Deploying a YouTube Clone App with DevSecOps and Jenkins Shared Library

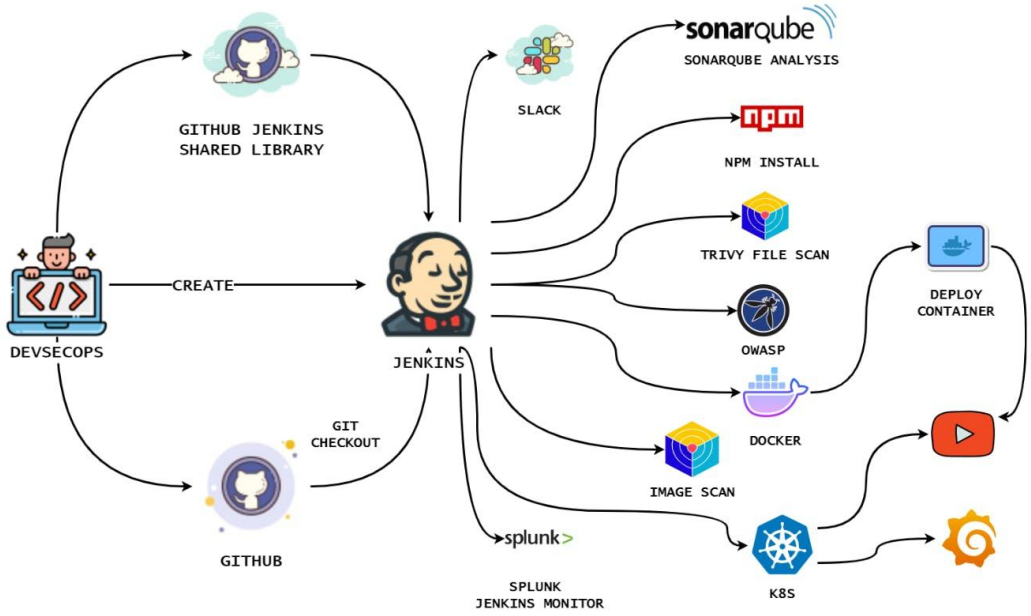


Table of contents

Step5A: Start Job

Step5B: Create a Jenkins shared library in GitHub

Step5C: Add Jenkins shared library to Jenkins system

Step5D: Run Pipeline

Step6: Install Plugins like JDK, Sonarqube Scanner, NodeJs

Step6A: Install Plugin

Step6B: Configure Java and Nodejs in Global Tool Configuration

Step6C: Configure Sonar Server in Manage Jenkins

Step6D: Add New stages to the pipeline

Step7: Install OWASP Dependency Check Plugins

Step8A: Docker Image Build and Push

Step8B: Create an API key from Rapid API

Step8C: Run the Docker container

Step9A: Kubernetes Setup

Step9B: Kubectl is to be installed on Jenkins

Step9C: K8S Master-Slave setup

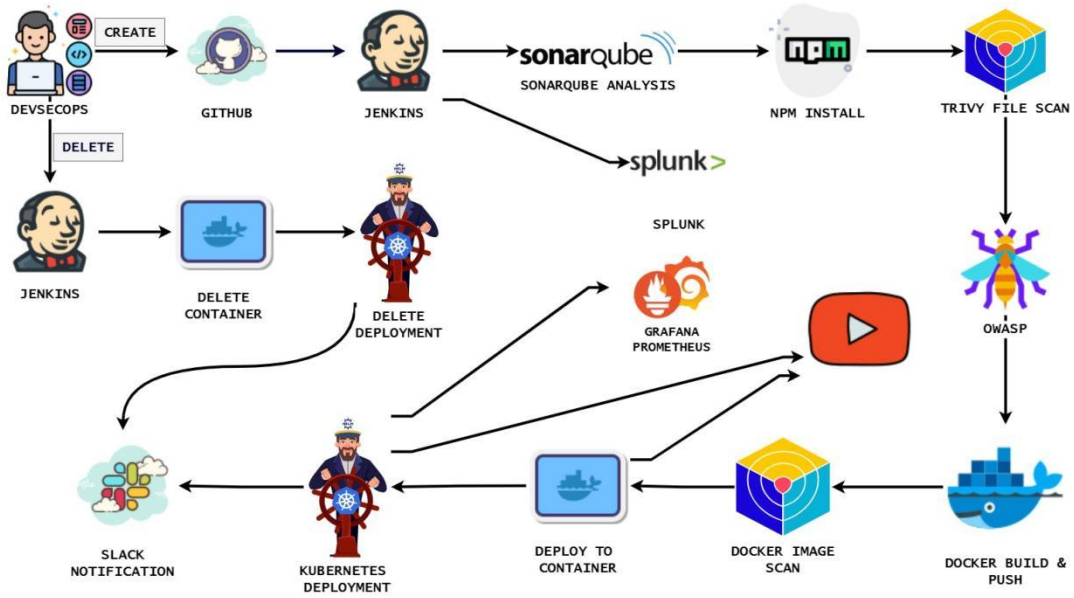
Step9D: Install Helm & Monitoring K8S using Prometheus and Grafana

Install kube-Prometheus-stack

Edit Prometheus Service

Edit Grafana Service

Step9E: K8S Deployment



GitHub Repo: https://github.com/chandrushal/Youtube-clone-app.git

GitHub Repo: https://github.com/chandrushal/Jenkins_shared_library.git

Step 1: Launch an Ubuntu 22.04 instance for Jenkins

Choose an Instance Type: Pick "t2.large."

Create a shell script to install Jenkins service

sudo vi install_jenkins.sh

Past the following script

```
#!/bin/bash
sudo apt update -y
wget -O - https://packages.adoptium.net/artifactory/api/gpg/key/public | tee
/etc/apt/keyrings/adoptium.asc
echo "deb [signed-by=/etc/apt/keyrings/adoptium.asc] https://packages.adoptium.net/artifactory/deb
$(awk -F= '/^VERSION_CODENAME/{print$2}' /etc/os-release) main" | tee
/etc/apt/sources.list.d/adoptium.list
sudo apt update -y
sudo apt install temurin-17-jdk -y
/usr/bin/java --version
curl -fsSL https://pkg.jenkins.io/debian-stable/jenkins.io-2023.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-stable
binary/ | sudo tee /etc/apt/sources.list.d/jenkins.list > /dev/null
sudo apt-get update -y
```

sudo apt-get install jenkins -y

sudo systemctl start jenkins

sudo systemctl status jenkins

Make the script executable:

sudo chmod +x install_jenkins.sh

Run the script:

./install_jenkins.sh

The script will install Jenkins and start the Jenkins service.

Step2A: Install Docker on the Jenkins machine

Run the below commands to install the docker

sudo apt-get update

sudo apt-get install docker.io -y

sudo usermod -aG docker $USER #my case is ubuntu

newgrp docker

sudo chmod 777 /var/run/docker.sock

After the docker installation, create a Sonarqube container (add 9000 ports in the security group).

docker run -d --name sonar -p 9000:9000 sonarqube:lts-community

Now copy the IP address of the ec2 instance

<ec2-public-ip:9000>

Enter username and password, click on login and change password
username : admin
password : admin

Step2B: Install Trivy on Jenkins machine

Create a shell script to install trivy

sudo vi trivy.sh

sudo apt-get install wget apt-transport-https gnupg lsb-release -y

wget -qO - https://aquasecurity.github.io/trivy-repo/deb/public.key | gpg --dearmor | sudo tee /usr/share/keyrings/trivy.gpg > /dev/null
echo "deb [signed-by=/usr/share/keyrings/trivy.gpg] https://aquasecurity.github.io/trivy-repo/deb $(lsb_release -sc) main" | sudo tee -a /etc/apt/sources.list.d/trivy.list
sudo apt-get update
sudo apt-get install trivy -y

Provide executable permissions and run the shell script
sudo chmod +x trivy.sh

./trivy.sh

This will install Trivy on our Jenkins machine.

Step3A: Launch an Ubuntu instance for Splunk

Step 1: Launch Instances , select T2.medium with 24GB of storage

Create a splunk account in splunk.com

Step 2: Install Splunk

wget -O splunk-9.1.1-64e843ea36b1-linux-2.6-amd64.deb "https://download.splunk.com/products/splunk/releases/9.1.1/linux/splunk-9.1.1-64e843ea36b1-linux-2.6-amd64.deb"

running the below command, you ensure that SSH access is permitted through your firewall, which is crucial for remote server management and administration.

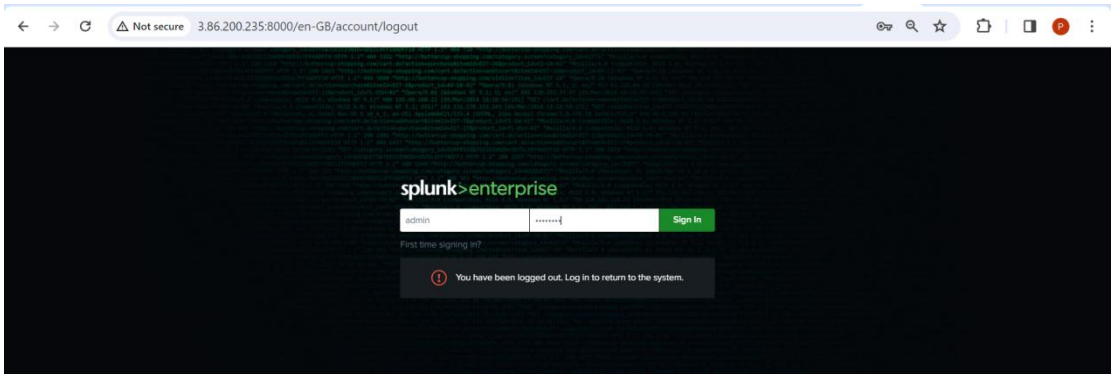
sudo ufw allow openSSH
sudo ufw allow 8000
sudo ufw status
sudo ufw enable

sudo /opt/splunk/bin/splunk start

The command `sudo /opt/splunk/bin/splunk start` is used to start the Splunk Enterprise application on your system. When you run this command with superuser privileges (using `sudo`), it initiates the Splunk service, allowing you to begin using the Splunk platform for data analysis, monitoring, and other data-related tasks.

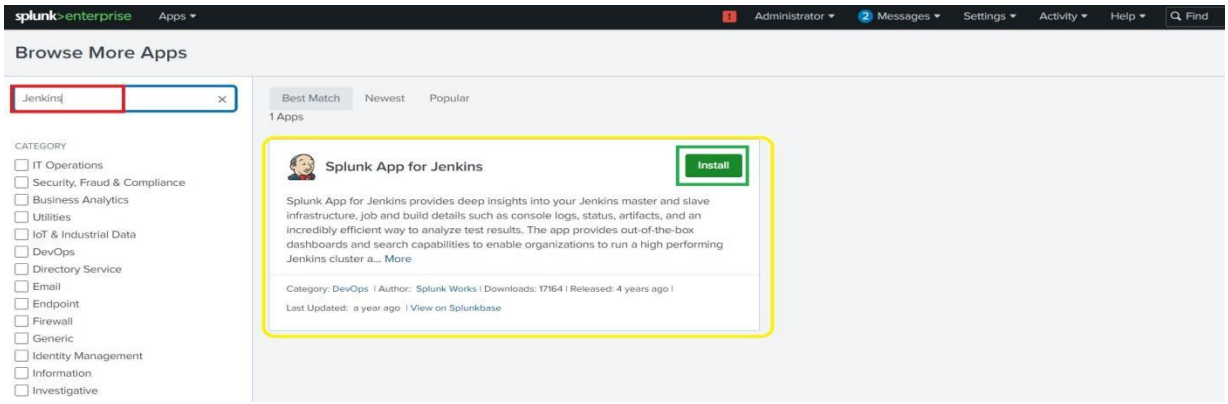
Access with

<splunk-public-ip:8000>

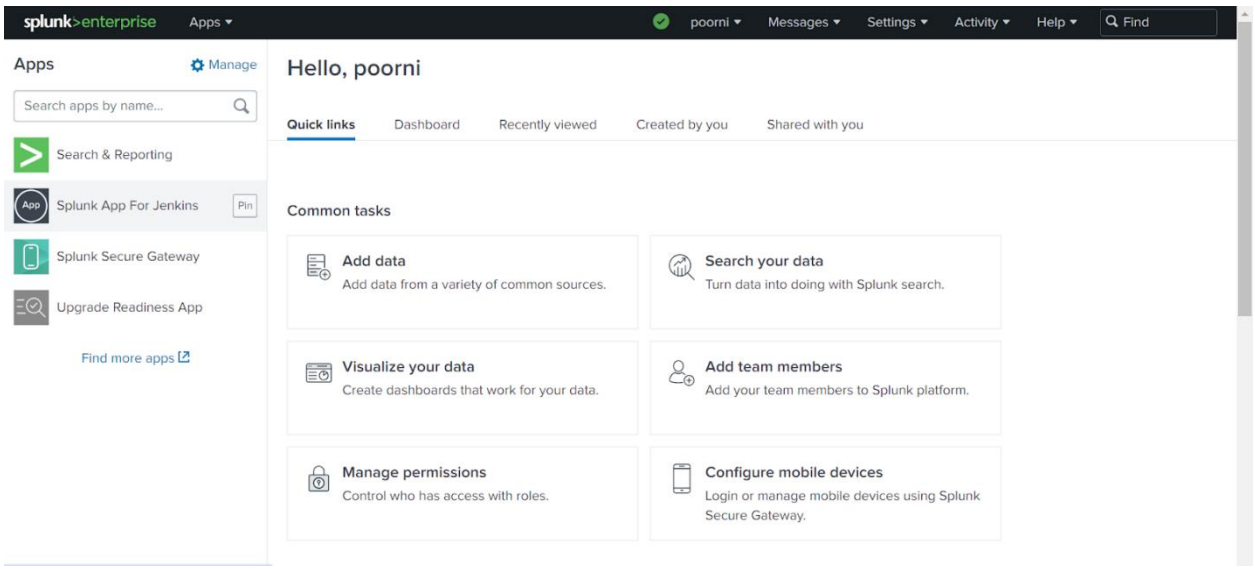


Step3B: Install the Splunk app for Jenkins

In Splunk Dashboard .Click on Apps --> Find more apps

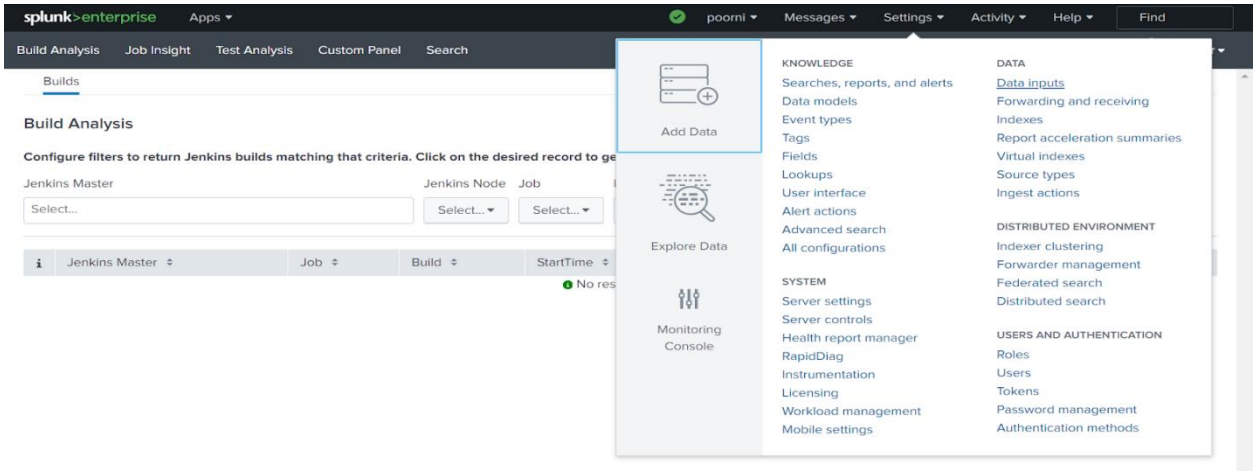


provide your Splunk credentials. That's why we created a Splunk account.

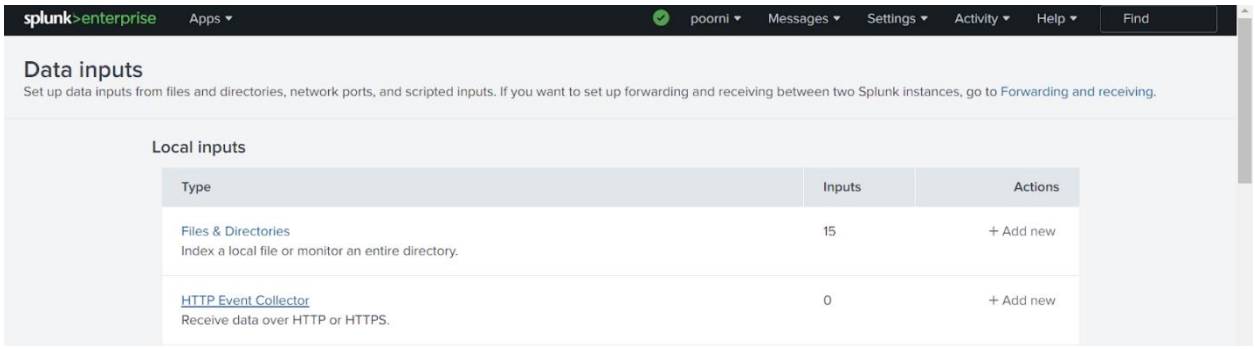


Splunk app for jenkins will be installed

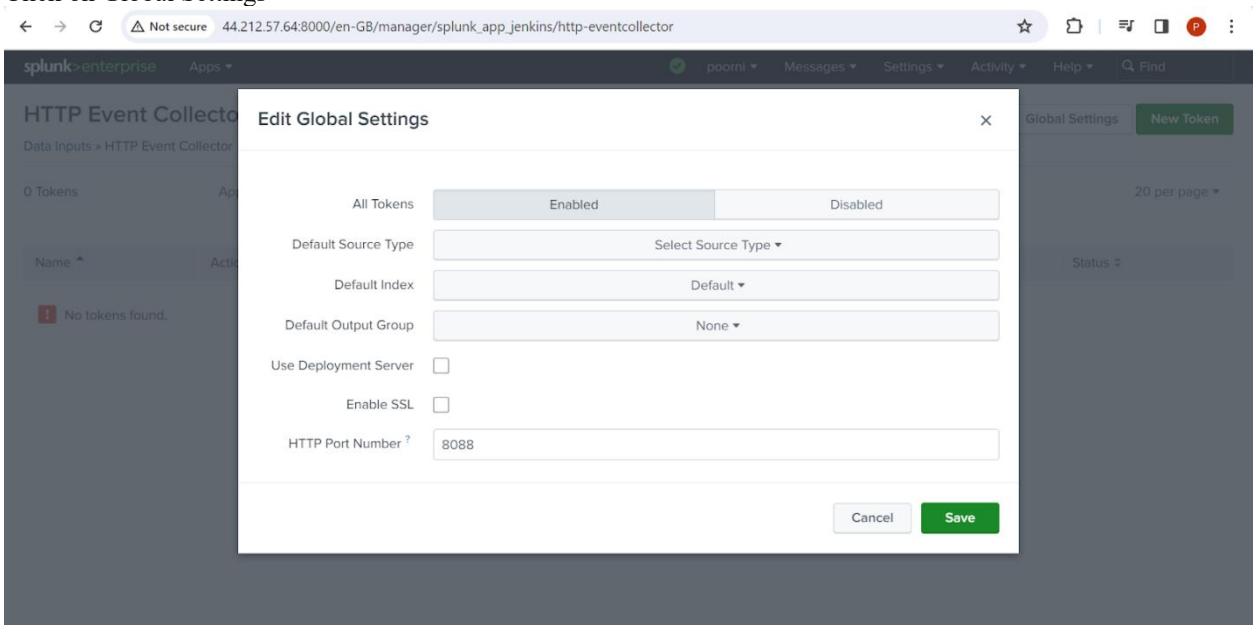
In the Splunk web interface, go to Settings > Data Inputs.



