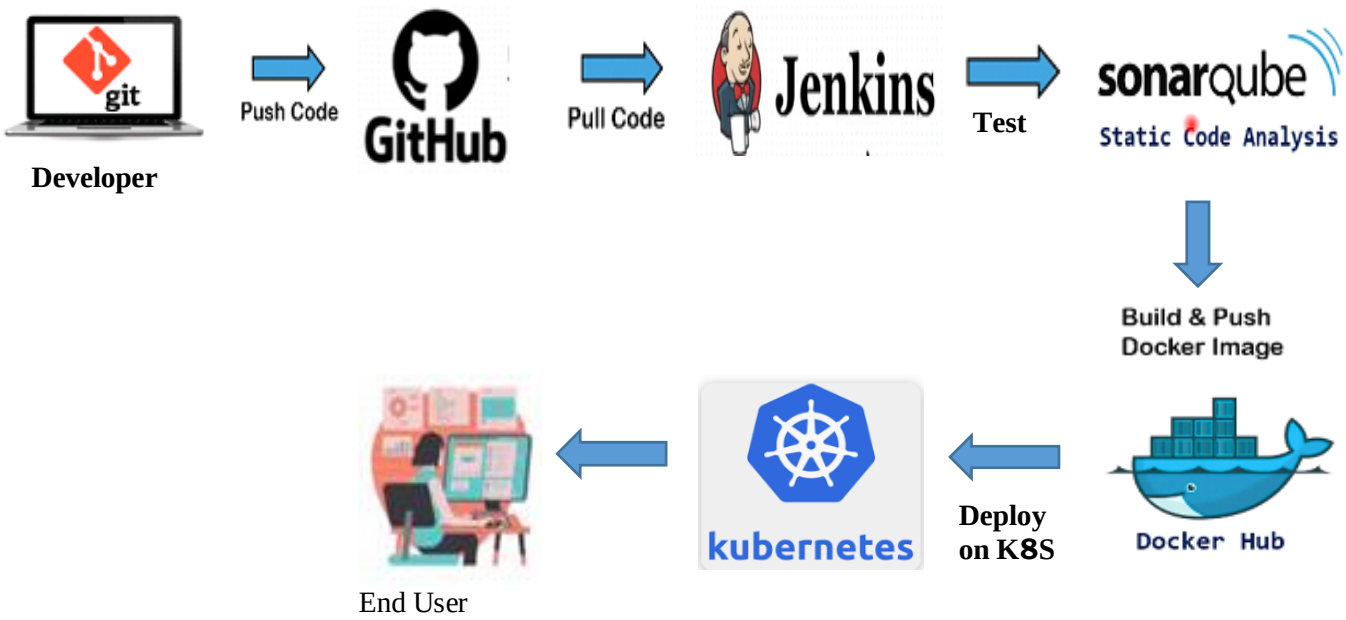
Continuous Integration & Continuous Deployment Pipeline for Web application development



Project Flow Summary

1. Development Phase (Local Development)

- The developer writes code using a local development environment.
- Version control is managed using Git.
- The developer pushes the code to GitHub, which acts as a central repository.

2. Continuous Integration (CI)

- Jenkins is configured to automatically pull the latest code from GitHub.
- Jenkins runs build jobs and executes automated tests to ensure the new changes do not break the application.

3. Code Quality & Security Checks

- SonarQube is used for static code analysis to check for bugs, vulnerabilities, and code quality issues. If issues are detected, the developer is alerted, and fixes are required before proceeding.

4. Build and Containerization

- After passing tests, Jenkins builds the application. The application is then containerized using Docker. A Docker image is built and pushed to Docker Hub, a container registry.

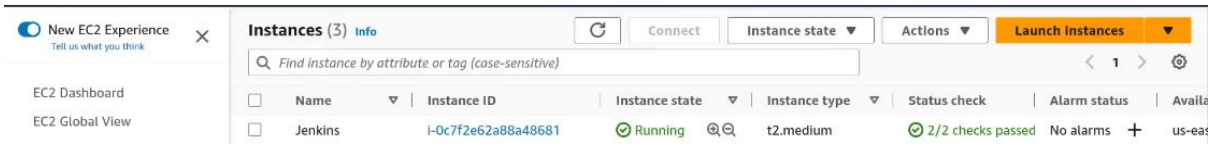
5. Continuous Deployment (CD)

- Once the Docker image is available, Jenkins triggers the deployment.
- The application is deployed to a Kubernetes (K8s) cluster.
- Kubernetes ensures the application is running with high availability and scalability.

6. End-User Access

- The end-user accesses the deployed web application through a browser or client interface.
- Kubernetes manages the traffic and scales the application based on demand.

Step: 1 To Create a Jenkins virtual server in AWS and login to Ubuntu linux with root user.




```
ubuntu@ip-172-31-11-179:~$ sudo su - root
root@ip-172-31-11-179:~# apt update
Hit:1 http://us-east-2.ec2.archive.ubuntu.com/ubuntu focal InRelease
Get:2 http://us-east-2.ec2.archive.ubuntu.com/ubuntu focal-updates InRelease [114 kB]
Get:3 http://us-east-2.ec2.archive.ubuntu.com/ubuntu focal-backports InRelease [108 kB]
Get:4 http://us-east-2.ec2.archive.ubuntu.com/ubuntu focal/universe amd64 Packages [8628 kB]
Get:5 http://us-east-2.ec2.archive.ubuntu.com/ubuntu focal/universe Translation-en [5124 kB]
Get:6 http://us-east-2.ec2.archive.ubuntu.com/ubuntu focal/universe amd64 c-n-f Metadata [265 kB]
Get:7 http://us-east-2.ec2.archive.ubuntu.com/ubuntu focal/multiverse amd64 Packages [144 kB]
Get:8 http://us-east-2.ec2.archive.ubuntu.com/ubuntu focal/multiverse Translation-en [104 kB]
Get:9 http://us-east-2.ec2.archive.ubuntu.com/ubuntu focal/multiverse amd64 c-n-f Metadata [9136 B]
Get:10 http://us-east-2.ec2.archive.ubuntu.com/ubuntu focal-updates/main amd64 Packages [2498 kB]
Get:11 http://us-east-2.ec2.archive.ubuntu.com/ubuntu focal-updates/main Translation-en [424 kB]
Get:12 http://us-east-2.ec2.archive.ubuntu.com/ubuntu focal-updates/main amd64 c-n-f Metadata [16.4 kB]
Get:13 http://us-east-2.ec2.archive.ubuntu.com/ubuntu focal-updates/restricted amd64 Packages [1776 kB]
Get:14 http://us-east-2.ec2.archive.ubuntu.com/ubuntu focal-updates/restricted Translation-en [249 kB]
Get:15 http://us-east-2.ec2.archive.ubuntu.com/ubuntu focal-updates/restricted amd64 c-n-f Metadata [636 B]
```

Step: 2 -> To install Jenkins on Ubuntu linux

```
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
sudo apt-get install jenkins
```

We have to install Java Development Kit before installing Jenkins on Ubuntu linux

```
-# apt search openjdk

# apt install openjdk-17-jre-headless
```

Now the Java Development Kit has been successfully install. To install Jenkins on Ubuntu linux

```
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
sudo apt-get install jenkins
```

```
root@ip-172-31-11-179:~# curl -fsSL https://pkg.jenkins.io/debian-stable/jenkins.io-2023.key | sudo tee \
>   /usr/share/keyrings/jenkins-keyring.asc > /dev/null
root@ip-172-31-11-179:~# 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
root@ip-172-31-11-179:~# sudo apt-get update
```

```
-# sudo apt-get install jenkins
```

To check the processes active or not under Jenkins by using the below command

```
root@ip-172-31-11-179:~# ps -ef | grep jenkins
jenkins     4074       1 46 05:50 ?        00:00:38 /usr/bin/java -Djava.awt.headless=true -jar /usr/share/java/jenkins.war --webroot=/
var/cache/jenkins/war --httpPort=8080
root        4468    1394  0 05:52 pts/0    00:00:00 grep --color=auto jenkins
```

To check the Jenkins installed or not by using the command # systemctl status Jenkins

We will build the Continuous Integration & Continuous Deployment Pipeline based on the five files such as Docker file, Form.html, Main.py, Pod.Yaml and requirement.txt file.

