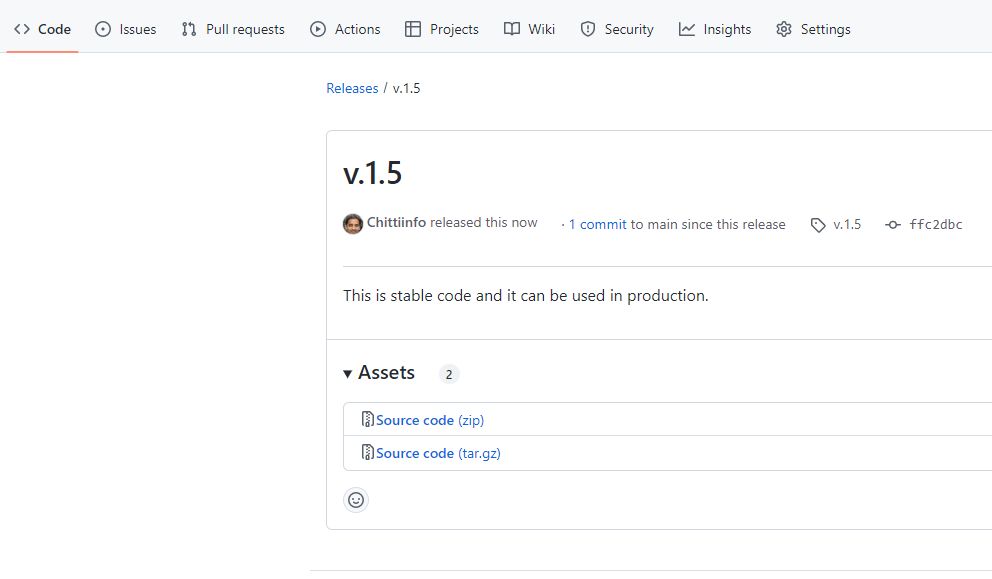


As we have no issues with production work flow and all checks are passed we can create a release for this stable code.

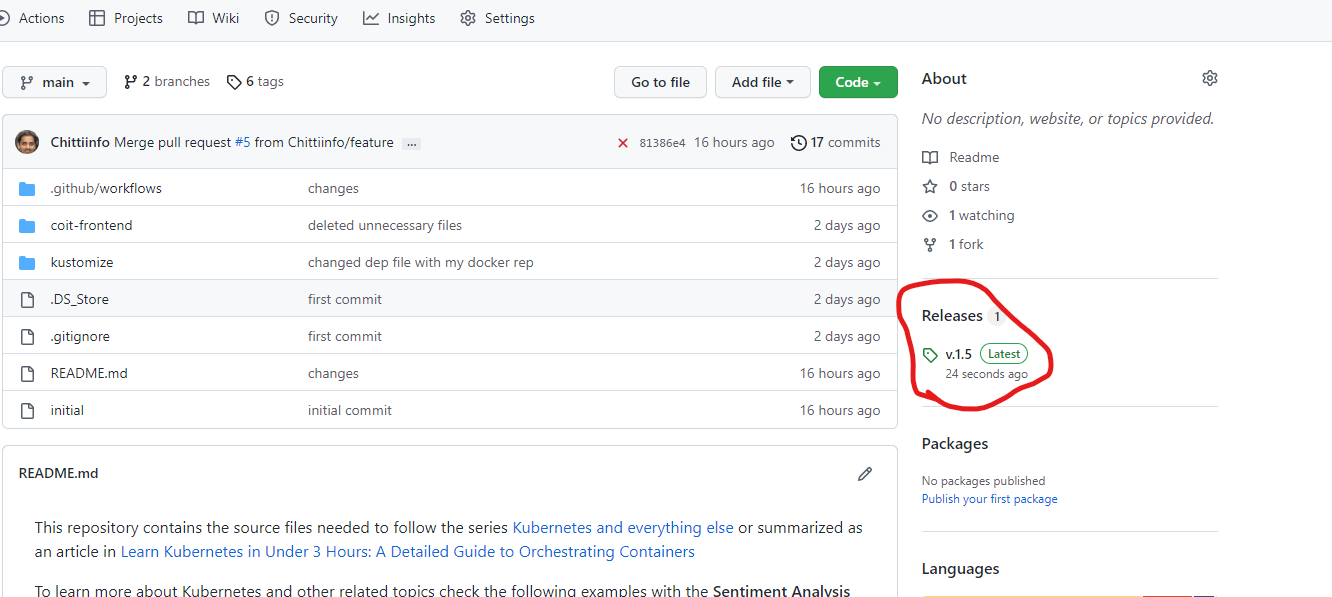
Follow below steps to create a release.







As release is created now anyone can download this code and use it anywhere.



**NOTE:** Below items are not documented in this doc as I was not able to find proper solution when I prepared this doc. I will update this doc when below items are addressed.

* Number of Issues by Sonarqube
* only changed files should be scanned by Sonar qube
* write a script that can check document and if there are any broken URLs in document CI job should be failed.