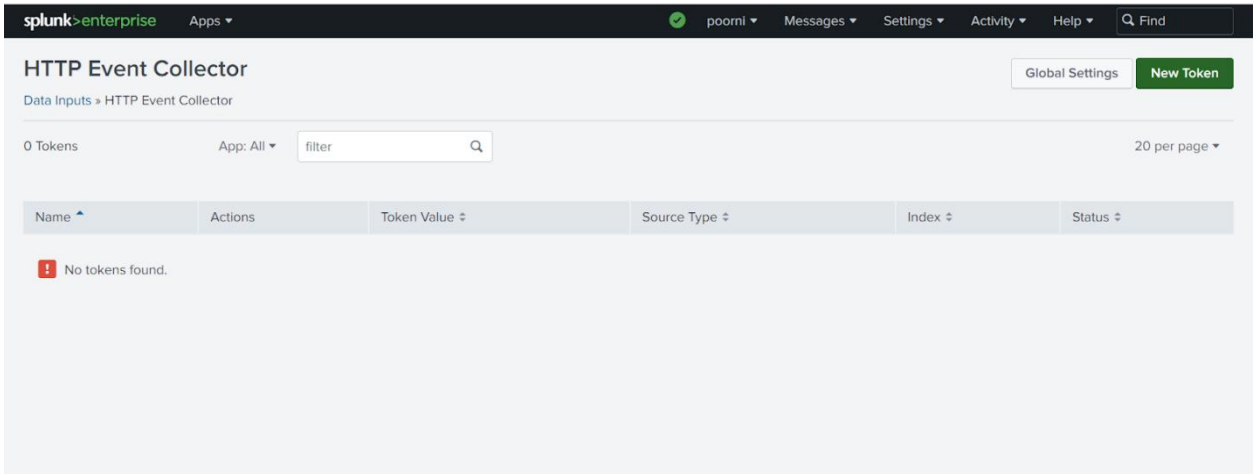
Click on HTTP Event Collector.

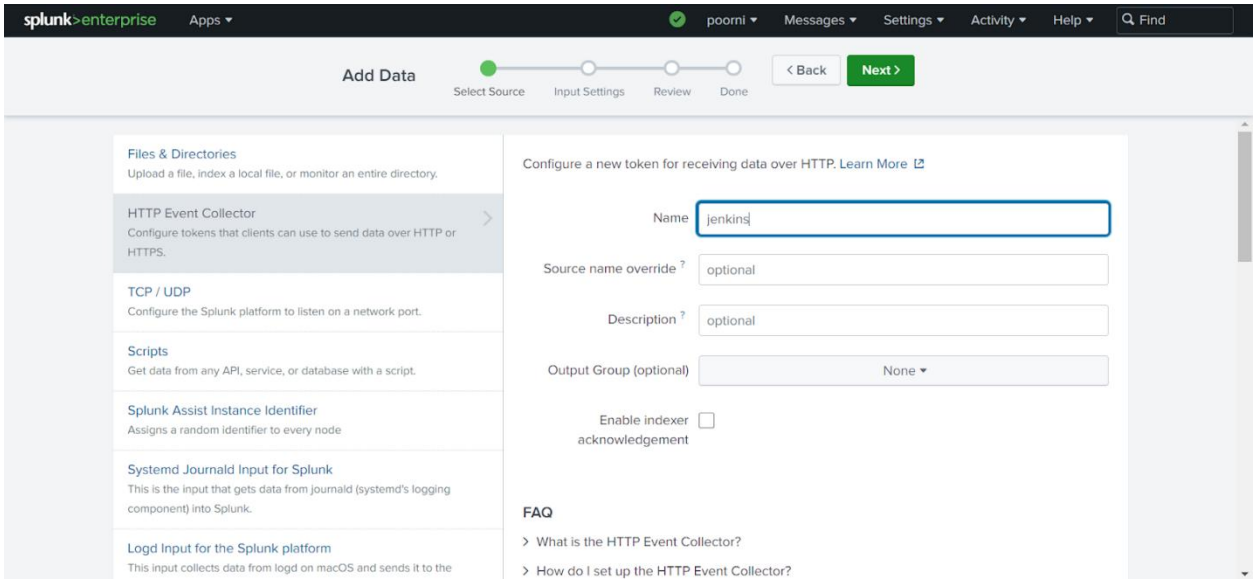


Click on Global Settings

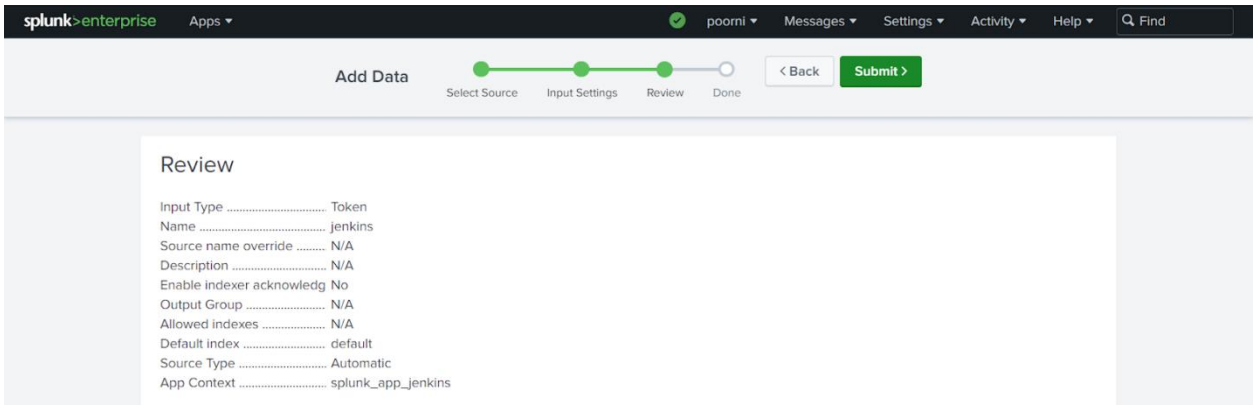


Generate a token

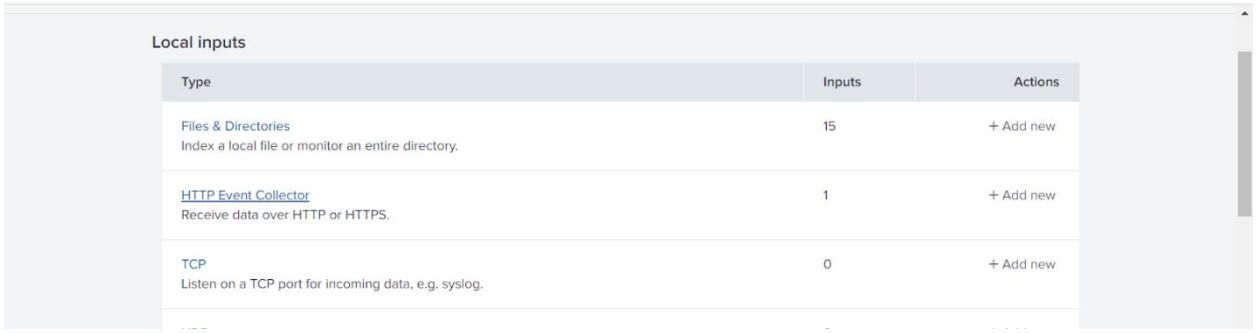




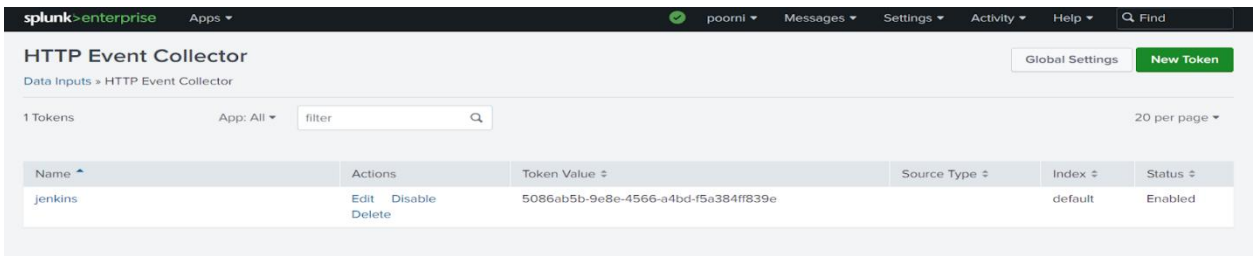
Click next and submit



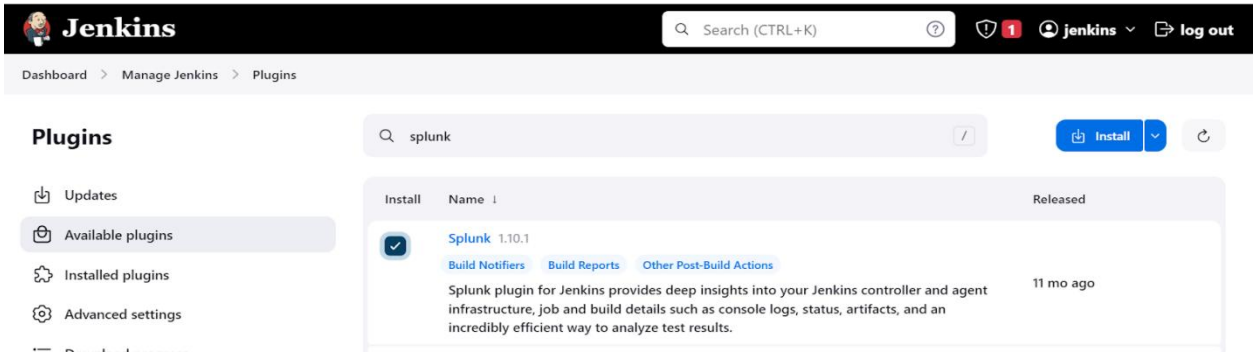
In the Splunk web interface, go to **Settings** > **Data Inputs**.Click on the HTTP event collector



Now copy your token and keep it saved



Add Splunk Plugin in Jenkins



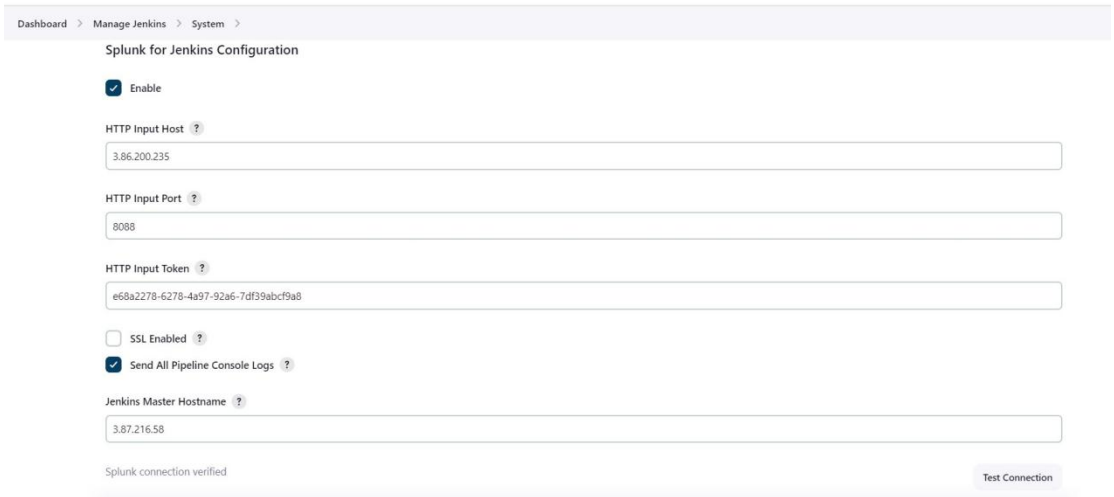
Again Click on Manage Jenkins --> System

Search for Splunk then Check to enable

HTTP input host as SPLUNK PUBLIC IP

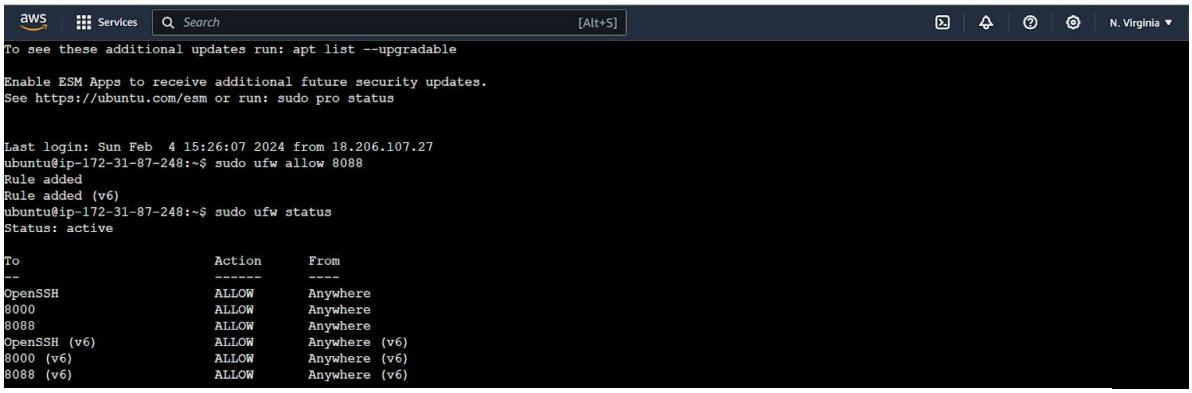
HTTP token that you generated in Splunk

Jenkins IP and apply.

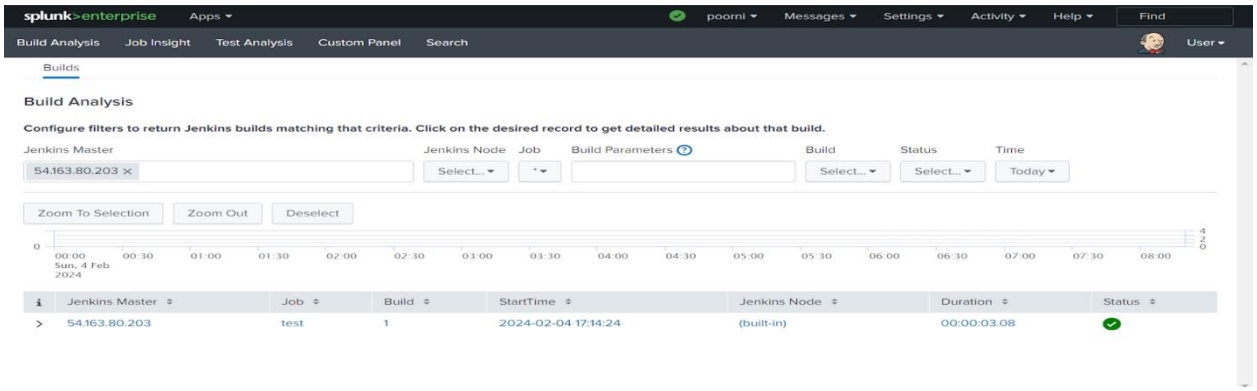


In Splunk machine run this command and test the connection

sudo ufw allow 8088

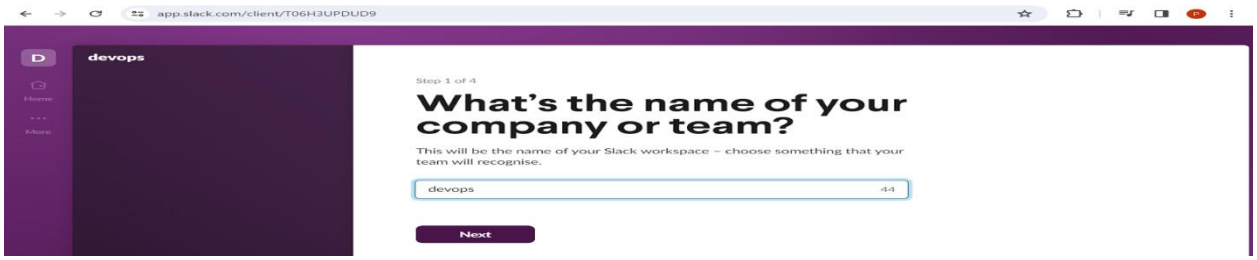


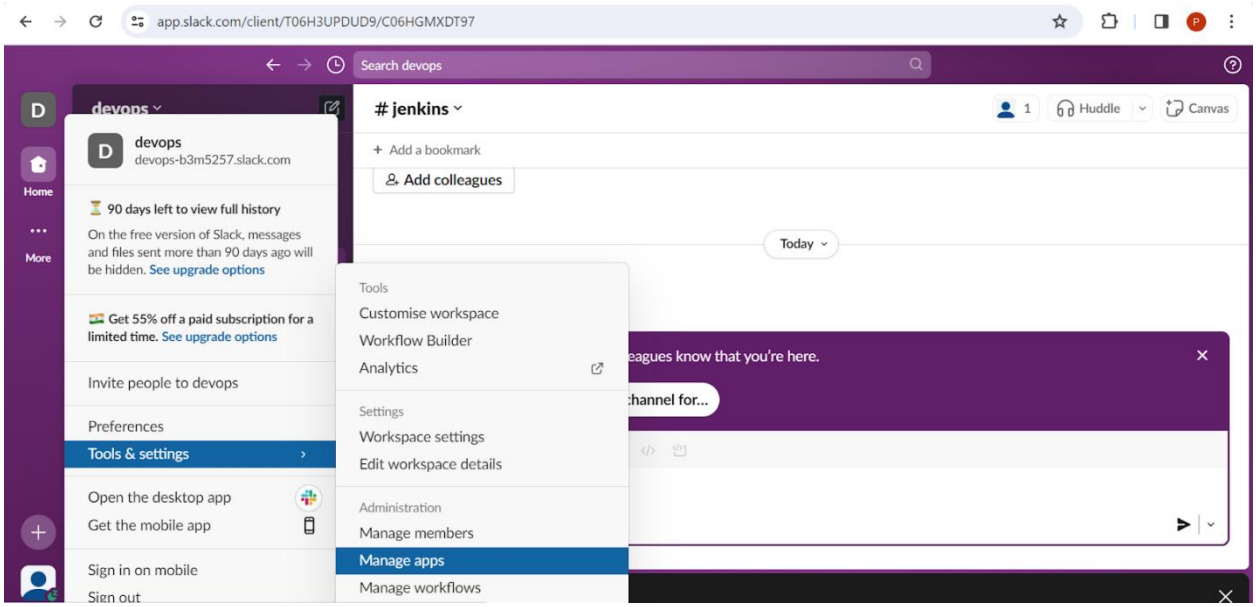
Restart Both Splunk and Jenkins .Let's Restart our Splunk machine ,Click on Settings --> Servecontrols



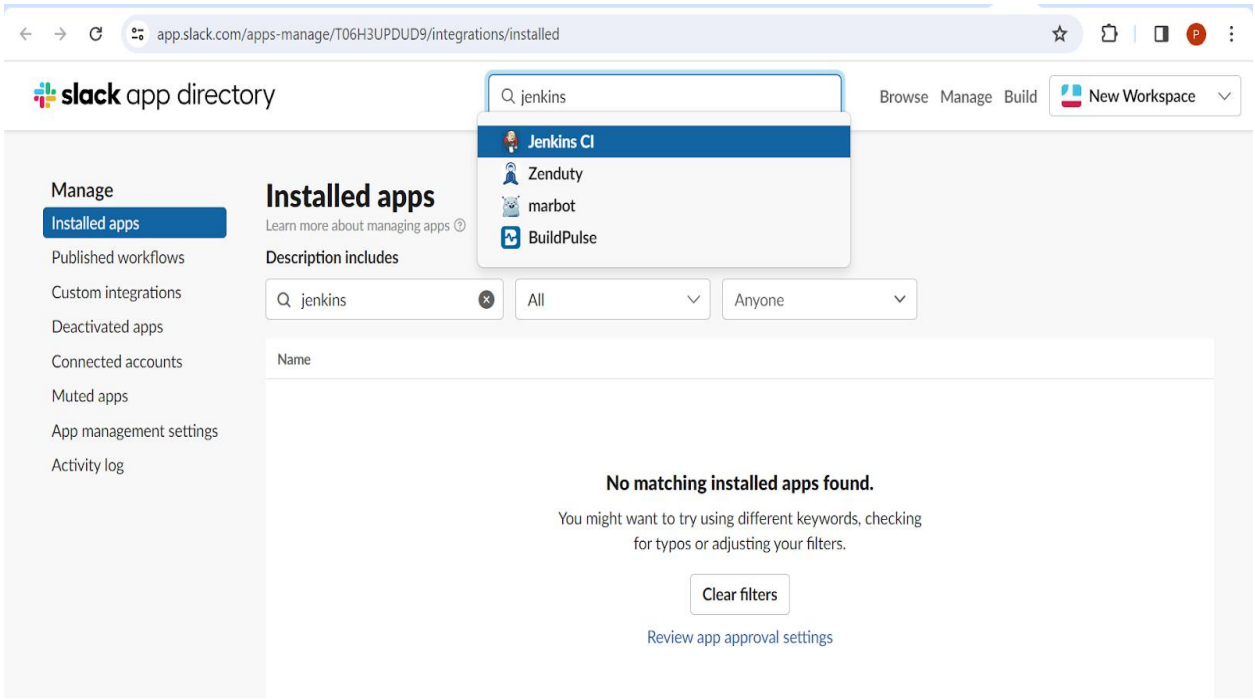
Step4A: Integrate Slack for Notifications

Create a Slack account and create a channel





Search for Jenkins CI and click on it



Click on Add to Slack

devops-b3m5257.slack.com/apps/A0F7VRFKN-jenkins-ci?tab=more_info

slack app directory | Search App Directory | Browse | Manage | Build | devops

< Browse apps

Jenkins CI

Description | Permissions | Security & compliance

Jenkins CI is a customisable continuous integration server with over 600 plugins, allowing you to configure it to meet your needs.