1. **Main.py File**

```python
#uvicorn main:app --reload
from fastapi import FastAPI, Request, Form
from fastapi.templating import Jinja2Templates

app = FastAPI()
templates = Jinja2Templates(directory="/code")

@app.get("/")
def form_post(request: Request):
    return templates.TemplateResponse('form.html', context={'request': request})
```

2. **Requirement.txt**

```
aiofiles==0.5.0
aniso8601==7.0.0
ansible-lint==5.0.7
astroid==2.5.1
async-exit-stack==1.0.1
async-generator==1.10
awscli==1.22.97
boto3==1.21.42
botocore==1.24.42
bracex==2.1.1
certifi==2021.5.30
cffi==1.14.5
chardet==4.0.0
charset-normalizer==2.0.4
click==7.1.2
colorama==0.4.3
commonmark==0.9.1
cryptography==3.4.7
dnspython==2.1.0
docutils==0.15.2
email-validator==1.1.3
enrich==1.2.6
fastapi==0.68.1
graphene==2.1.9
graphql-core==2.3.2
graphql-relay==2.0.1
h11==0.12.0
idna==3.2
isort==5.7.0
itsdangerous==1.1.0
Jinja2==2.11.3
jmespath==0.10.0
lazy-object-proxy==1.5.2
MarkupSafe==2.0.1
mccabe==0.6.1
orjson==3.6.3
packaging==20.9
pathspec==0.8.1
pdfminer.six==20211012
promise==2.3
py4j==0.10.9
pyasn1==0.4.8
PyYAML==5.4.1
requests==2.26.0
rich==10.1.0
rsa==4.5
ruamel.yaml==0.17.4
ruamel.yaml.clib==0.2.2
Rx==1.6.1
s3transfer==0.5.0
six==1.15.0
starlette==0.14.2
tenacity==7.0.0
toml==0.10.2
typing-extensions==3.7.4.3
ujson==4.1.0
urllib3==1.26.4
uvicorn==0.13.4
watchgod==0.7
wcmatch==8.1.2
websockets==8.1
wrapt==1.12.1
yamllint==1.26.1
```

3. **Form.html**

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>k8 Page</title>
</head>
<body bgcolor="Red">
    <h1>Hello k8 Project
</body>
</html>
```

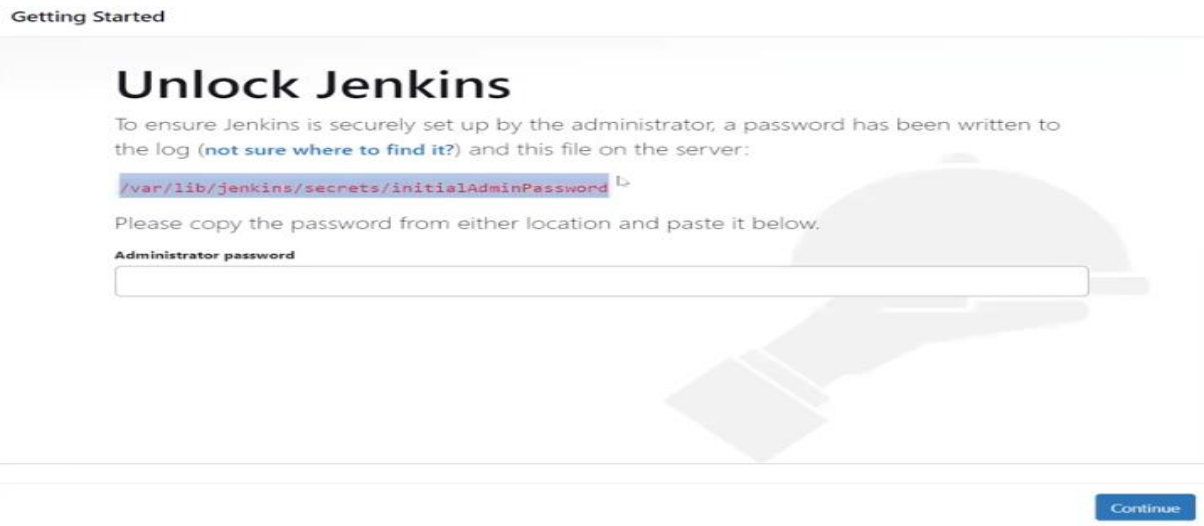
4. **Docker File**

```
FROM python:3.9
WORKDIR /code
COPY ./requirements.txt /code/requirements.txt
RUN pip install --no-cache-dir --upgrade -r /code/requirements.txt
COPY ./main.py /code/main.py
COPY ./form.html /code/form.html
CMD ["uvicorn", "main:app", "--host", "0.0.0.0", "--port", "80"]
```

5. **Pod.Yaml**

```
apiVersion: apps/V1
kind: Deployment
metadata:
  name: loadbalancer-pod
Spec:
  replicas: 3
  selector:
    matchLabels:
      app:  loadbalancer-pod
  template:
    metadata:
      labels:
        app: loadbalancer-pod
    spec:
      containers:
        - name: loadbalancer-pod
          image: <dockerhub username>/image name
          resources:
            limits:
              memory: "256Mi"
              cpu: "500m"
          ports:
          - containerport: 8080
```

To access the Jenkins webpage using Jenkins server “IP address:8080”

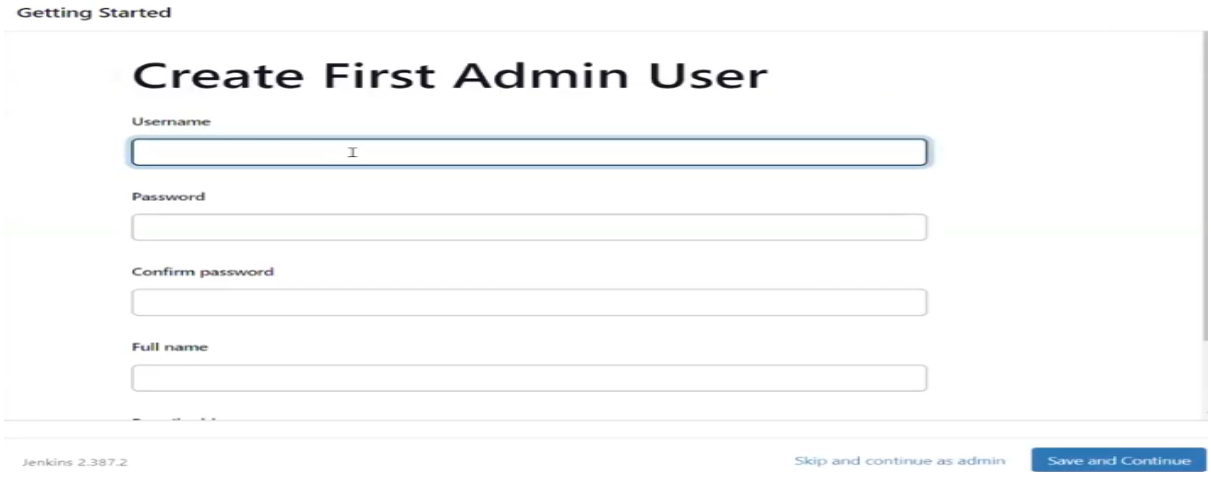


Open the jenkins initial admin password path on ubuntu linux to get the credentials.

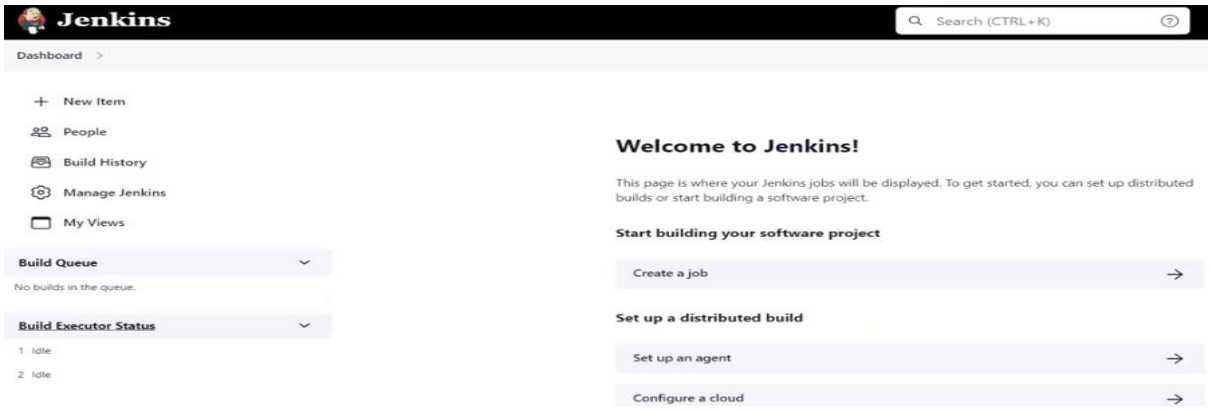
```
root@ip-172-31-11-179:~# cat /var/lib/jenkins/secrets/initialAdminPassword
5f5db8ca74b94f838ccbe
```

Click on “Install suggested plugins”

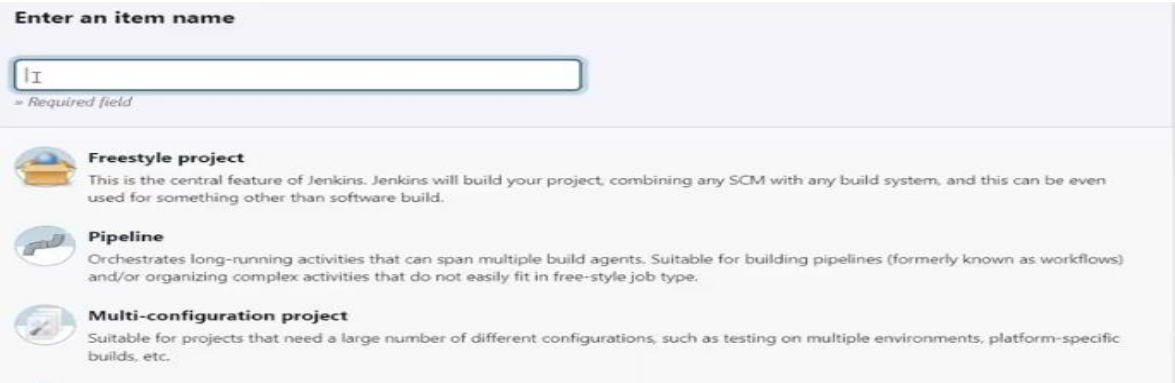
Create the username, password, your name and email address to login jenkins



Click on “New Item”



Click on “Pipeline” project



Go to Ubuntu linux to check whether the git is installed or not by using the below command