This integration will post build notifications to a channel in Slack.

Add to Slack

Learn more & Support
Privacy policy
Terms

devops-b3m5257.slack.com/apps/new/A0F7VRFKN-jenkins-ci

slack app directory | Search App Directory | Browse | Manage | Build | devops

Jenkins CI

An open-source continuous integration server.

Jenkins CI is a customisable continuous integration server with over 600 plugins, allowing you to configure it to meet your needs.

This integration will post build notifications to a channel in Slack.

Post to channel

Start by choosing a channel where Jenkins notifications will be posted.

jenkins

or create a new channel

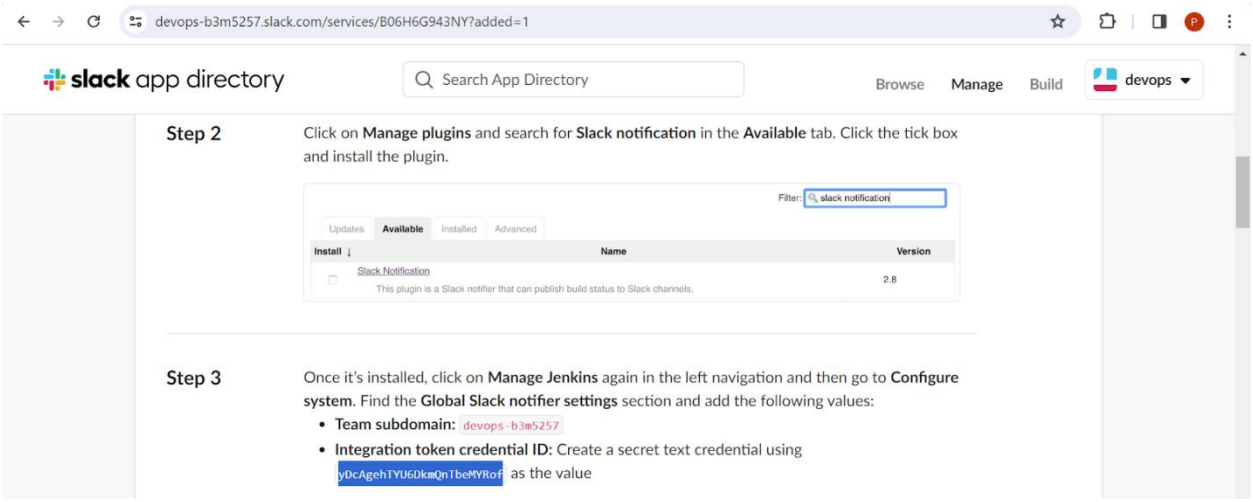
Add Jenkins CI integration

devops-b3m5257.slack.com/services/B06H6G943NY?added=1

slack app directory | Search App Directory | Browse | Manage | Build | devops

Step 1 In your Jenkins dashboard, click on **Manage Jenkins** from the left navigation.

Step 2 Click on **Manage plugins** and search for **Slack notification** in the **Available** tab. Click the tick box and install the plugin.

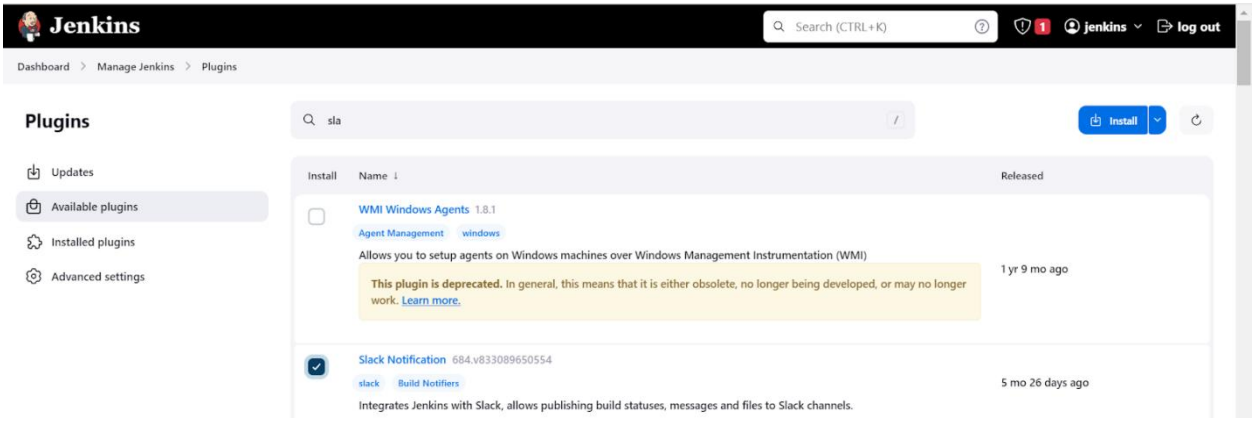


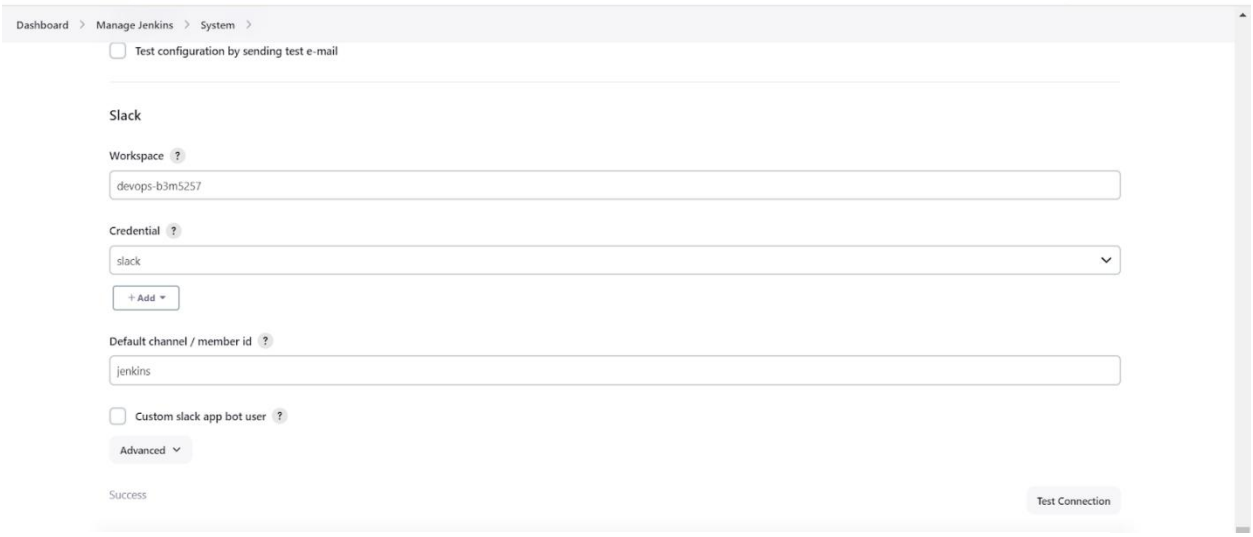
Click on Manage Jenkins --> Credentials --> Global

Select kind as Secret Text

At Secret Section Provide Your Slack integration token credential ID

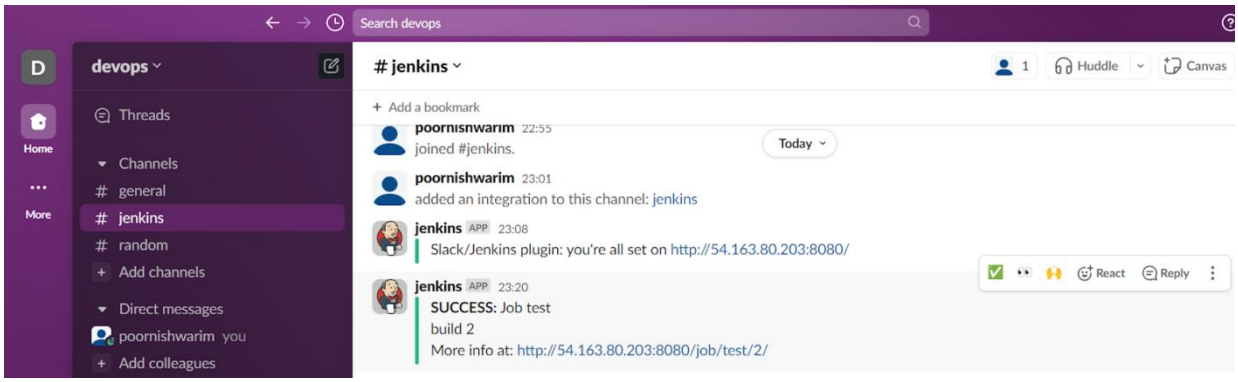
Click on manage Jenkins --> Plugins --> Available plugins ,Search for Slack Notification and install





Click on Apply and save and Add this to the pipeline

```
def COLOR_MAP = [
    'FAILURE' : 'danger',
    'SUCCESS' : 'good'
]


post {
    always {
        echo 'Slack Notifications'
        slackSend (
            channel: '#channel name',   #change your channel name
            color: COLOR_MAP[currentBuild.currentResult],
            message: "*${currentBuild.currentResult}:* Job ${env.JOB_NAME} \n build ${env.BUILD_NUMBER} \n More info at: ${env.BUILD_URL}"
        )
    }
}
```



You will get a Notification in Slack

Step5A: Start Job

Go to Jenkins dashboard and click on New Item. Provide a name for the Job & click on Pipeline.

Step5B: Create a Jenkins shared library in GitHub

Create a directory named Jenkins_shared_library

Create a Vars directory inside it

Run the below commands to push to GitHub

```
echo "# Jenkins_shared_library" >> README.md
git init
git add README.md
git commit -m "first commit"
git branch -M main
# make sure to change your repo Url here
git remote add origin https://github.com/git_user/Jenkins_shared_library.git
git push -u origin main
```

Create a **cleanWorkspace.groovy** file and add the below code

```
#cleanWorkspace.groovy //cleans workspace
def call() {
   cleanWs()
}
```

Create **checkoutGit.groovy** file and add the below code

```
def call(String gitUrl, String gitBranch) {
   checkout([
      $class: 'GitSCM',
      branches: [[name: gitBranch]],
      userRemoteConfigs: [[url: gitUrl]]
   ])
}
```

Now push them to GitHub using the below commands from vs code

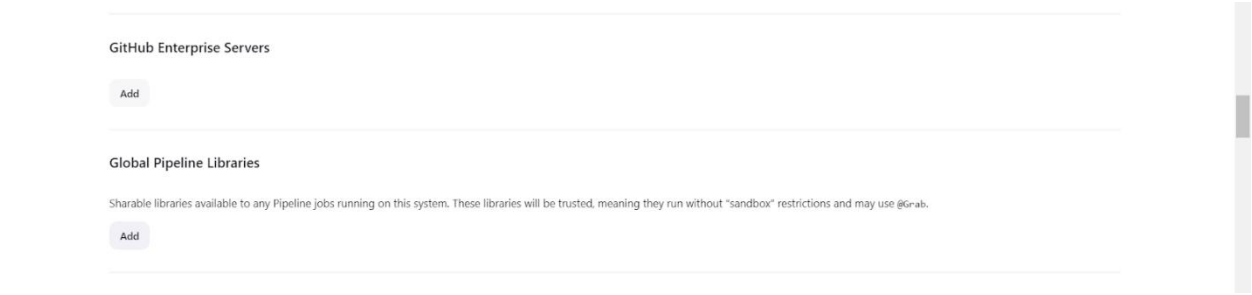
```
git add .
git commit -m "message"
git push origin main
```

Step5C: Add Jenkins shared library to Jenkins system

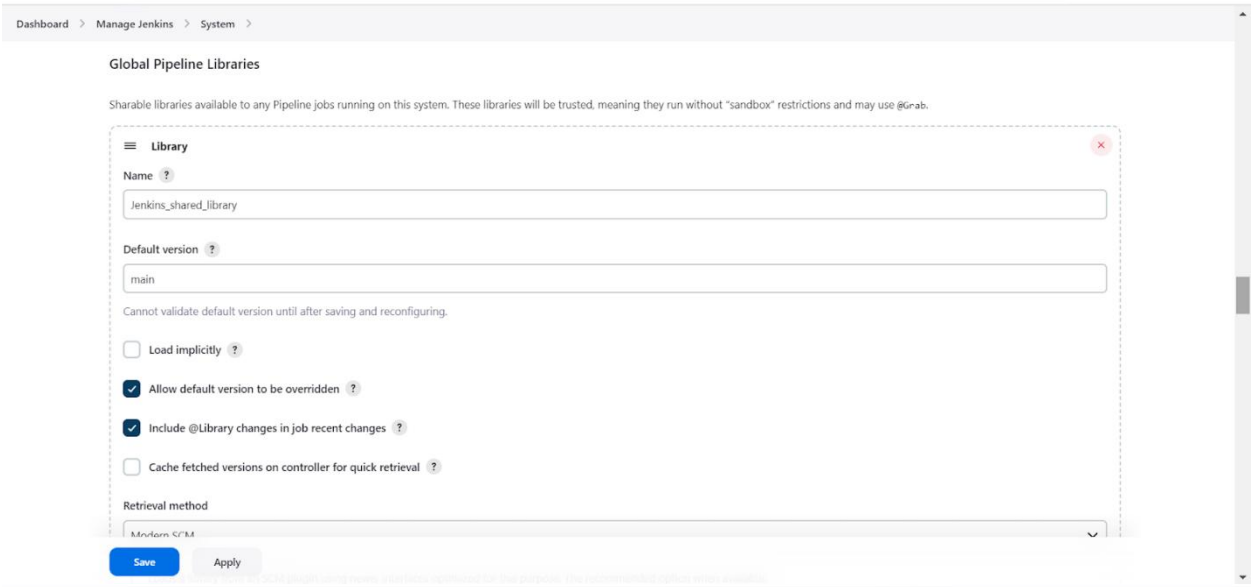
Go to Jenkins Dashboard

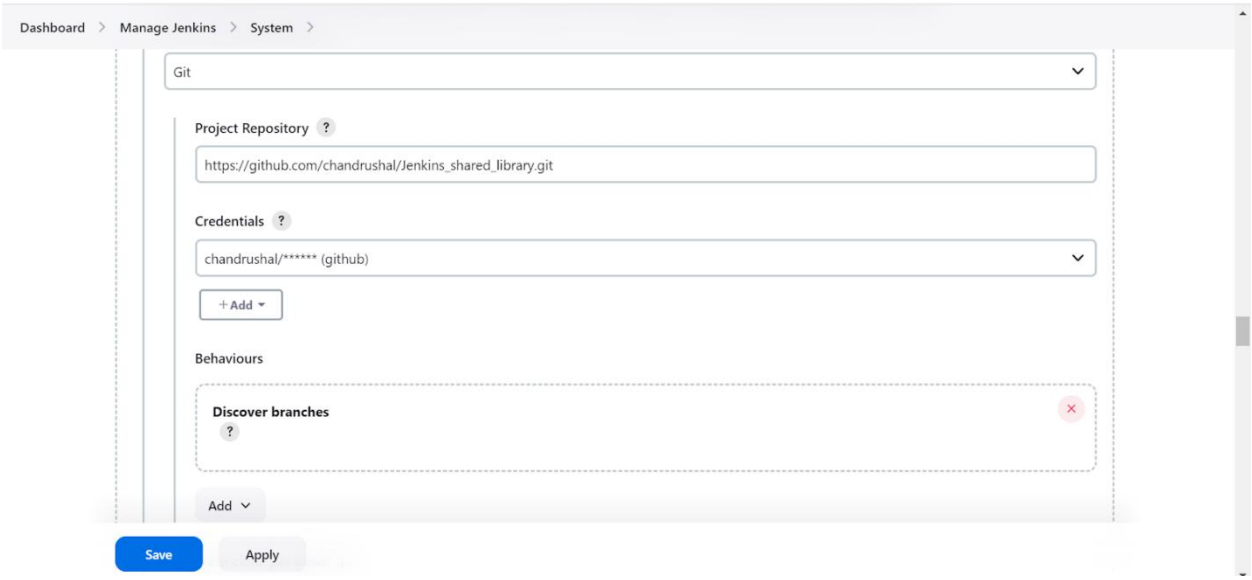
Click on Manage Jenkins --> system

Search for Global Pipeline Libraries and click on Add



Now Provide a name that we have to call in our pipeline





Click apply and save

Step5D: Run Pipeline

Go to Jenkins Dashboard again & select the job and add the below pipeline

```
@Library('Jenkins_shared_library') _  #name used in jenkins system for library
def COLOR_MAP = [
    'FAILURE' : 'danger',
    'SUCCESS' : 'good'
```

```
]

pipeline{
   agent any
   parameters {
      choice(name: 'action', choices: 'create\ndelete', description: 'Select create or destroy.')
   }
   stages{
      stage('clean workspace'){
         steps{
            cleanWorkspace()
         }
      }
      stage('checkout from Git'){
         steps{
            checkoutGit('https://github.com/chandrushal/Youtube-clone-app.git', 'main')
         }
      }
    }
    post {
       always {
          echo 'Slack Notifications'
          slackSend (
             channel: 'jenkins',   #change your channel name
             color: COLOR_MAP[currentBuild.currentResult],
             message:     "*${currentBuild.currentResult}:*     Job     ${env.JOB_NAME}     \n     build
${env.BUILD_NUMBER} \n More info at: ${env.BUILD_URL}"
           )
        }
     }
  }
```

Build with parameters and build

Stage view

Step6: Install Plugins like JDK, Sonarqube Scanner, NodeJs

Step6A: Install Plugin

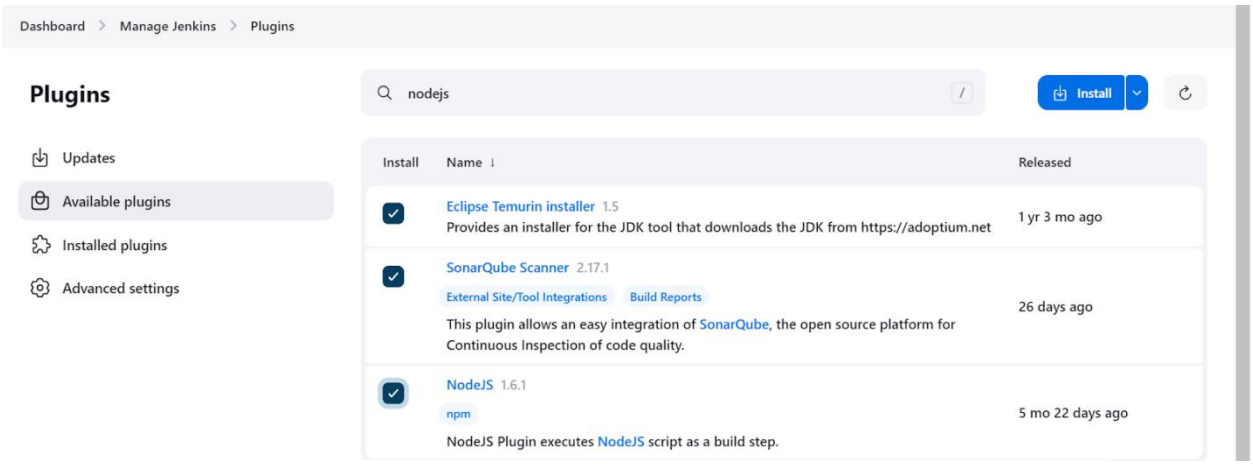
Goto Manage Jenkins →Plugins → Available Plugins →

Install below plugins

1 → Eclipse Temurin Installer (Install without restart)

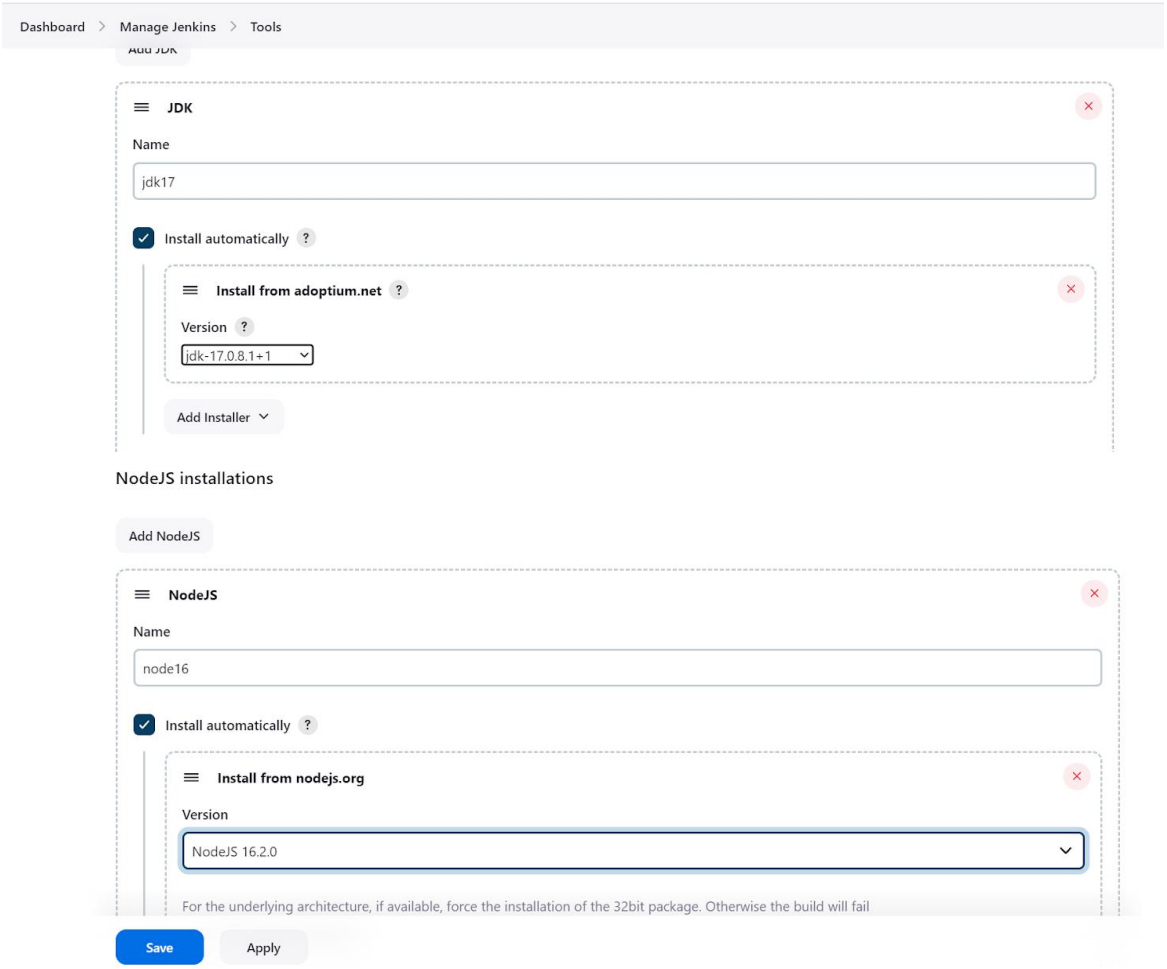
2 → SonarQube Scanner (Install without restart)

3 → NodeJs Plugin (Install Without restart)



Step6B: Configure Java and Nodejs in Global Tool Configuration

Goto Manage Jenkins → Tools → Install JDK(17) and NodeJs(16)→ Click on Apply and Save

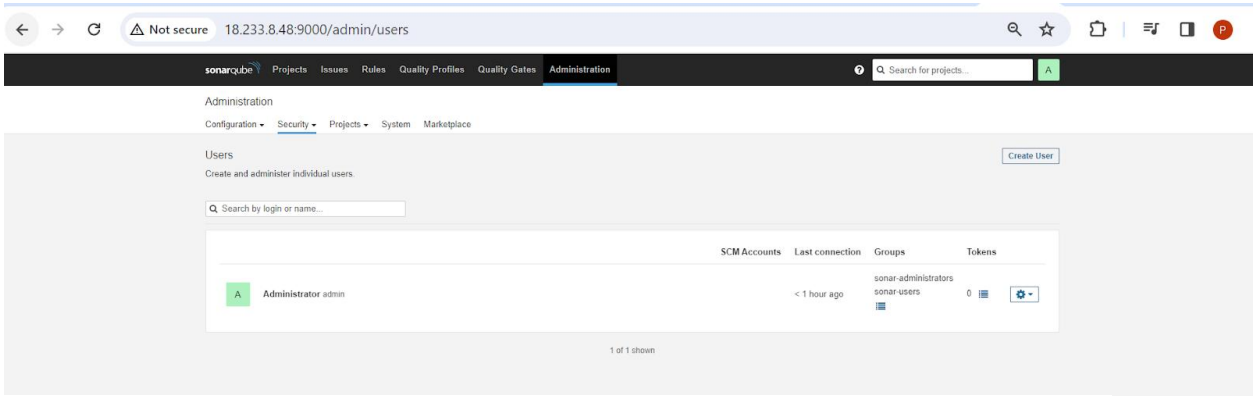


Apply and save it.

Step6C: Configure Sonar Server in Manage Jenkins

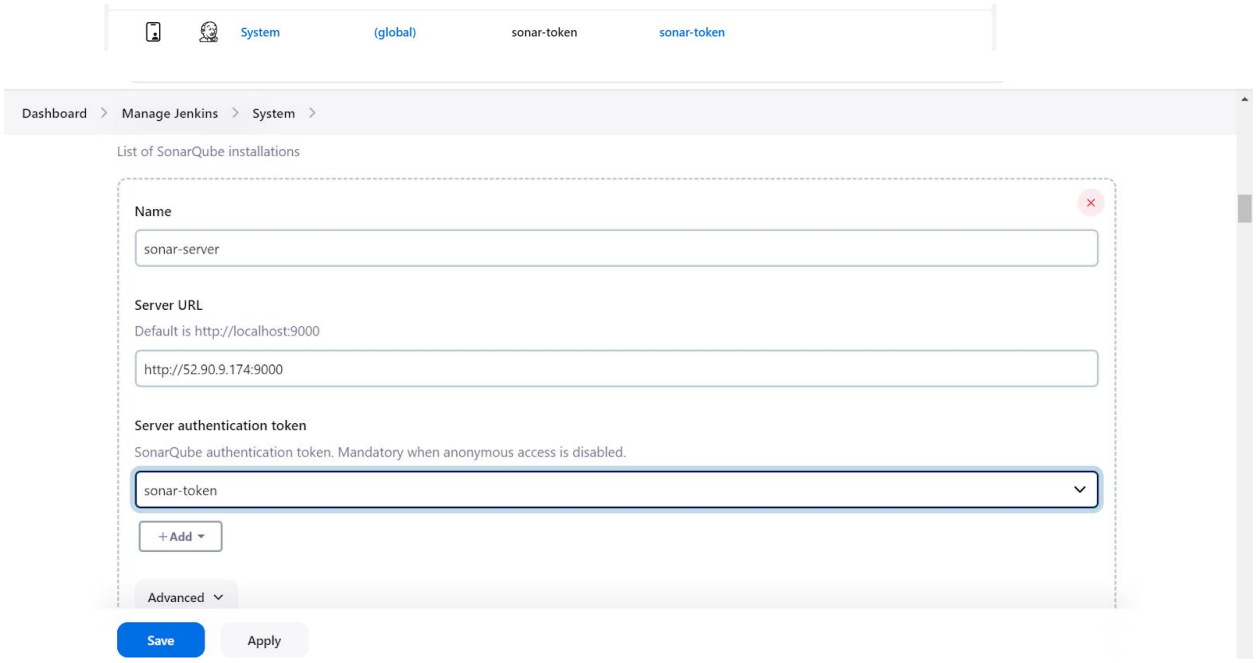
Goto your Sonarqube Server. Click on Administration → Security → Users → Click on Tokens and

Update Token → Give it a name → and click on Generate Token



copy Token

Goto Jenkins Dashboard → Manage Jenkins → Credentials → Add Secret Text.

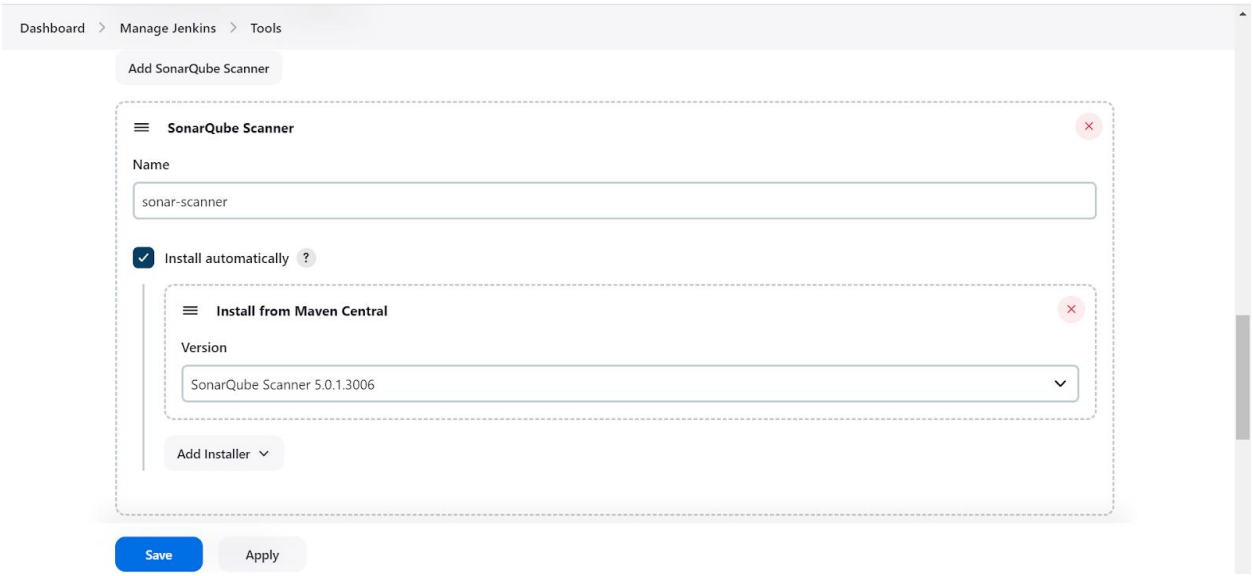


Click on Apply and Save

The Configure System option is used in Jenkins to configure different server

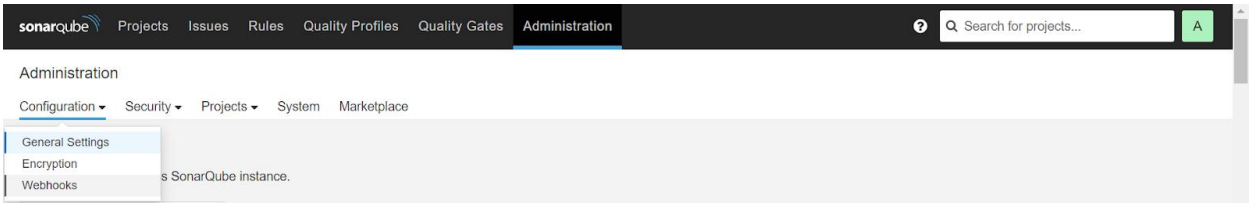
Global Tool Configuration is used to configure different tools that we install using Plugins

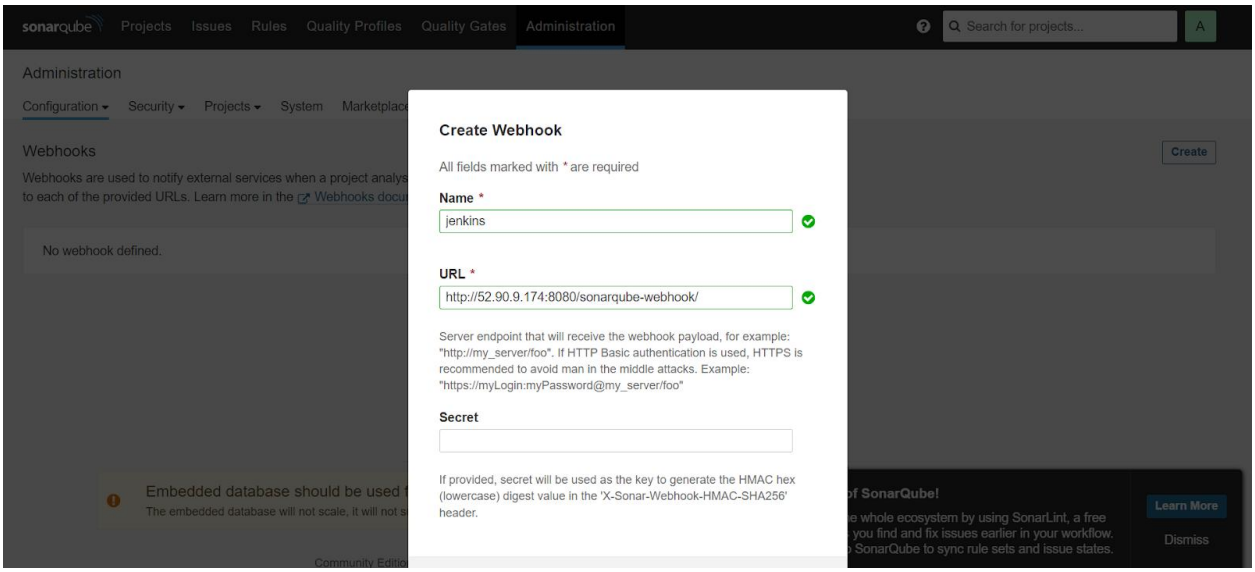
We will install a sonar scanner in the tools.



In the Sonarqube Dashboard add a quality gate also

Administration--> Configuration-->Webhooks





Click on Create

Step6D: Add New stages to the pipeline

Go to vs code and create a file **sonarqubeAnalysis.groovy** & add the below code and push to Jenkins

shared library GitHub Repo.

```
def call() {
   withSonarQubeEnv('sonar-server') {
      sh '''$SCANNER_HOME/bin/sonar-scanner -Dsonar.projectName=Youtube -
Dsonar.projectKey=Youtube '''
   }
}
```

Create another file for **qualityGate.groovy**

```
def call(credentialsId) {
   waitForQualityGate abortPipeline: false, credentialsId: credentialsId
}
```

Create another file for **npmInstall.groovy**

```
def call() {
   sh 'npm install'
}
```

Push them to the GitHub Jenkins shared library

```
git add .
git commit -m "message"
git push origin main
```

Add these stages to the pipeline now

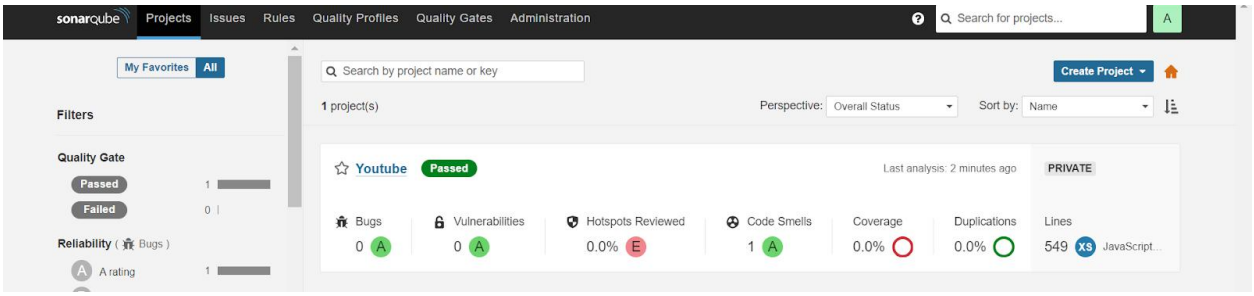
```
#under parameters
tools{
      jdk 'jdk17'
      nodejs 'node16'
   }
   environment {
      SCANNER_HOME=tool 'sonar-scanner'
   }

# add in stages
stage('sonarqube Analysis'){
      when { expression { params.action == 'create'}}
         steps{
            sonarqubeAnalysis()
         }
      }
      stage('sonarqube QualitGate'){
      when { expression { params.action == 'create'}}
         steps{
            script{
               def credentialsId = 'Sonar-token'
               qualityGate(credentialsId)
            }
         }
      }
      stage('Npm'){
      when { expression { params.action == 'create'}}
         steps{
            npmInstall()
```

```
    }
}
```

Build now.

Stage view

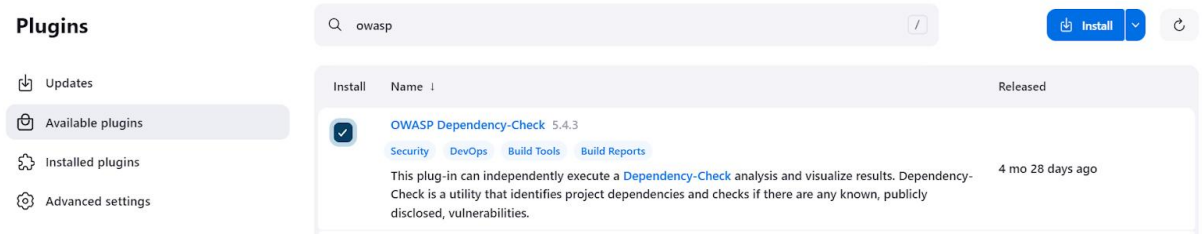
To see the report, you can go to Sonarqube Server and go to Projects.



Step7: Install OWASP Dependency Check Plugins

GotoDashboard → Manage Jenkins → Plugins → OWASP Dependency-Check. Click on it and install

it without restart.



Goto Dashboard → Manage Jenkins → Tools →

Click on Apply and Save here.