```
root@ip-172-31-11-179:~# git --version
git version 2.25.1
```

Git has already been installed on the latest Ubuntu linux OS.

To create the directory folder then initialize empty git repository

```
root@ip-172-31-11-179:~# mkdir dev
root@ip-172-31-11-179:~# cd dev
root@ip-172-31-11-179:~/dev# git init
Initialized empty Git repository in /root/dev/.git/
```

To create all five files in Linux, then copy all the code from Visual Studio Code, and then paste & save the code inside the file.

```
root@ip-172-31-11-179:~/dev# vi requirements.txt
root@ip-172-31-11-179:~/dev# vi main.py
root@ip-172-31-11-179:~/dev# vi form.html
root@ip-172-31-11-179:~/dev# vi Dockerfile
root@ip-172-31-11-179:~/dev# vi pod.yaml
```

Now we will move all the files from Ubuntu linux to GitHub remote repository by using the below commands

To move all the files from working directory to staging area by using "**git add .**" command. Dot refers to select all the files then to move all the files from staging area to local repository by using **"git commit"** command

```
root@ip-172-31-11-179:~/dev# git add .
root@ip-172-31-11-179:~/dev# git commit -m "First Commit"
```

```
root@ip-172-31-11-179:~/dev# git remote add origin https://github.com/
```
<Github repository name>

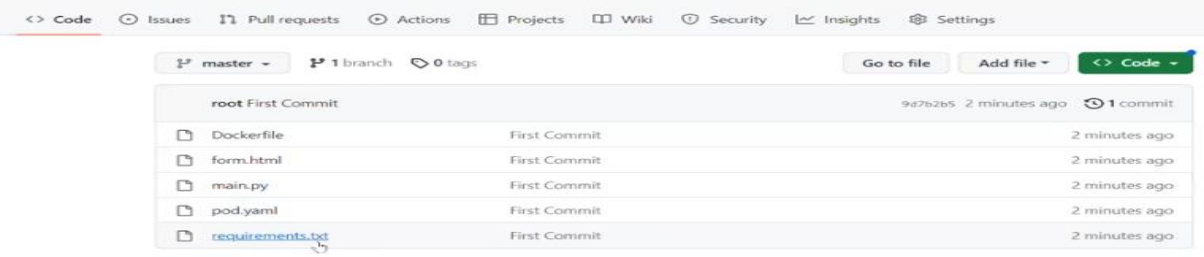
To move the files from the local repository to remote repository by using "**git push origin master**" command

```
root@ip-172-31-11-179:~/dev# git push origin master
```

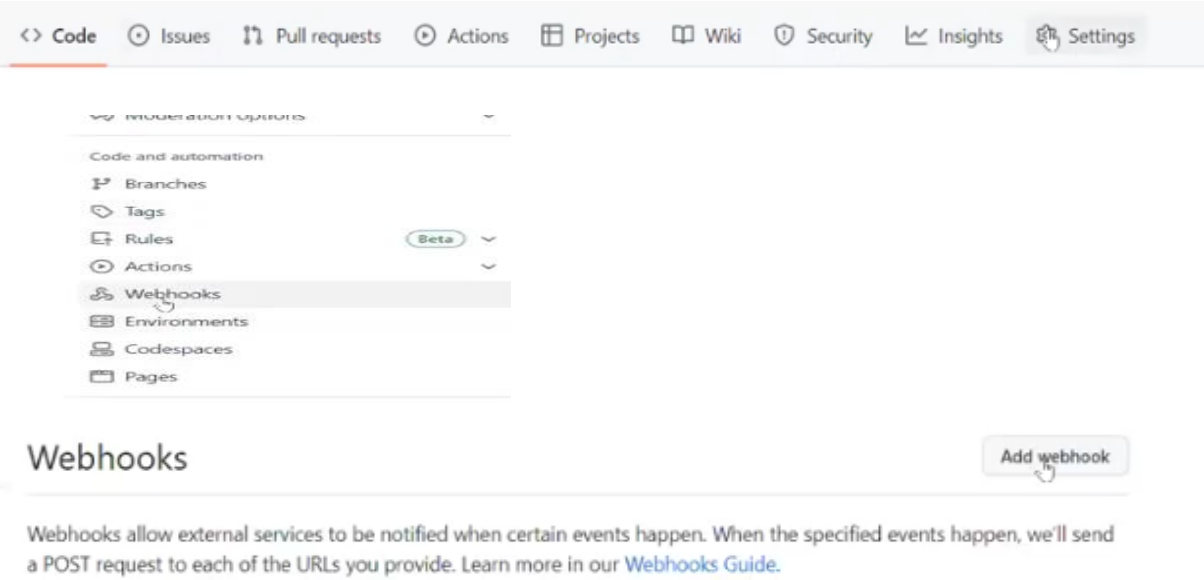
Enter the github username and password as access token

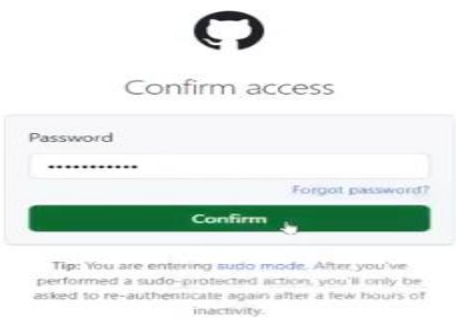
```
Username for
Password for
```

Now all the files has been successfully pushed to Github remote repository.

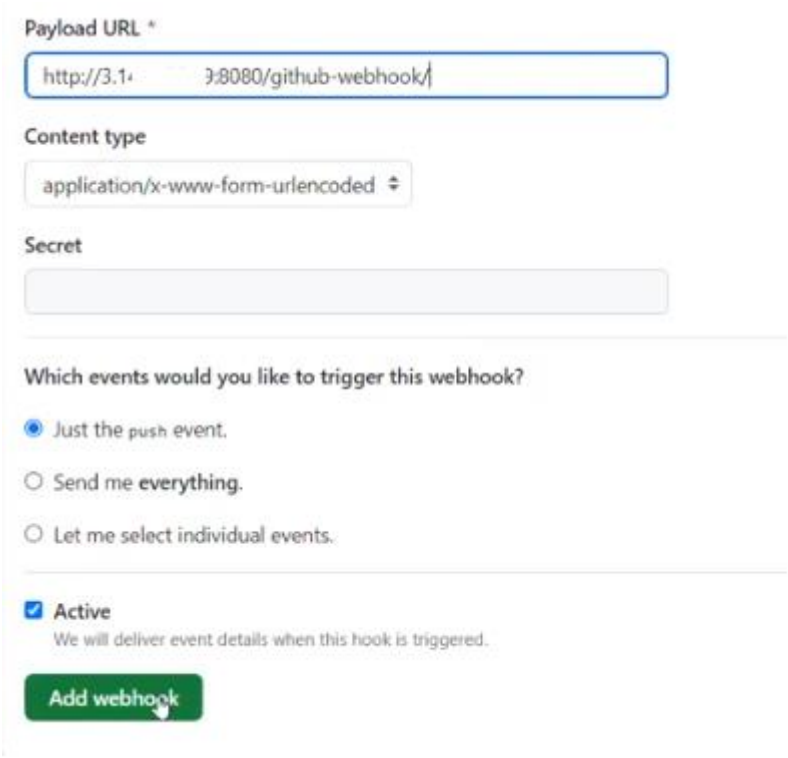


We will setup GitHub webhook. Go to GitHub settings -> Click on "Webhooks"

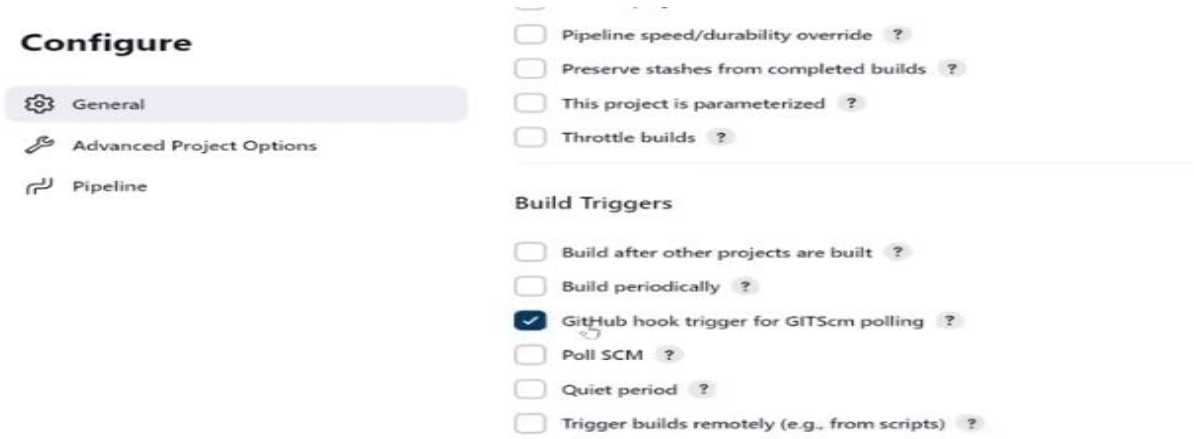




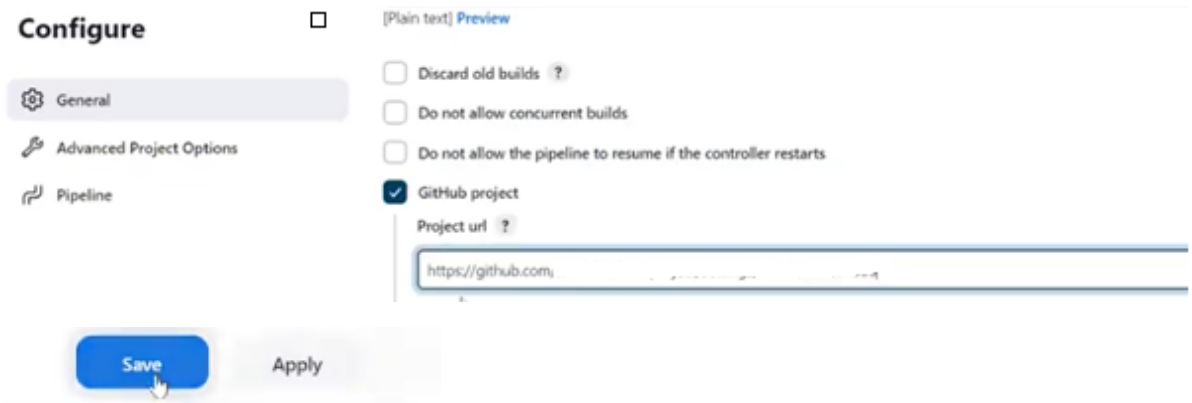
Give the Jenkins server URL in payload URL with adding **/github-webhook/**



Go to Jenkins website and Click on "General" and tick the check box of GitHub hook trigger for GITscm polling



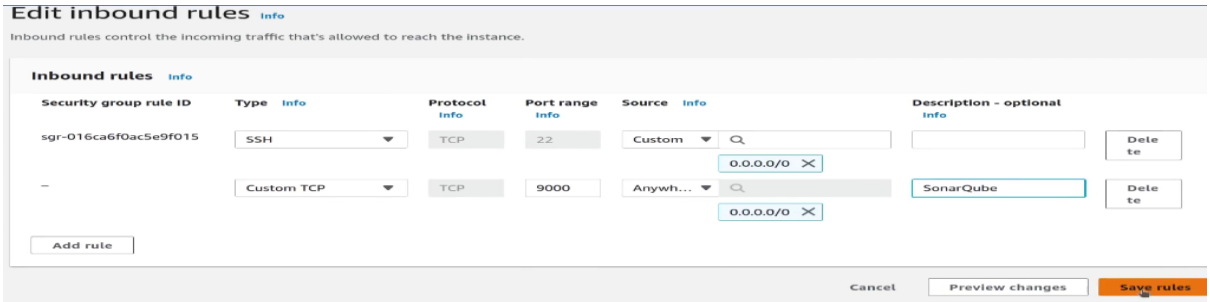
Give the github your project URL then click on "Save"



Github webhook has been successfully created

To create one virtual server for SonarQube on the AWS.

Allow the sonarQube port 9000 inbound security groups on the AWS virtual server.



We will install the sonarqube on Ubuntu linux by using wget command

```
wget https://binaries.sonarsource.com/Distribution/sonarqube/sonarqube-9.9.0.65466.zip
```

We will unzip the sonarqube by using the below command

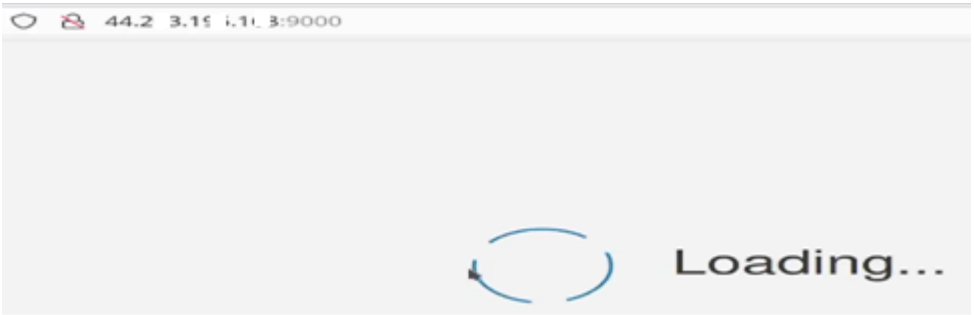
```
sudo apt install unzip
```

```
unzip sonarqube-9.9.0.65466.zip
```

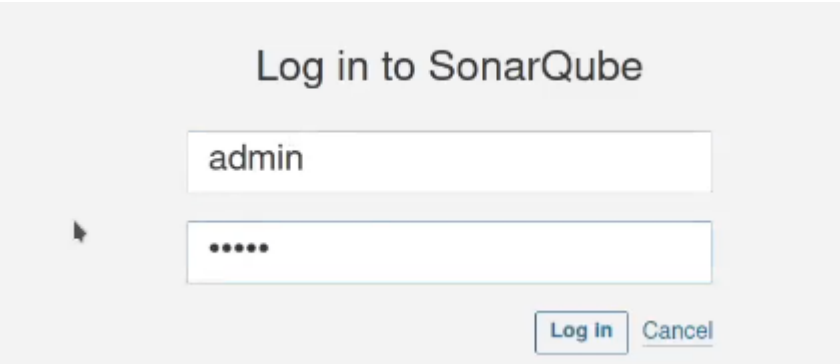
After Unzipped sonarqube, we are going to execute it