Create a file for **trivyFs.groovy**

```
def call() {
    sh 'trivy fs . > trivyfs.txt'
}
```

Push to GitHub

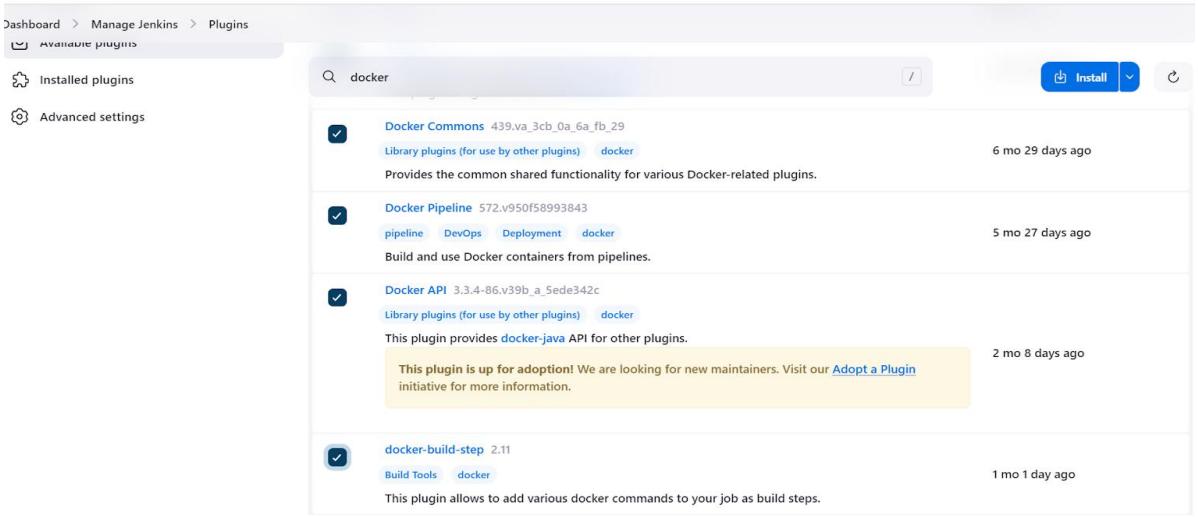
```
git add .
git commit -m "message"
git push origin main
```

Add the below stages to the Jenkins pipeline

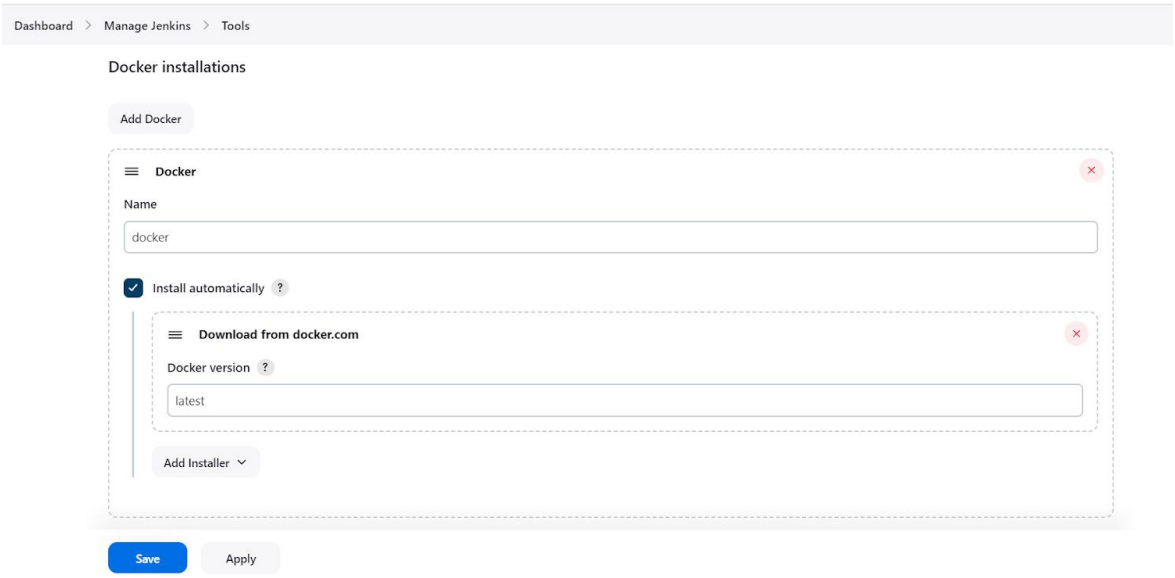
```
stage('Trivy file scan'){
        when { expression { params.action == 'create'}}
            steps{
                trivyFs()
            }
        }
        stage('OWASP FS SCAN') {
        when { expression { params.action == 'create'}}
            steps {
                dependencyCheck additionalArguments: '--scan ./ --disableYarnAudit --disableNodeAudit',
odcInstallation: 'DP-Check'
                dependencyCheckPublisher pattern: '**/dependency-check-report.xml'
            }
        }
```

Step8A: Docker Image Build and Push

We need to install the Docker tool in our system, Goto Dashboard → Manage Plugins → Available plugins → Search for Docker and install these plugins



Now, goto Dashboard → Manage Jenkins → Tools

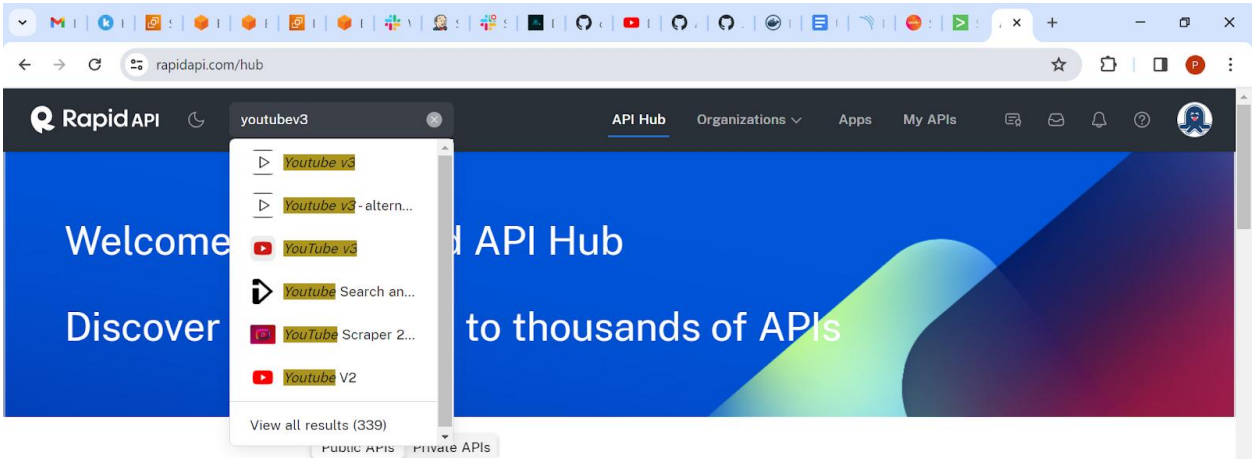


Add DockerHub Username and Password under Global Credentials

Step8B: Create an API key from Rapid API

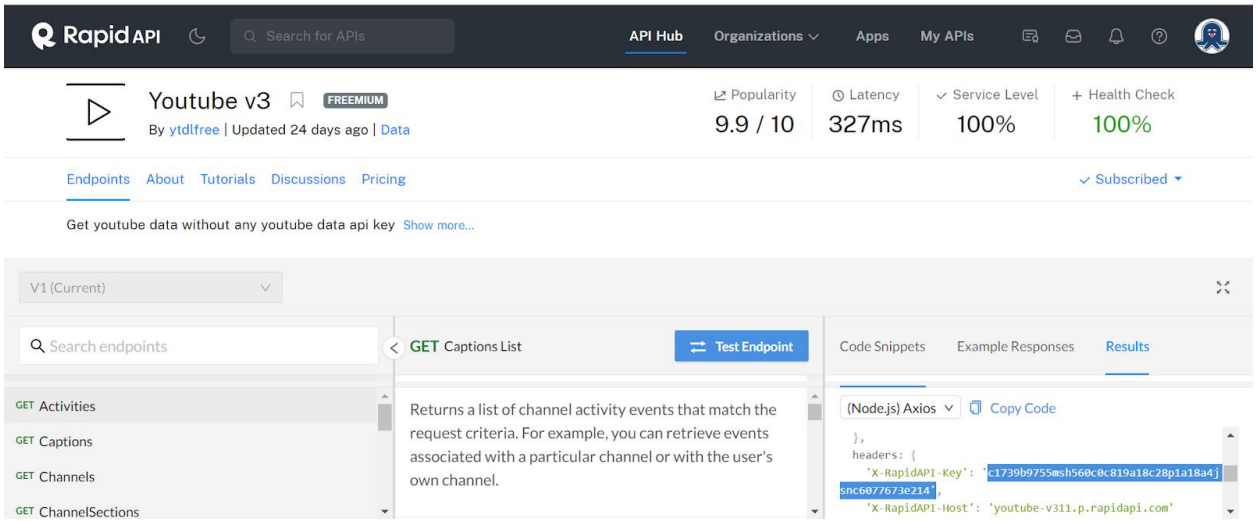
Open a new tab in the browser and search for rapidapi.com

It will automatically provide your mail and select a mail to create an account



Copy API and use it in the groovy file

docker build --build-arg REACT_APP_RAPID_API_KEY=<API-KEY> -t ${imageName} .



Create a shared library file for **dockerBuild.groovy**

```
def call(String dockerHubUsername, String imageName) {
   // Build the Docker image
   sh "docker build --build-arg REACT_APP_RAPID_API_KEY=f0ead79813mshb0aa -t ${imageName} ."
    // Tag the Docker image
   sh "docker tag ${imageName} ${dockerHubUsername}/${imageName}:latest"
   // Push the Docker image
   withDockerRegistry([url: 'https://index.docker.io/v1/', credentialsId: 'docker']) {
      sh "docker push ${dockerHubUsername}/${imageName}:latest"
   }
}
```

Create another file for **trivyImage.groovy**

```
def call() {
   sh 'trivy image sevenajay/youtube:latest > trivyimage.txt'
}
```

Push the above files to the GitHub shared library.

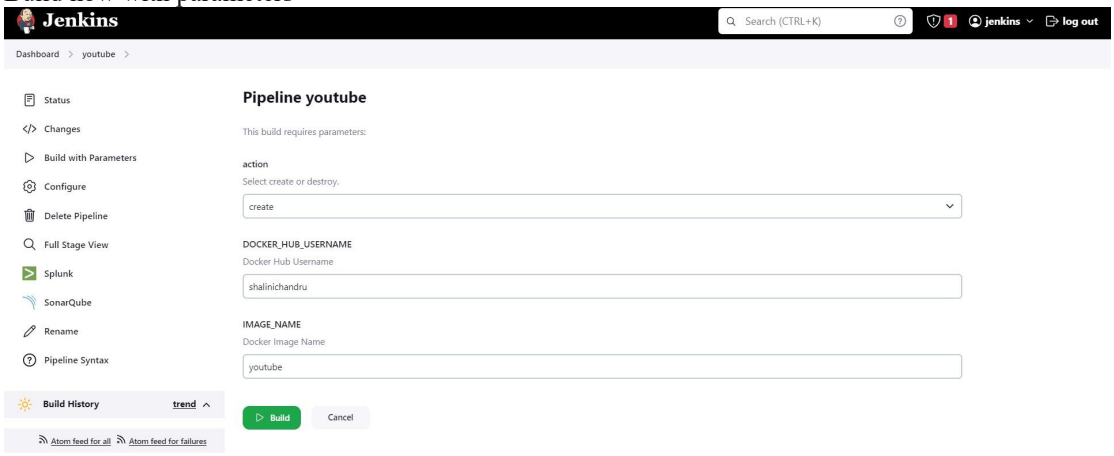
```
git add .
git commit -m "message"
git push origin main
```

Add this stage to your pipeline with parameters

```
#add inside parameter
 string(name: 'DOCKER_HUB_USERNAME', defaultValue: 'sevenajay', description: 'Docker Hub Username')
 string(name: 'IMAGE_NAME', defaultValue: 'youtube', description: 'Docker Image Name')
#stage
stage('Docker Build'){
        when { expression { params.action == 'create'}}
            steps{
                script{
```

```
            def dockerHubUsername = params.DOCKER_HUB_USERNAME
            def imageName = params.IMAGE_NAME

            dockerBuild(dockerHubUsername, imageName)
        }
    }
}
stage('Trivy iamge'){
when { expression { params.action == 'create'}}
    steps{
        trivyImage()
    }
}
```

Build now with parameters



When you log in to Dockerhub, you will see a new image is created

Step8C: Run the Docker container

Create a new file **runContainer.groovy**

```
def call(){
    sh "docker run -d --name youtube1 -p 3000:3000 sevenajay/youtube:latest"
}
```

Create Another file to remove container **removeContainer.groovy**

```
def call(){
    sh 'docker stop youtube1'
    sh 'docker rm youtube1'
}
```

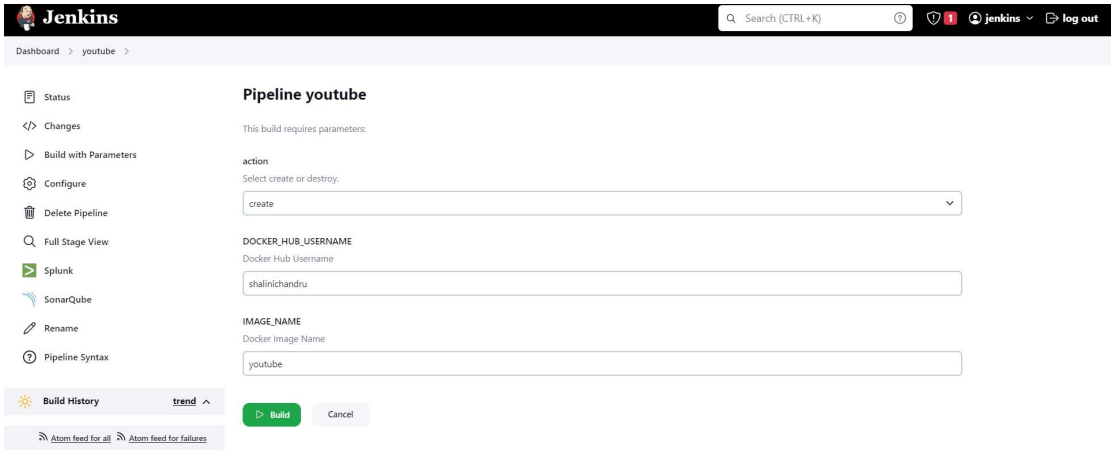
Push them to the Shared library GitHub repo

```
git add .
git commit -m "message"
git push origin main
```

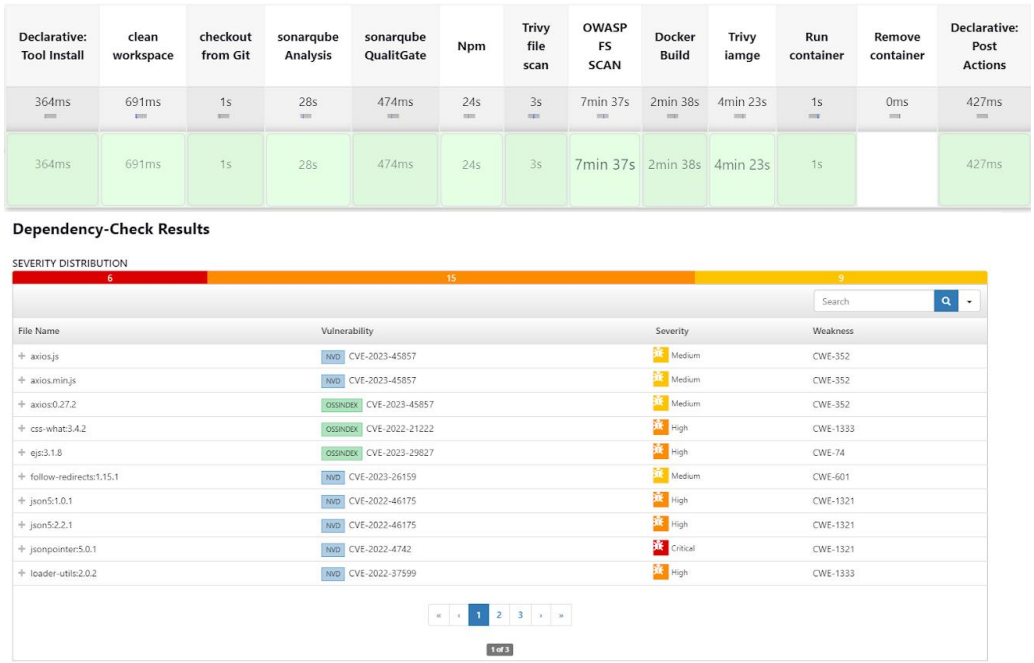
Add the below stages to the Pipeline

```
stage('Run container'){
        when { expression { params.action == 'create'}}
           steps{
              runContainer()
           }
        }
        stage('Remove container'){
        when { expression { params.action == 'delete'}}
           steps{
              removeContainer()
           }
        }
```

Build with parameters ”create”