```
./sonar.sh
```

We will login to SonarQube by using “Virtual server IP address : sonarqube running port”

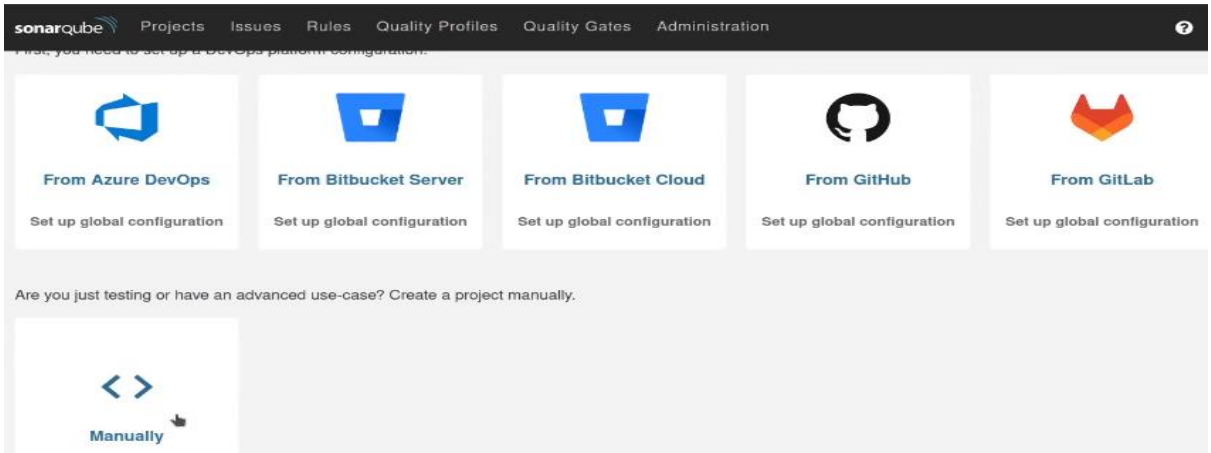


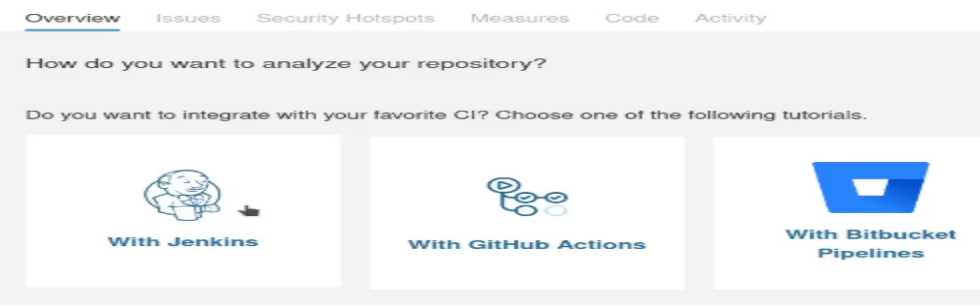
Now the sonarqube is working fine. Default username and password of sonarqube is admin.

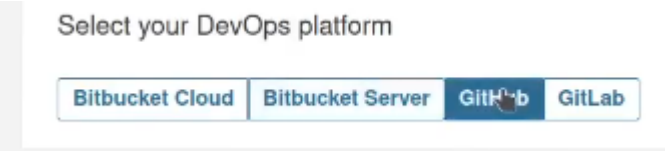


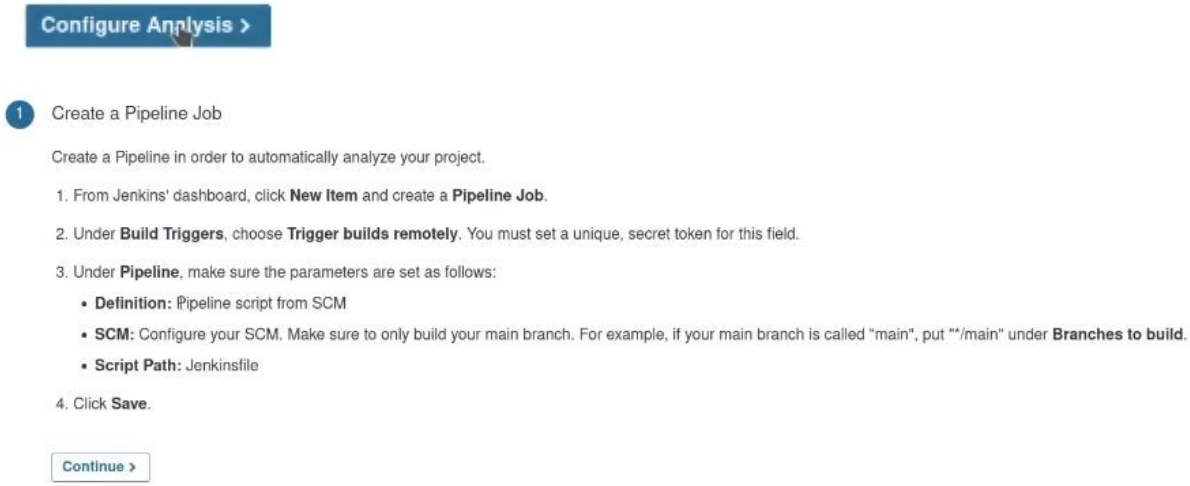
Change your password

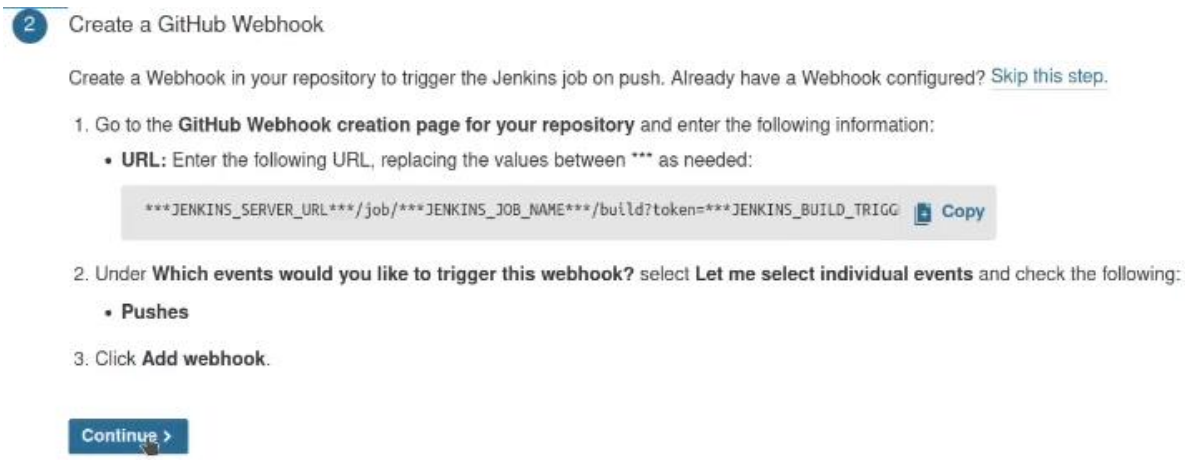
Click on “Manually “to create a project manually.







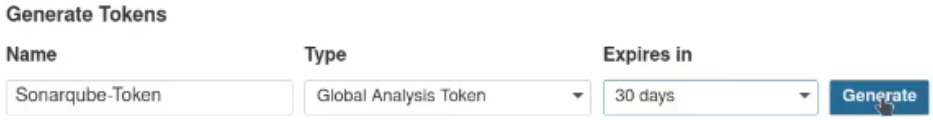




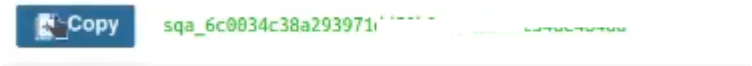
Copy the sonarqube project key. Once copied then click on “finish this tutorial”

Go to your sonarqube account and click on “security”

Now we have to create a token



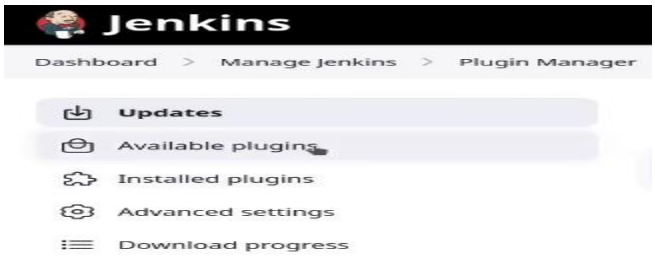
Copy this token



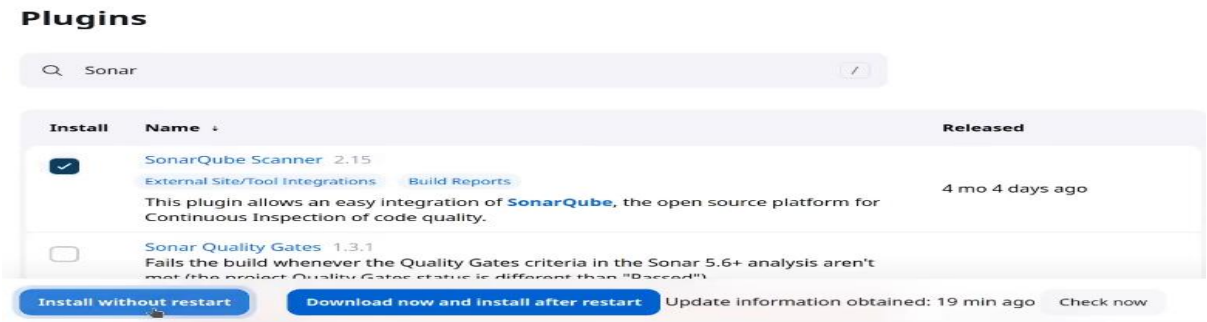
Now we go back to Jenkins to install plugins

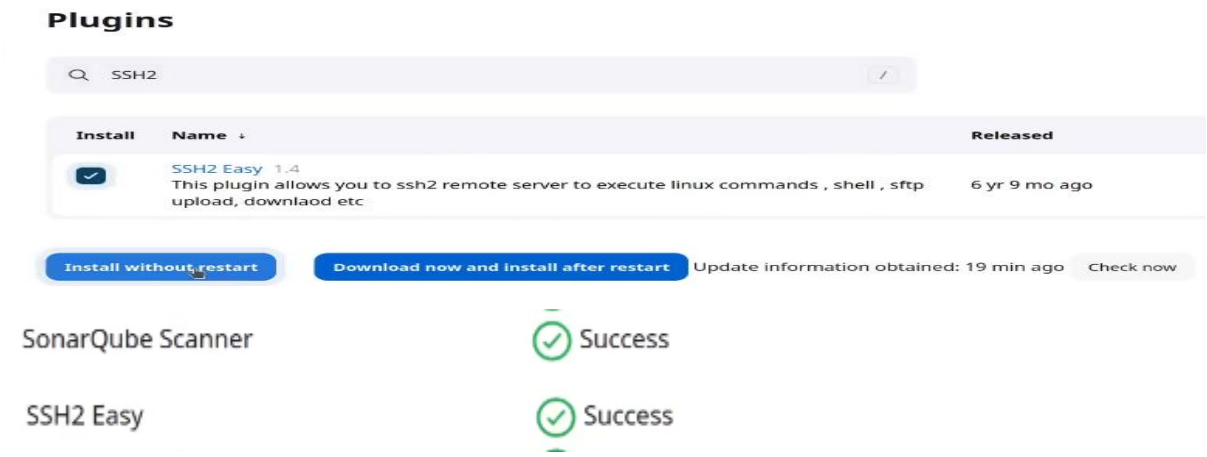
Click on “Manage Jenkins” & “Manage plugins”

Click on “Available Plugins”

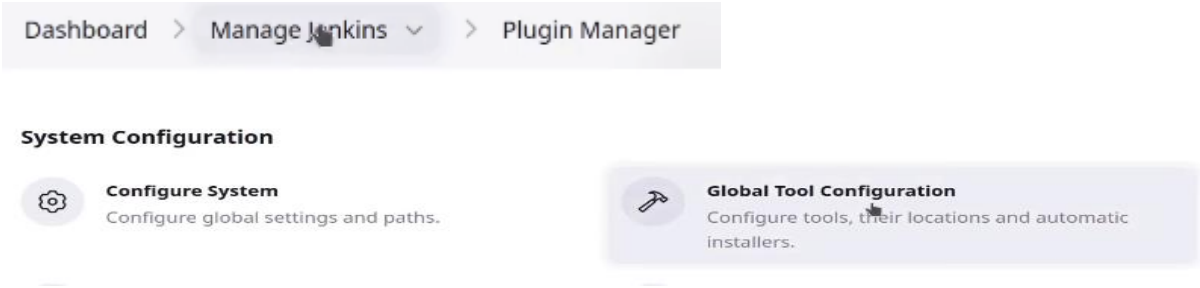


Search for SonarQube Scanner and SSH2