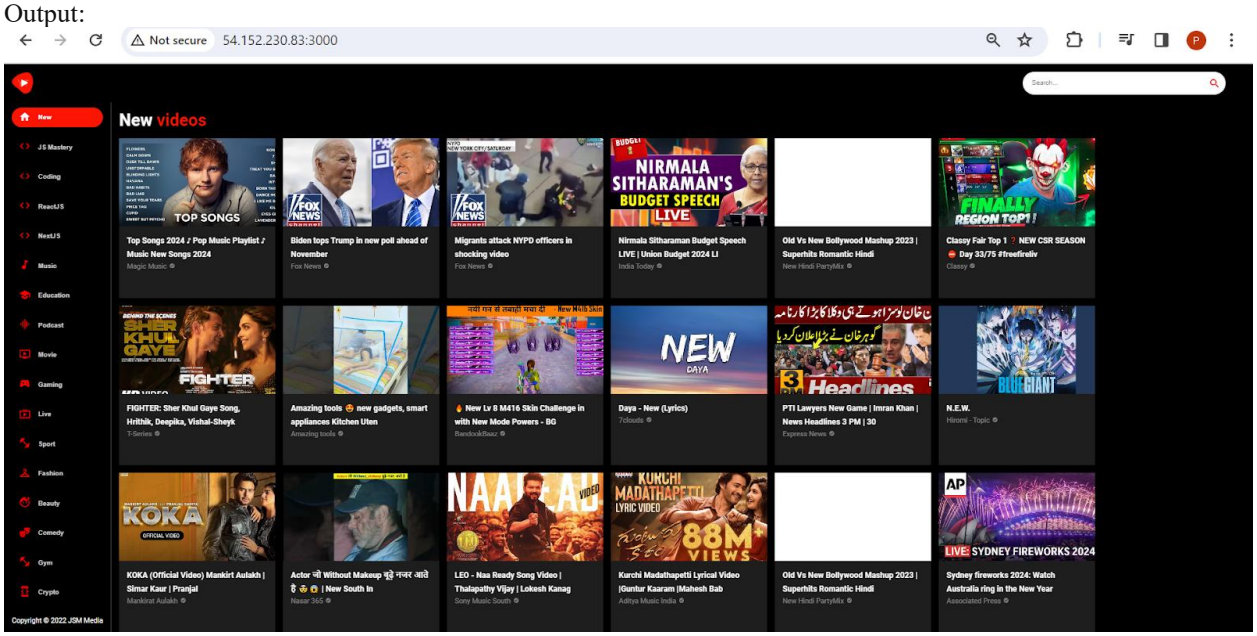
Stage view

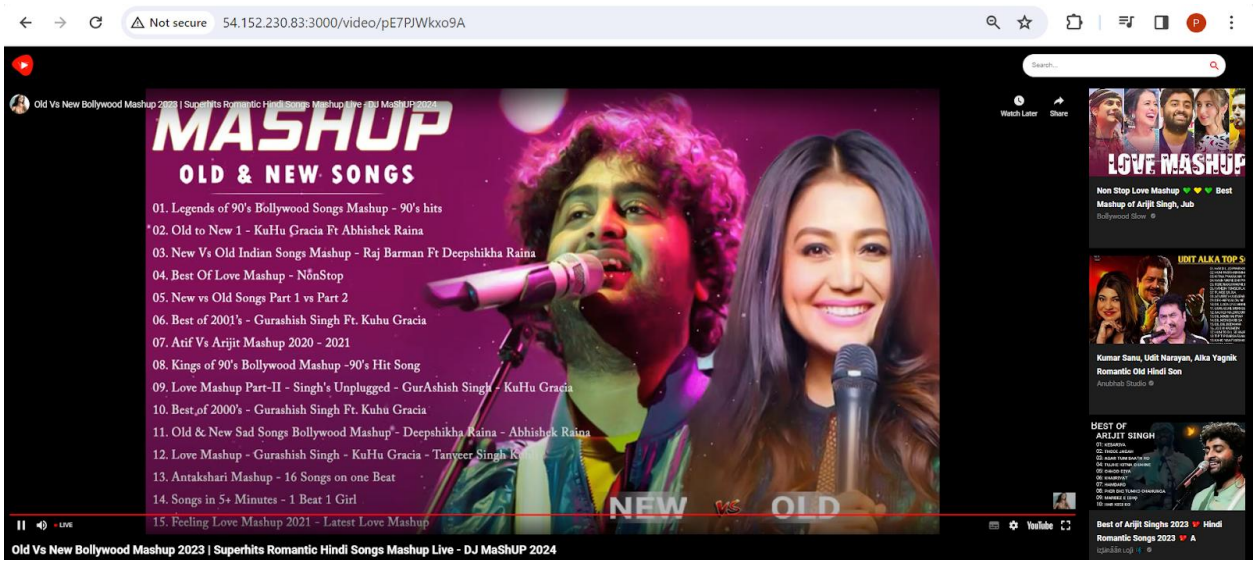


It will start the container

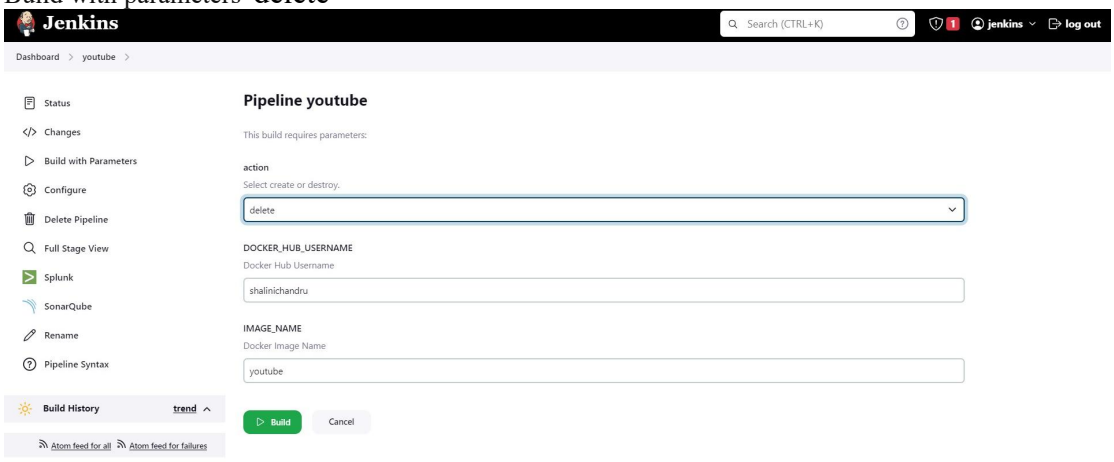
<public-ip-jenkins:3000>

Output:

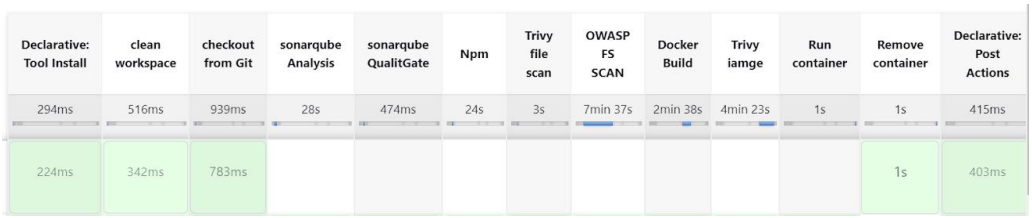




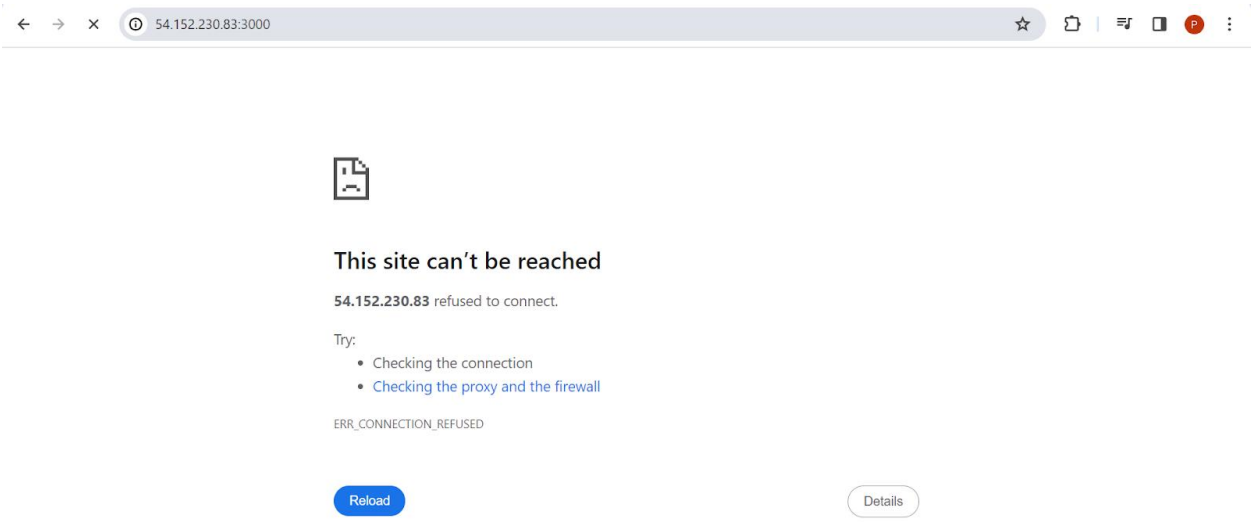
Build with parameters 'delete'



It will stop and remove the Container



Skipped some of the stages due to delete parameter



Step9A: Kubernetes Setup

Take-Two Ubuntu 20.04(t2.medium) instances one for k8s master and the other one for worker.

Install Kubectl on Jenkins machine also.

Step9B: Kubectl is to be installed on Jenkins

Connect your Jenkins machine

Create a shell script file kube.sh

sudo vi kube.sh

Paste the below commands

sudo apt update

sudo apt install curl -y

curl -LO https://dl.k8s.io/release/$(curl -L -s https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl

sudo install -o root -g root -m 0755 kubectl /usr/local/bin/kubectl

kubectl version --client

Step9C: K8S Master-Slave setup

Part 1 ----------Master Node----------

sudo hostnamectl set-hostname K8s-Master

----------Worker Node----------

sudo hostnamectl set-hostname K8s-Worker

Part 2 ----------Both Master & Node ----------

sudo apt-get update

sudo apt-get install -y docker.io
sudo usermod –aG docker Ubuntu
newgrp docker
sudo chmod 777 /var/run/docker.sock

sudo curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo apt-key add -

sudo tee /etc/apt/sources.list.d/kubernetes.list <<EOF
deb https://apt.kubernetes.io/ kubernetes-xenial main
EOF

sudo apt-get update

sudo apt-get install -y kubelet kubeadm kubectl

sudo snap install kube-apiserver

Part 3 ---------- Master ----------

sudo kubeadm init --pod-network-cidr=10.244.0.0/16

in case your in root exit from it and run below commands

mkdir -p $HOME/.kube

sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config

sudo chown $(id -u):$(id -g) $HOME/.kube/config

kubectl apply -f https://raw.githubusercontent.com/coreos/flannel/master/Documentation/kube-flannel.yml

```
[addons] Applied essential addon: kube-proxy

Your Kubernetes control-plane has initialized successfully!

To start using your cluster, you need to run the following as a regular user:

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

Alternatively, if you are the root user, you can run:

  export KUBECONFIG=/etc/kubernetes/admin.conf

You should now deploy a pod network to the cluster.
Run "kubectl apply -f [podnetwork].yaml" with one of the options listed at:
  https://kubernetes.io/docs/concepts/cluster-administration/addons/

Then you can join any number of worker nodes by running the following on each as root:

kubeadm join 172.31.81.6:6443 --token ncxrog.od0ao7t7d4mu1j82 \
        --discovery-token-ca-cert-hash sha256:15bab2a7e6eaeb701a61c15aa98cf31a78258687dc0c50a65316662dec176535
```

----------Worker Node----------

sudo kubeadm join <master-node-ip>:<master-node-port> --token <token> --discovery-token-ca-cert-hash <hash>

```
ubuntu@K8s-slave:~$ sudo kubeadm join 172.31.81.6:6443 --token ncxrog.od0ao7t7d4mu1j82 \
>         --discovery-token-ca-cert-hash sha256:15bab2a7e6eaeb701a61c15aa98cf31a78258687dc0c50a65316662dec176535
[preflight] Running pre-flight checks
[preflight] Reading configuration from the cluster...
[preflight] FYI: You can look at this config file with 'kubectl -n kube-system get cm kubeadm-config -o yaml'
[kubelet-start] Writing kubelet configuration to file "/var/lib/kubelet/config.yaml"
[kubelet-start] Writing kubelet environment file with flags to file "/var/lib/kubelet/kubeadm-flags.env"
[kubelet-start] Starting the kubelet
[kubelet-start] Waiting for the kubelet to perform the TLS Bootstrap...

This node has joined the cluster:
* Certificate signing request was sent to apiserver and a response was received.
* The Kubelet was informed of the new secure connection details.

Run 'kubectl get nodes' on the control-plane to see this node join the cluster.
```

Copy the config file to Jenkins master or the local file manager and save it

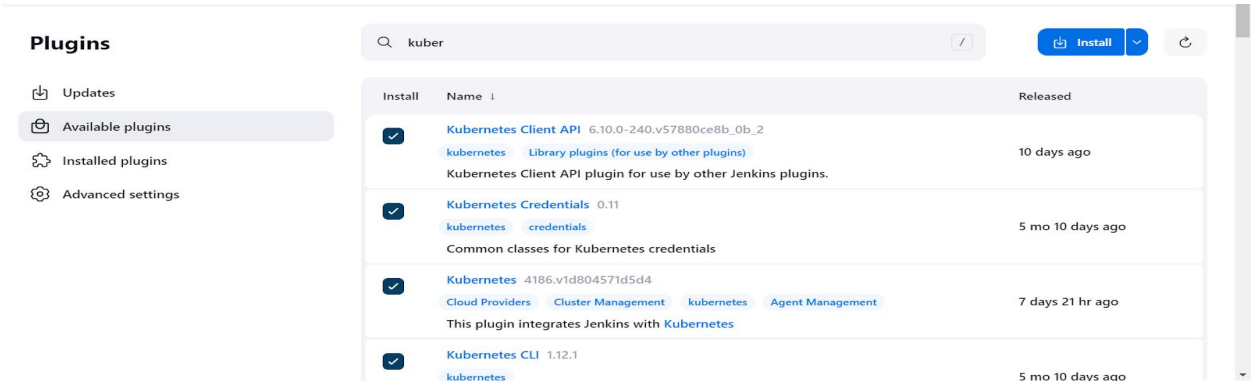
```
ubuntu@K8s-Master:~/.kube$ cat config
apiVersion: v1
clusters:
- cluster:
    certificate-authority-data: LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSURCVENDQWUyZ0F3SUJBZ0lJWHRib3pJUkRBRDh3RFFZSktvWklodmNOQVFFTEJRQXdGVEVUTUJFR0E
xVUUKQXhNS2EzVmlaWEp1WlhSbGN6QWVGdzB5TkRBeU1UTXhNRE0yTWpSYUZ3MHpOREF5TVRBeE1EUXhNalJhTUJVeApFekFSQmdOVkJBTVRDbXQxWW1WeWJtVjBaWE13Z2dFaU1BMEdDU3FHU0liM0
RRRUJBUVVBQTRJQkR3QXdnZ0VLCkFvSUJBUURIMEhBWS8zcmlZZGJ2bVVFZ28wNkJud2RoWVllN05VWGc0MzJNSGl5RFZzWEJwVzhCNkM4OVpaV0kKVDFkQzVPZ3FkTEZIOVFDSU95M1FTd2RLN01Ib
kR6ZWFSbHNieFRMVjB2QS8zcUFoSVhtOXBvQmJtcy9JTjVHagppYzBsK1FJVGRPdHVqZndlaTJJNUdlaHlMcFdhVEdlUmFtTTBjZXlqL25DNEFaVlJsZ0t1R0psSEFaNlNod0lBClRPSmoyVEJCQVRm
WGRUR0pGWEJmcEg0ZnN6MTdhL3lJY3hnWTNFemVuUnRvNElobWlyVTBVSDl2anBmNkFaNzgKYkRXYXZVUTRtcUR0ckt5dlBiNUNtVHhnT0hocGlpNldkYXdMc0luWS9hK0JqSDFWMGZOcXJsdmVLQVZ
yakF2cwo5eE5qQmttSU44ZElVWWVGNkgwOWpVOWp6WVF0QWdNQkFBR2pXVEJYTUE0R0ExVWREd0VCL3dRRUF3SUNwREFQCkJnTlZIUklCQWY4RUJUQURBUUgvTUIwR0ExVWREZ1FXQkJTS1AzREtIUz
IyRXlxU0xyZDFac1dxdnNxV0lqQVYKQmdOVkhSRUVEakFNZ2dwcmRXSmxjbTVsZEdWek1BMEdDU3FHU0liM0RRRUJDdlVBQTRJQkFRQS9IRjA3WjlMZQphUTgxOFZtOHBPYzg0TzNrbHFHRVRFNlBOW
EVrcFplOGU0alBZTC8wWUsyRStMZkpVT0ltcHlya1YwVlB6RERxCjQ2NDcxN1k0UDBISUdxbSt2bTNWdmZsL1hSa2RDZ2FxVE03T3M0WDhPQk9sa3hheDMzbWlwcFpDVHdjOW1OcnIKSy8wa2JVbTNm
OWRUaURmcCtGTGJKRE1jbm9XQlNhOE1uYnY1S1FLVTZvdnpFczZkTFlGNW5za3dmd1lZZmJTTWpVdEdhSHZBSFJLaFp2aDBJbnB0c1BXZlpqcC9tUExRdW9LSXAxZkdvNXVwU2hYWi91OFVLM3VvUlE
wZDFGd0hECktwSmxxOENjS3ZNVGd4ZWtuQVArU0hJdlIONjFoVGN1VTNLb3J3cmV1NUhYeUw4Y0xlOUo0WFZxcDU5T213QWEKeFFzSmZFQTJVNDFTCi0tLS0tRU5EIENFUlRJRklDQVRFLS0tLS0K
    server: https://172.31.81.6:6443
  name: kubernetes
```

```
preferences: {}
users:
- name: kubernetes-admin
  user:
    client-certificate-data: LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSURJVENDQWdtZ0F3SUJBZ0lJWTdxdkdxK0F3M0l3RFFZSktvWklodmNOQVFFTEJRQXdGVEVUTUJFR0ExVU
UKQXhNS2EzVmlaWEp1WlhSbGN6QWVGdzB5TkRBeU1UTXhNRE0yTWpSYUZ3MHlOVEF5TVRJeE1EUXhNamRhTURReApGekFWQmdOVkJBb1REbk41YzNSbGJUcHRZWE4wWlhKek1Sa3dGdlEVlFRREV4Q
nJkV0psY2OlbGRHVnpMV02rCmJXbHVNSUlCSWpBTkJna3Foa2lHOXcwQkFRRUZBQU9DVE4QU1JSUJDZ0tCQVFQXBxZVFjVUp6TFpGdzFWMTUKNHolV0J1dC9qYlhCcnBBTVh2S3JGMXY3WU1qbHRz
dmswRmlKU2lDdTRlWlBCYTdxU2FGTVhmNFpNN2VGb0RWNQp5ZlhQcjF1Z2l6L05nNjFselJISGxSRkxGbnZzM2xHNjZBNkwrYmZjaXVXZ3FuVWlSVXl0a3pkT3puS0RLNVc5CnduRlRwcXVtNlpwTkx
KeW9yOHF0N3dncThDTUE2ZTJwNjd3WGszMmxkWGZkZTNDRmtwYUd3ZEVTcXFIYVFTSGgKcXRXanRPWmthSUNVeHlHWVlOM25wOFRJYVB3NUVYdDRaYm5HbVhJZ3J2eXAxNUZhWm9paGd6MlBETjVQVW
5MOQpBUzZzdWozS3RDUlJqZHphL3JQYWl0OTBmZmlPY04rWFZIMDBKWG1qaHR0YVc3QnZOT1lHWnRJVUxYZ2F4eUJmCmNhaXdrd0lEQVFBQm8xWXdWREFPQmdOVkhROEJBZjhFQkFNQ0JhQXdFdlEV
lIwbEJBd3dDZ1lJS3dZQkJRVUgKQXdJd0RBWURWUjBUQVFIL0JBSXdBREFmQmdOVkhTTUVHREFXZ0JTS1AzREtIUzIyRXlxU0xyZDFac1dxdnNxVwpJakFOQmdrcWhraUc5dzBCQVFzRkFBTONBUUVB
a0l2UHV5dzJUS05GcXhXMkVsNEJBN2lYMEpwNFBaSktWZnB3Cks0Q2dES1E5Nj10aTU0UEd3bjBaZTdRdXVIbmZpb0hrcjFJODVUZHo4b0RPR29aRXVJclhPVXkwbDNNZkFRS04KNzQ3TnUxdERQR2t
EZGlkS1MrSUJCWndkV2NpVC9CYzJTTzhEbGN3Q3ZZRnhFZVNQcjZrbzVWRjVEWFVyZnovWgpQdGE5UmhmQ05hbXA0TUdoWG9MakRHTE5QclRpNzBBQjNQbVpKSm5SSkFRU0NaankvVHJMVmVJOVNwZm
RjRU1sCmFweDdxV3ZuQlFZa2lmc2pmQTlhdkxKRmsxcmpRUUdKajd0cVNHMTFBbUU1OWdCYkt1Ty80dENDc2NFeTNZblgKZmJlUW5Od2slNFRwM1hOL2lPQlpBWmcvYUtMRVpWUlRadjlMMFFtNDh4b
XFGU2JVdnc9PQotLS0tLUVORCBDRVJUSUZJQ0FURS0tLS0tCg==
    client-key-data: LS0tLS1CRUdJTiBSU0EgUFJVkFURSBLRVktLS0tLQpNSUlFcFFJQkFBS0NBUUVBcHFlUWNVSnpMWk23MVYxNTR6NVdCdXQvamNYQnJwQUlYdktyRjF2N1lNamx0c3ZrCj
BGaUpTbUN1NGVaUEJhN3FTYUZNWGY0Wk03ZUZvRFY1eWZYUHIxdWdpei9OZzYxbHpSSEhsUkZMRm52czNsRzYKNkE2TCtiZmNpdVdncW5VaVJVeXRremRPem5LREs1VzI3bkZUcHF1bTdacE5MSnlvc
jhxdDd3Z3E4Q01BNmUycAo2N3dYazMybGRYZmRlMONGa3BhR3dkRVNxcUhhUVNIaHF0V2p0T1prYUlDVXh5R1lZTjNucDhUSWFQdzVFWHQ0ClpibkdtWElncnZ5cDElRmFab2loZ3oyUERONVBVbkw5
QVM2c3VqM0t0Q1JSamR6YS9yUGFpdDkwZmZpT2NOK1gKVkgwMEpYbWpodHRhVzdCdnRPWUdadElVTFhnYXh5QmZjYWl3a3dJREFRQUJBb0lCQUYldEFPVjdmaUVxa0NFRQpwclJBRktRMUdQK3VOaEI
vWTlNRWNqWjVFcFNESDgwVDkrQ28yWmsvOExia05KQUd6STdOaXFTVERacUUrSHlPCjhDT2pXSHFHVXcrekpmSGU5SmhkNDVTdHlMckRtckZZNDZzS1d2c3gxdzhJRHFHbVhyV2ZQWlgxTjN5MzIyUj
kKb3BnaWVBQXVzQlZlWVMzTzBiRlJDSDR3Ul12cmowb0JoajFZUHpHcTRaUzhpN3hSa3Awc0UyNDZNTHdnYWphgoxYzkyVWlHb3FSRWxHT2x1WlM1QytvTnZUN1Qyby9FU2clWEJzGdFUDF3ZOhEO
Uk0OG8zcOJiTVBHdlQ2OVJmCnJneUpqcUtvYWExUFFKcWdUQXZrTVdCZitidlRQVVphSHJtOFFucUZ4V2tuZUc5Qk4KSTFxWE9ka0NnWUVBMFQ5RThLQkNTV25IOG05bklP
aEZxRE5kbDZyN1dNcVdRMHhQdkc1dkpYOXJIS2ZEWFFYUgpHVFgwaTg2TWFhalluUy9FancxcXNRblhVZlBWWFU2SGtZTlBxeWRROFViYVpWMnBUbDlFMTFXaDJkVUhLTENoCjAwWXZEcVhMZTJtV2R
nOWE3dWpaalJwR2xTSVpYSnVkV0Y4WUYrMVloZHBOZWtOSXYyVUtEZlVDZllFQXkrUUsKVHdKQnQ4SHdUZjRrbElVMXVUVURBUHY2VlJZUkV6ZEVLRmZZVXkwRVdmWXViUkJZd2pPaHJOTCsrYmJPbj
dJMQo4NzhROUk1Ujd3VXF3ZHRsQnpJeENmTHBHVW5WOGphZHpoRjRYSnoySkRpU25acDdjNFFtTXoydDF1ZzRNdW1TC1B1WlpHbGRtQzdUQWs1ZE9mblDPM2x1TGg0ZjBUSW0wN0NmUjUyY0NnWUVBc
2tPZG53WlpFcVFzOU15VjlDT2QKM0d0T1MvMGR2d3B3czRyYXoyU2lDaWNnZ0x3b0F5TUdmZHpxcHdUanVNeVhyS3BhWWFrRGdTb3FKNlFlWFN5ZQo4SWdYSW5BUFZFNlJseFFSUGpTSnVJNDZvaE1M
MDAyUHdDZWx4V3ZZREI3clVVOUpvQW45K05HUzJsQy9uSmJ5Cm1LdORJYmRSMkU4N1A1bk5zMGtJSk5VQ2dZRUF1ZnIwL0VUT3RWT09ZMVlqQUNFemRCRkxBQ2JZblZKamlPZWMKb0VBZzd4M3REakp
rL3M4ZVoyYTVYdS9IN1BnM1dhd3hVbmxDZWhFbjQ2TU84VzJ5cGZHK2taaWx6NDlUEZaZApjNDZ6LzZhZFQwVGJXQytGMUVVQ28xTHJzbWIwZkZ4YUxrM25rUU9QN2JXZXhBSUljNFhBSUV2WTVuTz
QvK2ZNCnhHMCtKcGtDZllFQWk1RVNGWkxneE9yTTZ1T0R6dElyS1dlNGx3SEFKQlVHYUxpQXluaVVoQnA3cGY4UnQvalkKb0VHZ1doYnRDdUFIQWNhK3dvbnBPdDBONExxHWlFteEVTcEt5MVRHcVAzQ
2JYc01VeWlJQmdMRDQwNlp3dDJjTApoQmowS2JDRmwvdFdialNFTS8zSzFadGVBSmU0SDBlckxHRHlGL2w2VDhTcjFCVDQ2T2hEOGdRPQotLS0tLUVORCBSU0EgUFJJVkFURSBLRVktLS0tLQo=
```

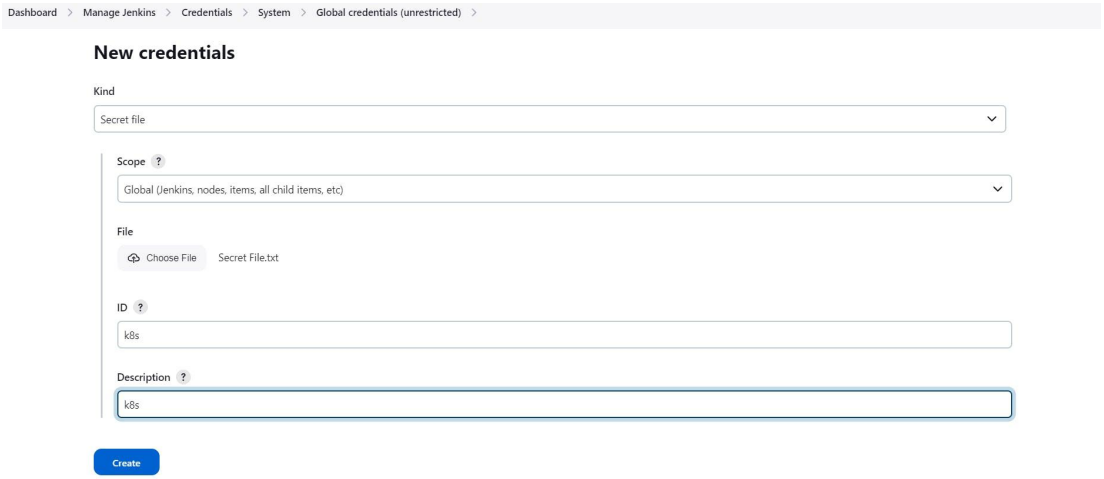
copy it and save it in documents or another folder save it as secret-file.txt

Note: create a secret-file.txt in your file explorer save the config in it and use this at the kubernetes credential section.

Install Kubernetes Plugin, Once it's installed successfully



goto manage Jenkins --> manage credentials --> Click on Jenkins global --> add credentials



Step9D: Install Helm & Monitoring K8S using Prometheus and Grafana

install the helm in Kubernetes Master

curl -fsSL -o get_helm.sh https://raw.githubusercontent.com/helm/helm/main/scripts/get-helm-3
chmod 700 get_helm.sh
./get_helm.sh

See the Helm version:
helm version --client

We need to add the Helm Stable Charts for your local client. Execute the below command:

helm repo add stable https://charts.helm.sh/stable

Add Prometheus Helm repo:

helm repo add prometheus-community https://prometheus-community.github.io/helm-charts

Create Prometheus namespace:

kubectl create namespace prometheus

Install kube-Prometheus-stack

Below is the command to install kube-prometheus-stack. The helm repo kube-stack-Prometheus (formerly Prometheus-operator) comes with a Grafana deployment embedded.

helm install stable prometheus-community/kube-prometheus-stack -n prometheus

Let's check if the Prometheus and Grafana pods are running or not

kubectl get pods -n prometheus

Now See the services

kubectl get svc -n prometheus

```
ubuntu@K8s-Master:~$ kubectl get nodes
NAME         STATUS   ROLES           AGE     VERSION
k8s-master   Ready    control-plane   2m37s   v1.28.2
k8s-slave    Ready    <none>          19s     v1.28.2
ubuntu@K8s-Master:~$ curl -fsSL -o get_helm.sh https://raw.githubusercontent.com/helm/helm/main/scripts/get-helm-3
ubuntu@K8s-Master:~$ chmod 700 get_helm.sh
ubuntu@K8s-Master:~$ ./get_helm.sh
Downloading https://get.helm.sh/helm-v3.14.0-linux-amd64.tar.gz
Verifying checksum... Done.
Preparing to install helm into /usr/local/bin
helm installed into /usr/local/bin/helm
ubuntu@K8s-Master:~$ helm version --client
version.BuildInfo{Version:"v3.14.0", GitCommit:"3fc9f4b2638e76f26739cd77c7017139be81d0ea", GitTreeState:"clean", GoVersion:"go1.21.5"}
ubuntu@K8s-Master:~$
ubuntu@K8s-Master:~$ helm repo add stable https://charts.helm.sh/stable
"stable" has been added to your repositories
ubuntu@K8s-Master:~$ helm repo add prometheus-community https://prometheus-community.github.io/helm-charts
"prometheus-community" has been added to your repositories
ubuntu@K8s-Master:~$ kubectl create namespace prometheus
namespace/prometheus created
ubuntu@K8s-Master:~$ helm install stable prometheus-community/kube-prometheus-stack -n prometheus
NAME: stable
LAST DEPLOYED: Tue Feb 13 10:46:01 2024
NAMESPACE: prometheus
STATUS: deployed
REVISION: 1
NOTES:
kube-prometheus-stack has been installed. Check its status by running:
  kubectl --namespace prometheus get pods -l "release=stable"

Visit https://github.com/prometheus-operator/kube-prometheus for instructions on how to create & configure Alertmanager and Prometheus instances using th
e Operator.
```

```
ubuntu@K8s-Master:~$ kubectl get pods -n prometheus
NAME                                                     READY   STATUS    RESTARTS   AGE
alertmanager-stable-kube-prometheus-sta-alertmanager-0   2/2     Running   0          24s
prometheus-stable-kube-prometheus-sta-prometheus-0       1/2     Running   0          22s
stable-grafana-6889bcb489-srdtc                          2/3     Running   0          33s
stable-kube-prometheus-sta-operator-c548659cd-tj6dg      1/1     Running   0          33s
stable-kube-state-metrics-7fdbdb97cb-gqm54               1/1     Running   0          33s
stable-prometheus-node-exporter-fhthc                    1/1     Running   0          33s
stable-prometheus-node-exporter-qpggq                    1/1     Running   0          33s
ubuntu@K8s-Master:~$ kubectl get svc -n prometheus
NAME                                      TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)                      AGE
alertmanager-operated                     ClusterIP   None             <none>        9093/TCP,9094/TCP,9094/UDP   28s
prometheus-operated                       ClusterIP   None             <none>        9090/TCP                     26s
stable-grafana                            ClusterIP   10.102.174.220   <none>        80/TCP                       37s
stable-kube-prometheus-sta-alertmanager   ClusterIP   10.110.121.0     <none>        9093/TCP,8080/TCP            37s
stable-kube-prometheus-sta-operator       ClusterIP   10.109.78.45     <none>        443/TCP                      37s
stable-kube-prometheus-sta-prometheus     ClusterIP   10.108.127.30    <none>        9090/TCP,8080/TCP            37s
stable-kube-state-metrics                 ClusterIP   10.102.180.145   <none>        8080/TCP                     37s
stable-prometheus-node-exporter           ClusterIP   10.106.61.84     <none>        9100/TCP                     37s
```

This confirms that Prometheus and grafana have been installed successfully using Helm.

To make Prometheus and grafana available outside the cluster, use LoadBalancer or NodePort instead of ClusterIP.

Edit Prometheus Service

kubectl edit svc stable-kube-prometheus-sta-prometheus -n prometheus

```
  ipFamilyPolicy: SingleStack
  ports:
  - name: http-web
    nodePort: 32548
    port: 9090
    protocol: TCP
    targetPort: 9090
  - appProtocol: http
    name: reloader-web
    nodePort: 32558
    port: 8080
    protocol: TCP
    targetPort: reloader-web
  selector:
    app.kubernetes.io/name: prometheus
    operator.prometheus.io/name: stable-kube-prometheus-sta-prometheus
  sessionAffinity: None
  type: LoadBalancer
status:
  loadBalancer:
    ingress:
    - hostname: a785b472f5aaa413ea0c5e1576444326-1069929702.us-east-1.elb.amazonaws.com
                                                                                58,1          Bot
```

Edit Grafana Service

kubectl edit svc stable-grafana -n prometheus

```
  name: stable-grafana
  namespace: prometheus
  resourceVersion: "6259"
  uid: 972955be-5d44-4ab5-b96b-c21d0bd5ecfb
spec:
  clusterIP: 10.100.206.88
  clusterIPs:
  - 10.100.206.88
  internalTrafficPolicy: Cluster
  ipFamilies:
  - IPv4
  ipFamilyPolicy: SingleStack
  ports:
  - name: http-web
    port: 80
    protocol: TCP
    targetPort: 3000
  selector:
    app.kubernetes.io/instance: stable
    app.kubernetes.io/name: grafana
  sessionAffinity: None
  type: LoadBalancer
```

kubectl get svc -n prometheus

Verify if the service is changed to LoadBalancer and also get the Load BalancerPorts OR NodePort.

```
ubuntu@ip-172-31-42-150:~$ kubectl get svc -n prometheus
NAME                                      TYPE           CLUSTER-IP       EXTERNAL-IP
S)                        AGE
alertmanager-operated                     ClusterIP      None             <none>
TCP,9094/TCP,9094/UDP     6m20s
prometheus-operated                       ClusterIP      None             <none>
TCP                       6m20s
stable-grafana                            LoadBalancer   10.100.206.88    a972955be5d444ab5b96bc21d0bd5ecf-1946628392.us-east-1.elb.amazonaws.com
379/TCP                   6m27s
stable-kube-prometheus-sta-alertmanager   ClusterIP      10.100.33.74     <none>
TCP,8080/TCP              6m27s
stable-kube-prometheus-sta-operator       ClusterIP      10.100.145.137   <none>
CP                        6m27s
stable-kube-prometheus-sta-prometheus     LoadBalancer   10.100.146.3     a785b472f5aaa413ea0c5e1576444326-1069929702.us-east-1.elb.amazonaws.com
32548/TCP,8080:32558/TCP  6m27s
stable-kube-state-metrics                 ClusterIP      10.100.112.226   <none>
TCP                       6m27s
stable-prometheus-node-exporter           ClusterIP      10.100.114.191   <none>
TCP                       6m27s
```

Access Grafana UI in the browser

Get the external IP from the above screenshot and put it in the browser

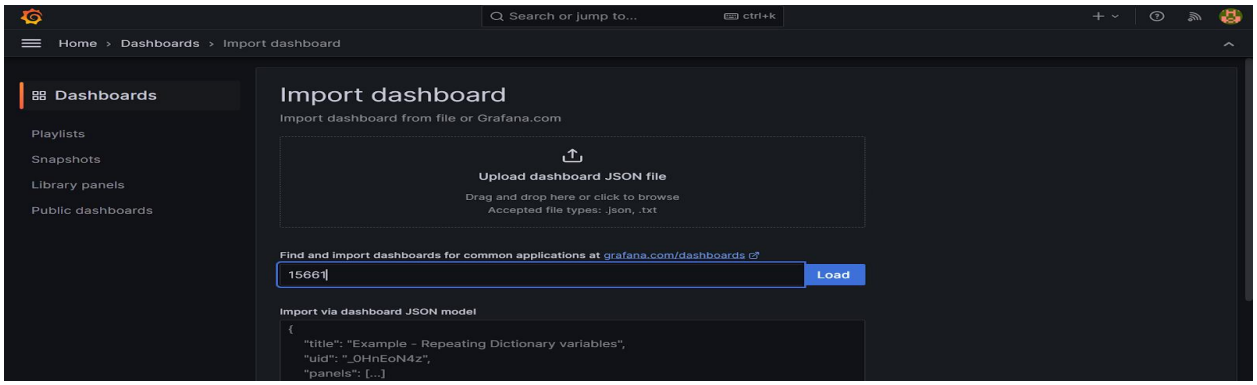
Login to Grafana

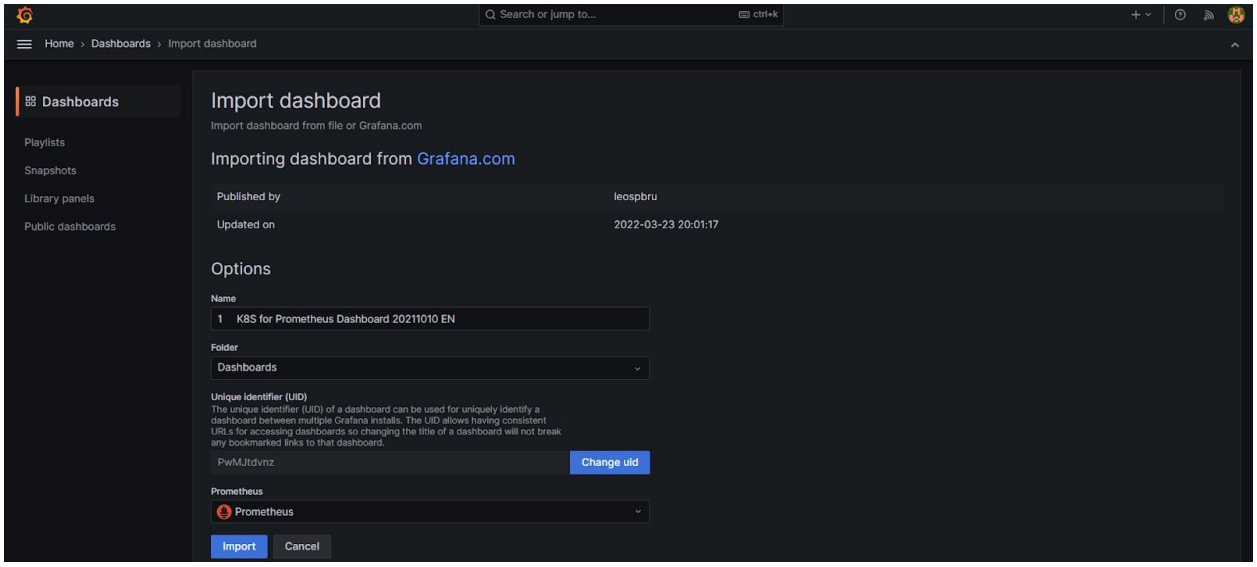
UserName: admin
Password: prom-operator

For creating a dashboard to monitor the cluster:

Click the '+' button on the left panel and select ‘Import’.

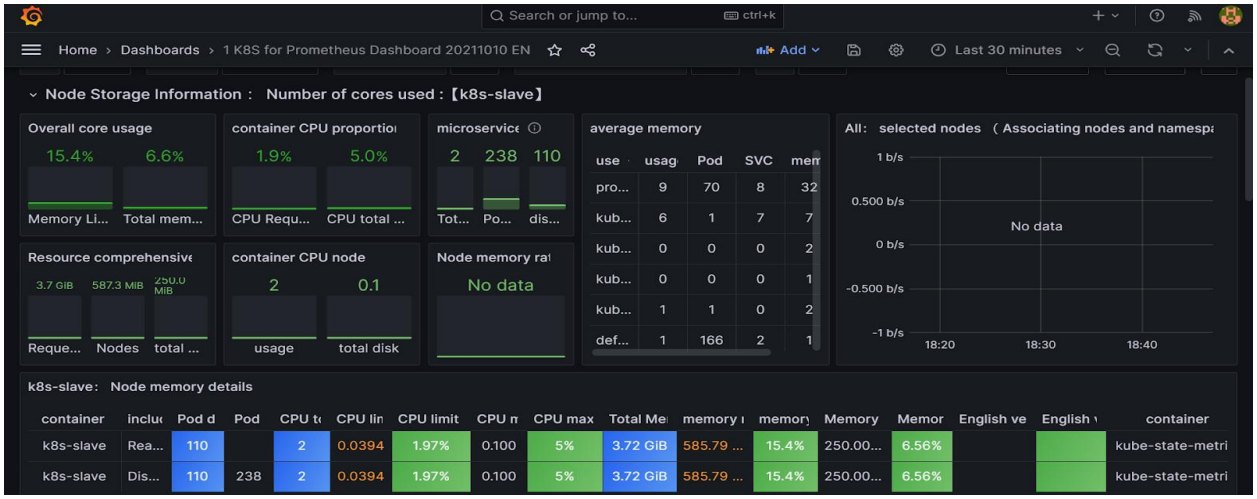
Enter the 15661 dashboard id under Grafana.com Dashboard.Click ‘Load’.