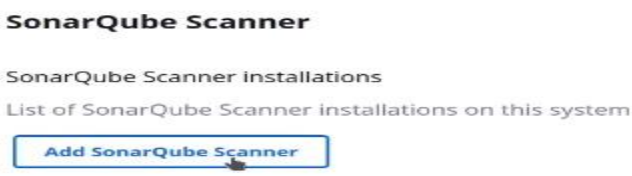




Click on “manage Jenkins” & “Global Tool Configuration”



Click on “Add SonarQube Scanner”



Give any name that you want

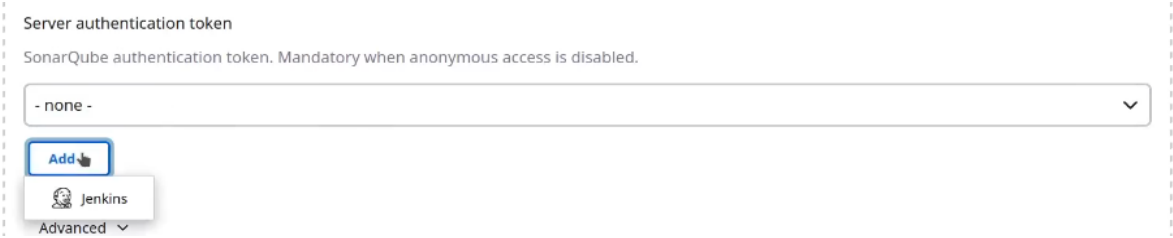
After saving it. We have to go to configure system

Scroll down to SonarQube servers & Click on “Add SonarQube”

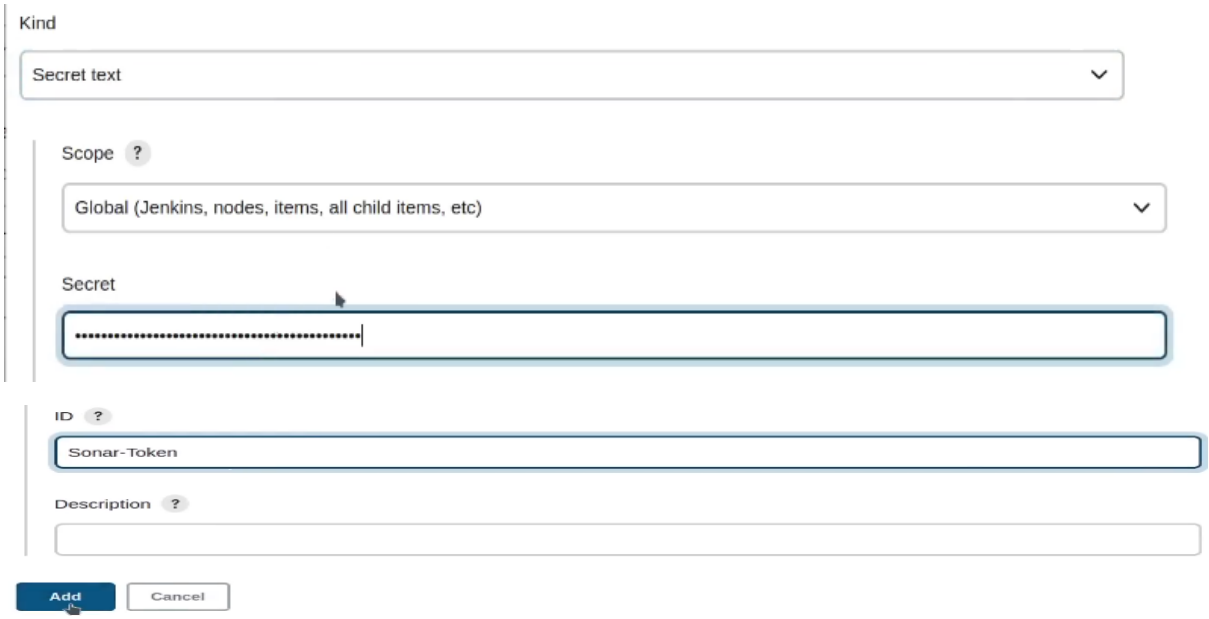
Give any name that you want

Give URL of the SonarQube

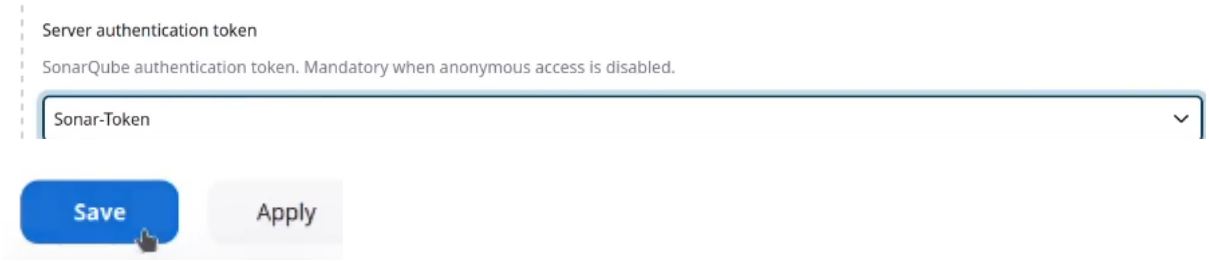
Click on “Add Jenkins”



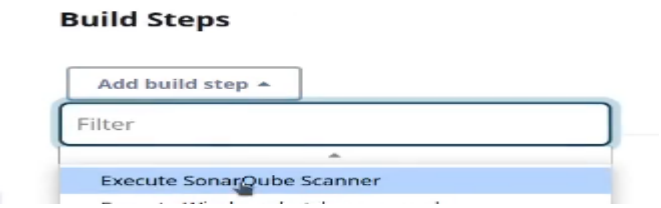
Select “secret text” and paste the token secret code (refer page no 12)



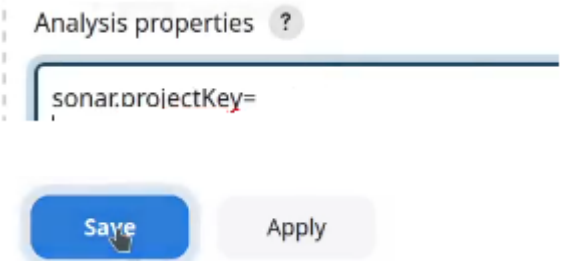
Select -> “Sonar Token” & “Save it”



Go to Build Steps and select “Execute SonarQube Scanner”

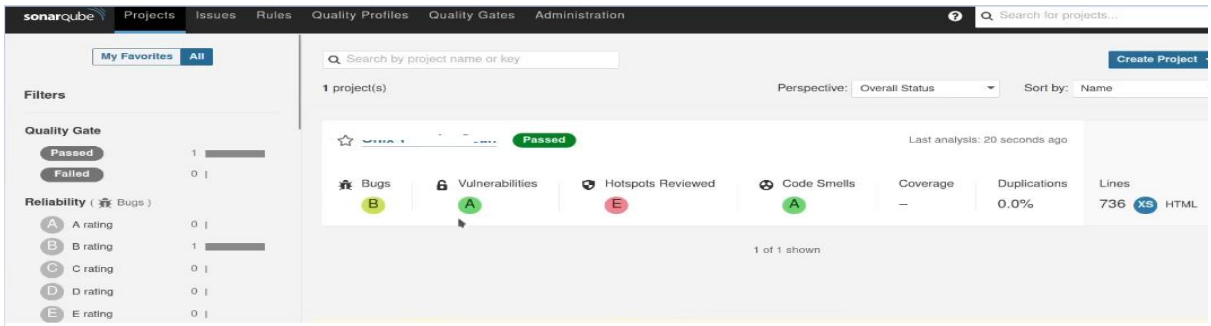


Paste the Sonar project key (Refer Page no 11) and save it.

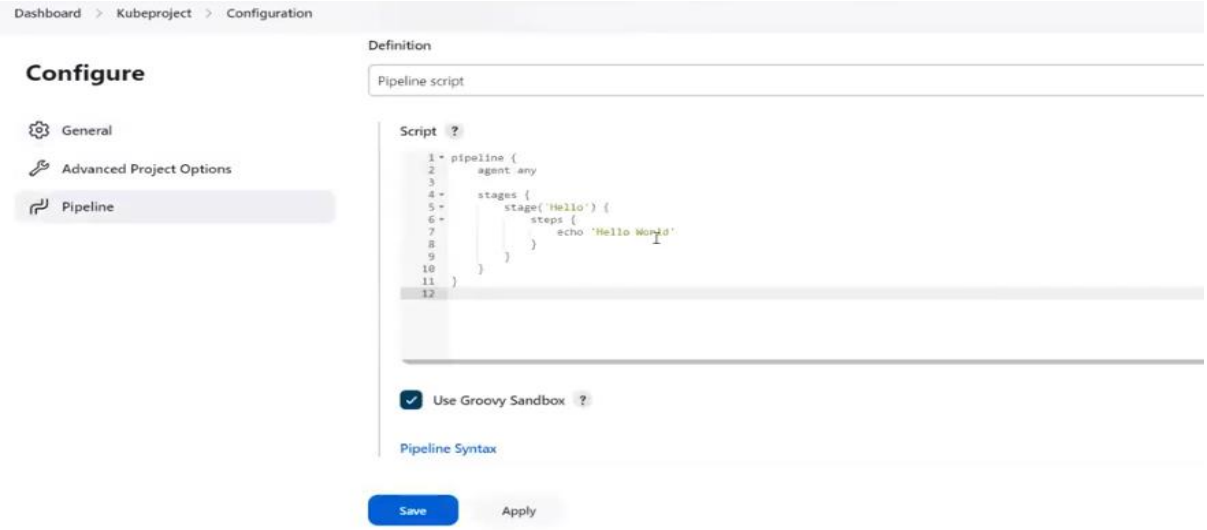


Click on “Build Now”

Now the Sonarqube scanning is working fine



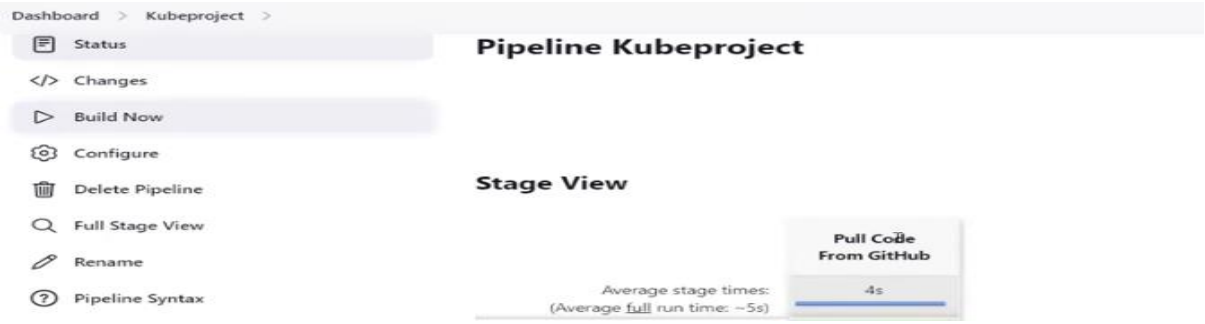
Now we have to write the four stages of declarative groovy pipeline scripts on the Jenkins.



Stage 1: Pull the code from GitHub