Click ‘Import’.

This will show the monitoring dashboard for all cluster nodes



How to Create Kubernetes Cluster Monitoring Dashboard?

For creating a dashboard to monitor the cluster:

Click the '+' button on the left panel and select ‘Import’.

Enter 3119 dashboard ID under Grafana.com Dashboard.

Click ‘Load’.

Select ‘Prometheus’ as the endpoint under the Prometheus data sources drop-down.

Click ‘Import’.

Create a POD Monitoring Dashboard

For creating a dashboard to monitor the cluster:

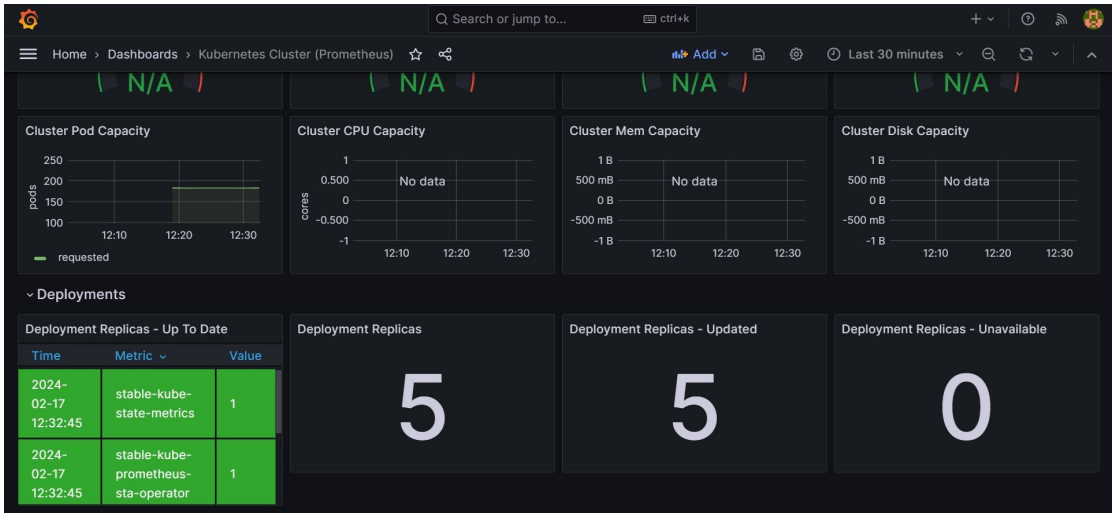
Click the '+' button on the left panel and select ‘Import’.

Enter 6417 dashboard ID under Grafana.com Dashboard.

Click ‘Load’.

Select ‘Prometheus’ as the endpoint under the Prometheus data sources drop-down.

Click ‘Import’.



Step9E: K8S Deployment

Let's Create a Shared Jenkins library file for K8s deploy and delete

Name **kubeDeploy.groovy**

```
def call() {

    withKubeConfig(caCertificate: '', clusterName: '', contextName: '', credentialsId: 'k8s', namespace: '', restrictKubeConfigAccess: false, serverUrl: '') {

        sh "kubectl apply -f deployment.yml"
```

```
    }
}
```

To delete deployment

Name **kubeDelete.groovy**

```
def call() {
    withKubeConfig(caCertificate: '', clusterName: '', contextName: '', credentialsId: 'k8s', namespace: '', restrictKubeConfigAccess: false, serverUrl: '') {
        sh "kubectl delete -f deployment.yml"
    }
}
```

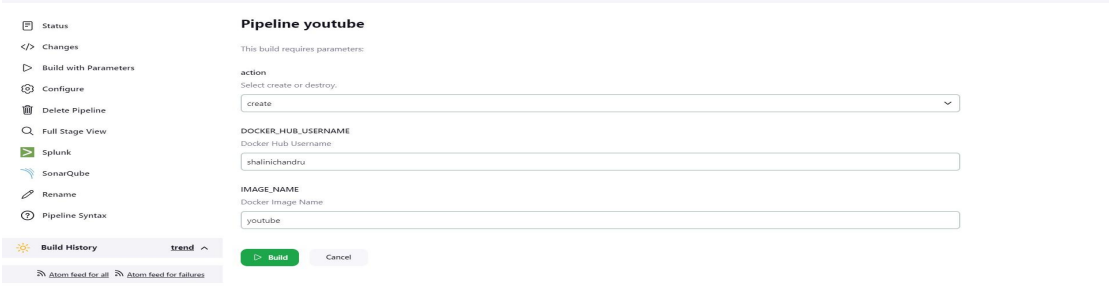
Let's push them to GitHub

```
git add .
git commit -m "message"
git push origin main
```

The final stage of the Pipeline

```
stage('Kube deploy'){
        when { expression { params.action == 'create'}}
            steps{
                kubeDeploy()
            }
        }
        stage('kube deleter'){
        when { expression { params.action == 'delete'}}
            steps{
                kubeDelete()
            }
        }
```

Build Now with parameters 'create'



It will apply the deployment

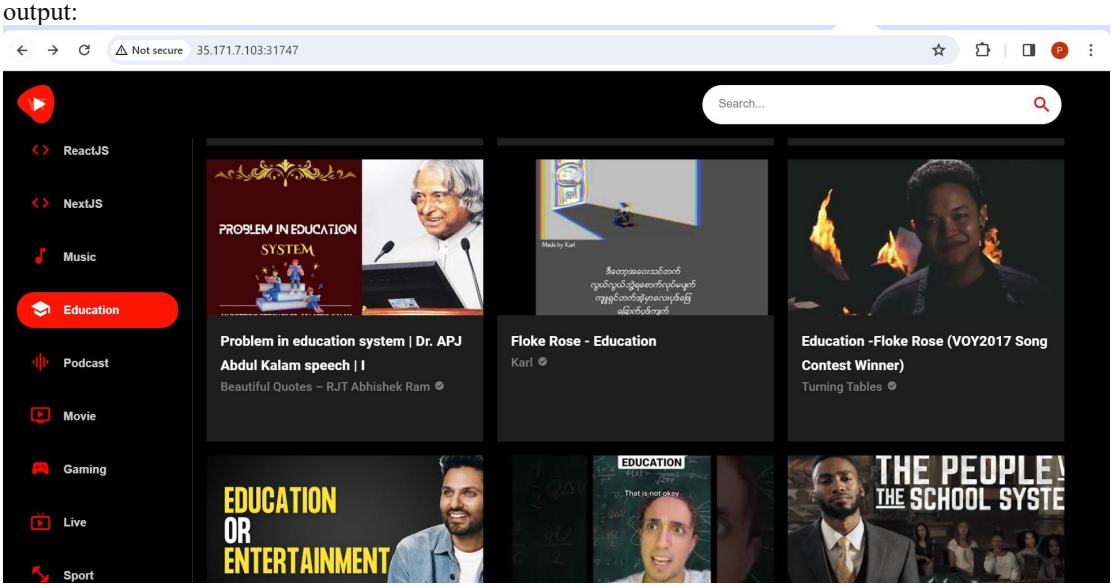
kubectl get all (or)

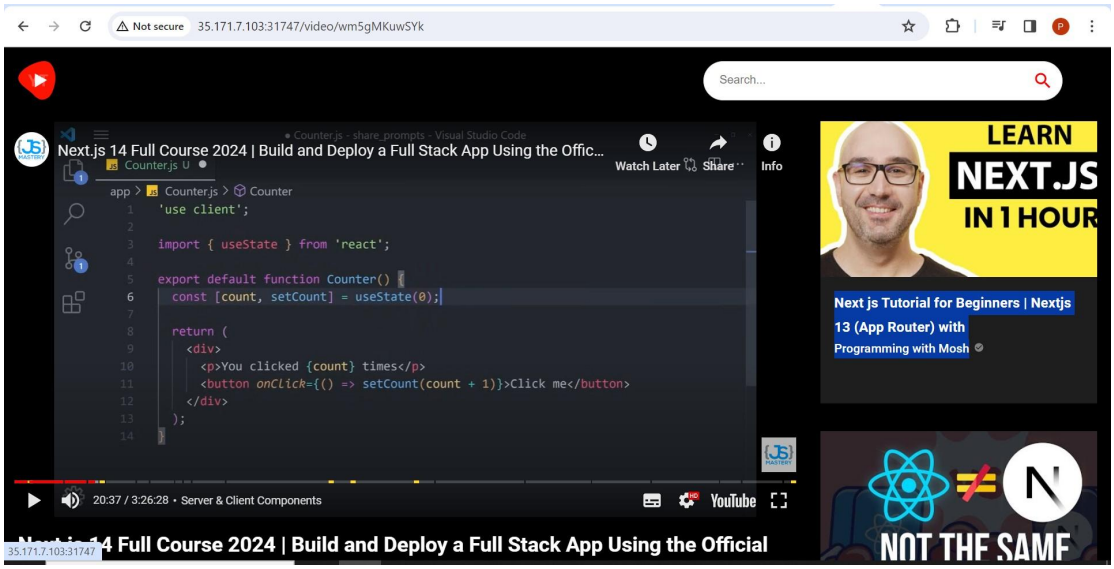
kubectl get svc

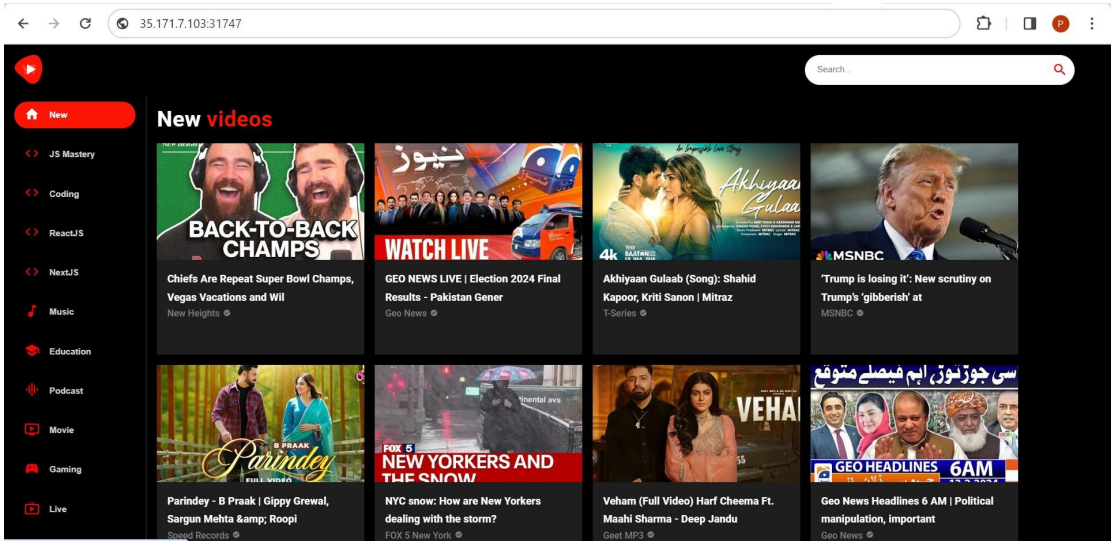
<kubernetes-worker-ip:svc port>

```
NAME                                    READY   STATUS    RESTARTS   AGE
pod/my-app-deployment-97f7c58bc-kx5dk   0/1     Error     0          15m
pod/my-app-deployment-97f7c58bc-t5zxx   1/1     Running   0          8m37s
pod/my-app-deployment-97f7c58bc-wbdfk   0/1     Error     0          14m

NAME                     TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)        AGE
service/kubernetes       ClusterIP   10.96.0.1       <none>        443/TCP        3d20h
service/my-app-service   NodePort    10.97.215.183   <none>        80:31747/TCP   15m

NAME                                READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/my-app-deployment   1/1     1            1           15m

NAME                                           DESIRED   CURRENT   READY   AGE
replicaset.apps/my-app-deployment-97f7c58bc    1         1         1       15m
ubuntu@K8s-Master:~$
```

output:







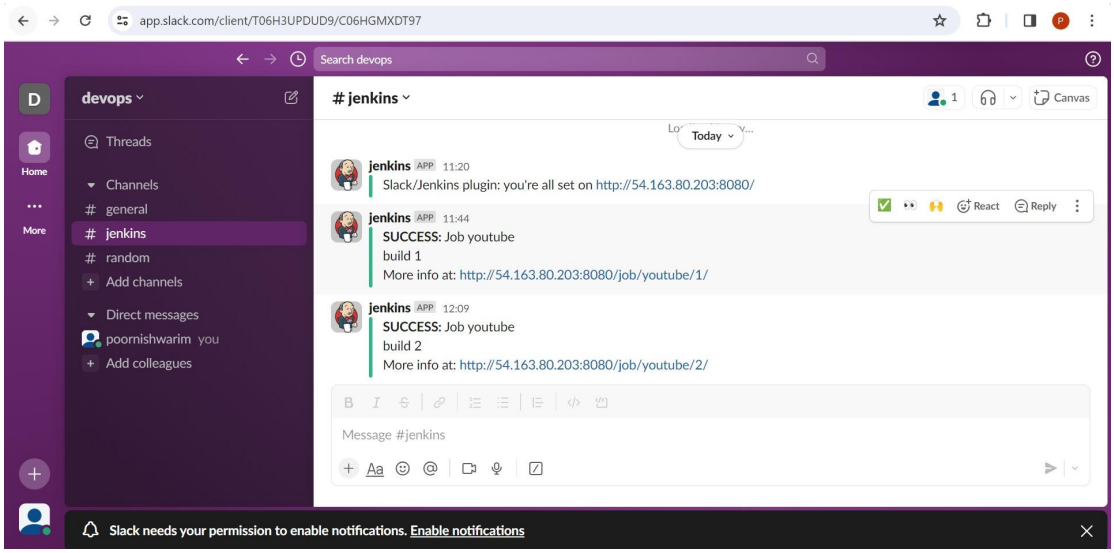
Build with parameter 'delete'

It will destroy Container and Kubernetes deployment.

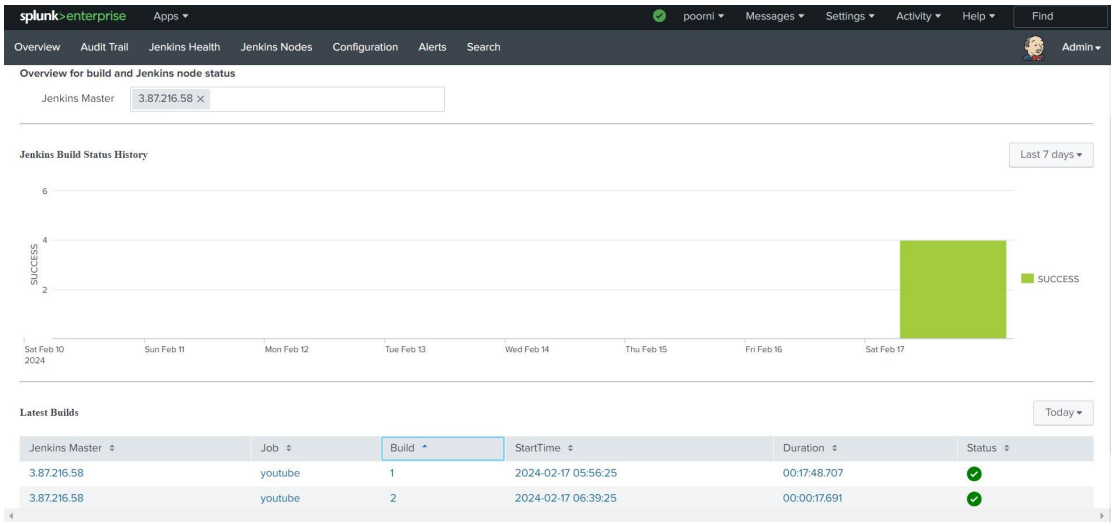
Stage view

Declarative: Tool Install	clean workspace	checkout from Git	sonarqube Analysis	sonarqube QualitGate	Npm	Trivy file scan	OWASP FS SCAN	Docker Build	Trivy iamge	Run container	Remove container	Kube deploy	kube deleter	Declarative: Post Actions
325ms	572ms	998ms	32s	848ms	25s	7s	4min 30s	3min 23s	4min 10s	2s	1s	4s	678ms	205ms
192ms	378ms	866ms					11s				1s		678ms	179ms
458ms	766ms	1s	32s	848ms (paused for 2s)	25s	7s	8min 50s	3min 23s	4min 10s	2s		4s		231ms

Slack Notifications:



Splunk:



Pipeline:

```
@Library('Jenkins_shared_library') _

def COLOR_MAP = [

    'FAILURE' : 'danger',

    'SUCCESS' : 'good'

]
```

```
pipeline{
    agent any
    parameters {
        choice(name: 'action', choices: 'create\ndelete', description: 'Select create or destroy.')

        string(name: 'DOCKER_HUB_USERNAME', defaultValue: 'shalinichandru', description: 'Docker Hub Username')
        string(name: 'IMAGE_NAME', defaultValue: 'youtube', description: 'Docker Image Name')
    }
    tools{
        jdk 'jdk17'
        nodejs 'node16'
    }
    environment {
        SCANNER_HOME=tool 'sonar-scanner'
    }
    stages{
        stage('clean workspace'){
            steps{
                cleanWorkspace()
            }
        }
        stage('checkout from Git'){
            steps{
                checkoutGit('https://github.com/chandrushal/Youtube-clone-app.git', 'main')
            }
        }
        stage('sonarqube Analysis'){
        when { expression { params.action == 'create'}}
            steps{
```

```
            sonarqubeAnalysis()
        }
    }
    stage('sonarqube QualitGate'){
    when { expression { params.action == 'create'}}
        steps{
            script{
                def credentialsId = 'Sonar-token'
                qualityGate(credentialsId)
            }
        }
    }
    stage('Npm'){
    when { expression { params.action == 'create'}}
        steps{
            npmInstall()
        }
    }
    stage('Trivy file scan'){
    when { expression { params.action == 'create'}}
        steps{
            trivyFs()
        }
    }
    stage('OWASP FS SCAN') {
        steps {
            dependencyCheck additionalArguments: '--scan ./ --disableYarnAudit --disableNodeAudit', odcInstallation: 'DP-Check'
            dependencyCheckPublisher pattern: '**/dependency-check-report.xml'
        }
```

```
}
stage('Docker Build'){
when { expression { params.action == 'create'}}
    steps{
        script{
            def dockerHubUsername = params.DOCKER_HUB_USERNAME
            def imageName = params.IMAGE_NAME


            dockerBuild(dockerHubUsername, imageName)
        }
    }
}
stage('Trivy iamge'){
when { expression { params.action == 'create'}}
    steps{
        trivyImage()
    }
}
stage('Run container'){
when { expression { params.action == 'create'}}
    steps{
        runContainer()
    }
}
stage('Remove container'){
when { expression { params.action == 'delete'}}
    steps{
        removeContainer()
    }
```

```
        }
        stage('Kube deploy'){
        when { expression { params.action == 'create'}}
            steps{
                kubeDeploy()
            }
        }
        stage('kube deleter'){
        when { expression { params.action == 'delete'}}
            steps{
                kubeDelete()
            }
        }
    }
    post {
    always {
        echo 'Slack Notifications'
        slackSend (
            channel: 'jenkins',  / #change your channel name
            color: COLOR_MAP[currentBuild.currentResult],
            message: "*${currentBuild.currentResult}:* Job ${env.JOB_NAME} \n build ${env.BUILD_NUMBER} \n More info at: ${env.BUILD_URL}"
        )
    }
}
}
```