```
pipeline {
     agent any
     stages {
         stage('pull code from GitHub') {
             steps {
                 git <git hub remote repository URL>
              }
         }
```

Click on “Build Now”



Now the code has been successfully pulled from github

```
root@ip-172-31-11-179:~# cd /var/lib/jenkins
root@ip-172-31-11-179:/var/lib/jenkins# ls
config.xml                                  jenkins.telemetry.Correlator.xml                          secret.key
hudson.model.UpdateCenter.xml               jobs                                                      secret.key.not-so-secret
hudson.plugins.git.GitTool.xml              nodeMonitors.xml                                          secrets
identity.key.enc                            nodes                                                     updates
jenkins.install.InstallUtil.lastExecVersion org.jenkinsci.plugins.workflow.flow.FlowExecutionList.xml  userContent
jenkins.install.UpgradeWizard.state         plugins                                                   users
jenkins.model.JenkinsLocationConfiguration.xml queue.xml                                              workspace
root@ip-172-31-11-179:/var/lib/jenkins# cd workspace/
root@ip-172-31-11-179:/var/lib/jenkins/workspace# ls
Kubeproject
root@ip-172-31-11-179:/var/lib/jenkins/workspace# cd Kubeproject/
root@ip-172-31-11-179:/var/lib/jenkins/workspace/Kubeproject# ls
Dockerfile  form.html  main.py  pod.yaml  requirements.txt
```

We will install the Docker on Ubuntu linux

```
root@ip-172-31-11-179:~# apt install docker.io
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  bridge-utils containerd dns-root-data dnsmasq-base libidn11 pigz runc ubuntu-fan
Suggested packages:
  ifupdown aufs-tools cgroupfs-mount | cgroup-lite debootstrap docker-doc rinse zfs-fuse | zfsutils
The following NEW packages will be installed:
  bridge-utils containerd dns-root-data dnsmasq-base docker.io libidn11 pigz runc ubuntu-fan
0 upgraded, 9 newly installed, 0 to remove and 26 not upgraded.
Need to get 66.1 MB of archives.
After this operation, 293 MB of additional disk space will be used.
```

After the Docker installation, Login to Docker on linux.

```
root@ip-172-31-11-179:~# docker login
Login with your Docker ID to push and pull images from Docker Hub. If you don't have a Docker ID, head over to https://hub.docker.com t
o create one.

Username:
Password:

Login Succeeded
```

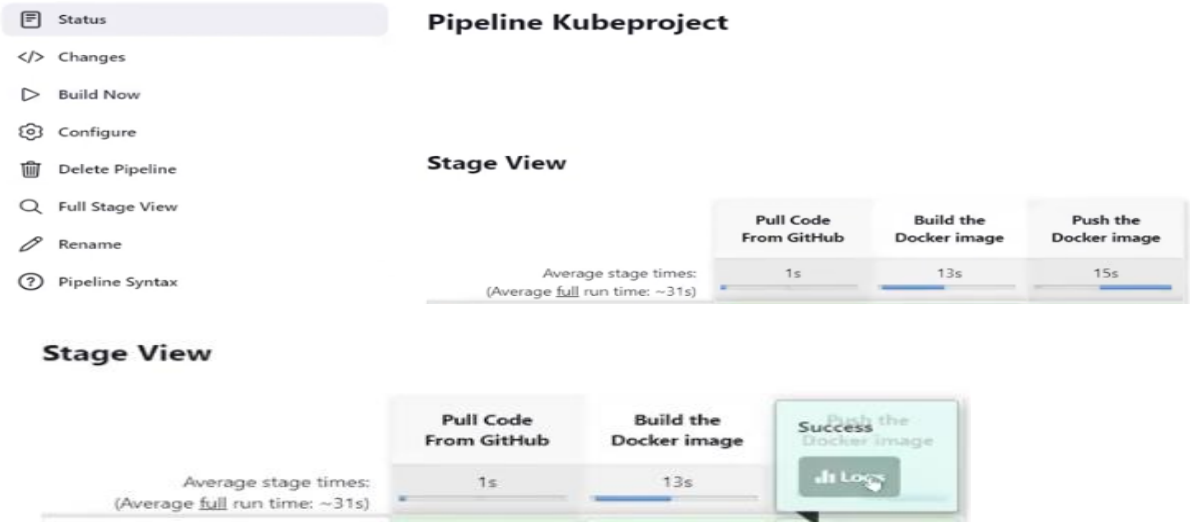
We will build the docker image from the docker file

Stage 2: Build the Docker image and run on Jenkins declarative pipeline script

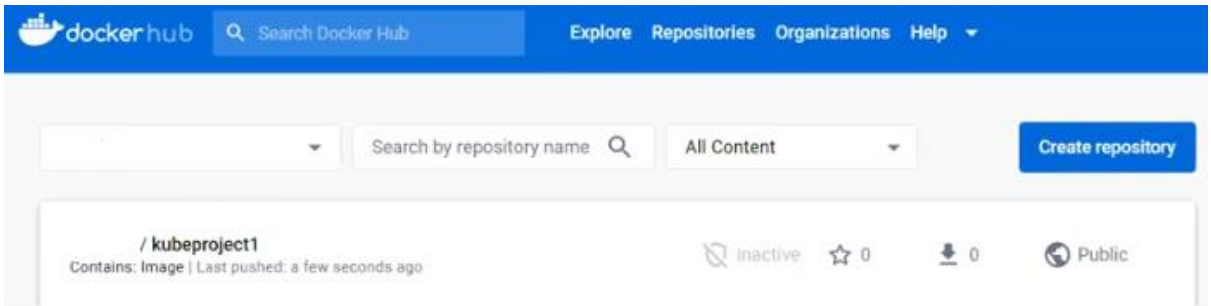
```
        stage('Build the Docker image')
            steps {
                sh 'sudo docker build -t <dockerhub username>/image name: latest /var/lib/jenkins/workspace/kubeprojects'
                sh 'sudo docker tag <dockerhub username>/image name: latest <dockerhub username>/image name:${BUILD_NUMBER}'
            }
        }
    }
}
```

Stage 3: Push the Docker image and run on Jenkins declarative pipeline script

```
        stage('push the Docker image'){
            steps {
                sh 'sudo docker image push <dockerhub username>/image name: latest
                sh 'sudo docker image push <dockerhub username>/image name:${BUILD_NUMBER}'
            }
        }
    }
}
```



Now the Docker image has been successfully pushed to docker hub



KOPS (Kubernetes Operations) install on Ubuntu linux

PREREQUIREMENTS

1. Kops binary (kubernetes cluster initiate)
2. Kubectl binary (kubernetes deployments)

KOPS BINARY SETUP

```
# curl -Lo kops https://github.com/kubernetes/kops/releases/download/$(curl
-s https://api.github.com/repos/kubernetes/kops/releases/latest | grep
tag_name | cut -d '"' -f 4)/kops-linux-amd64
# chmod +x ./kops
# sudo mv ./kops /usr/local/bin/
```

KUBECTL BINARY SETUP

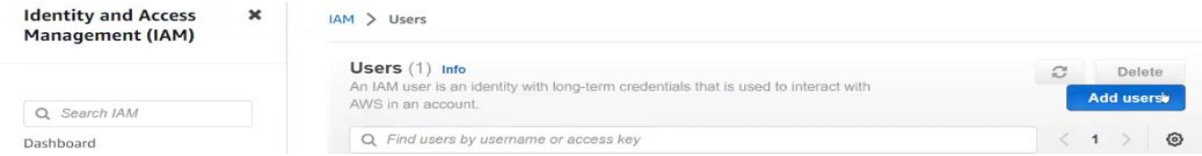
```
# curl -Lo kubectl https://storage.googleapis.com/kubernetes-
release/release/$(curl -s https://storage.googleapis.com/kubernetes-
release/release/stable.txt)/bin/linux/amd64/kubectl
# chmod +x ./kubectl
# sudo mv ./kubectl /usr/local/bin/kubectl
```

Kops and Kubectl binary setup has been successfully installed on Ubuntu linux by using the above commands

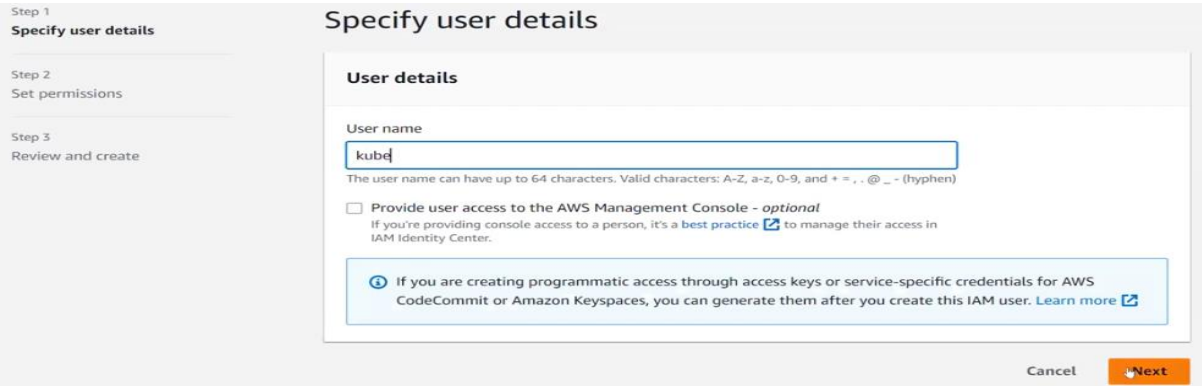
3. Setup IAM User (Kops access aws resource) -> kindly configure AWS Command line interface packages in your linux machines.

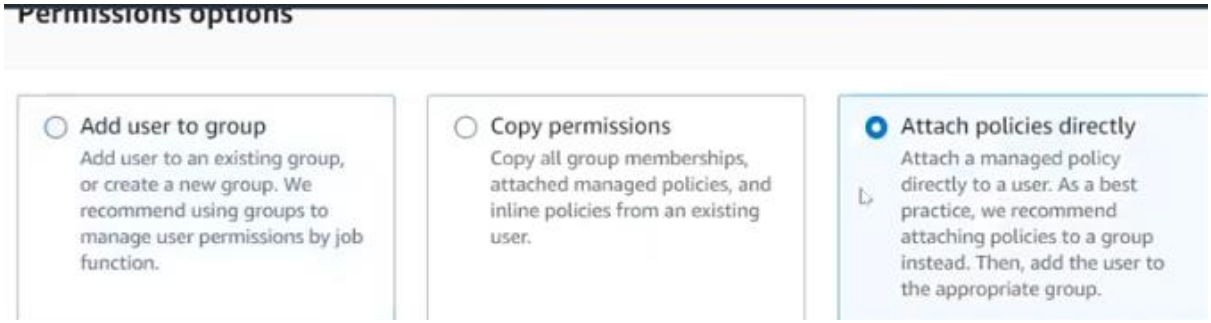
```
root@ip-172-31-11-179:~# apt install awscli
```

In order to build clusters within AWS, we will create a dedicated IAM user for kops. This user requires API credentials in order to use kops. Create the user and credentials using the AWS console.

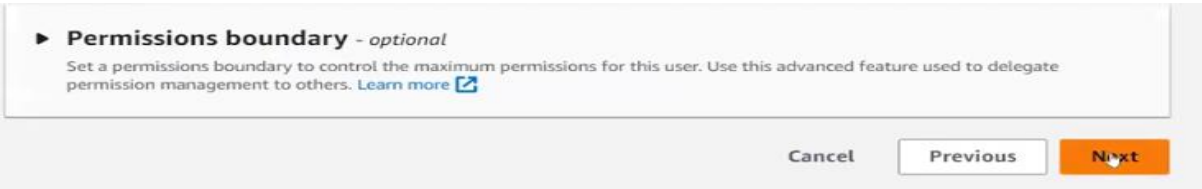


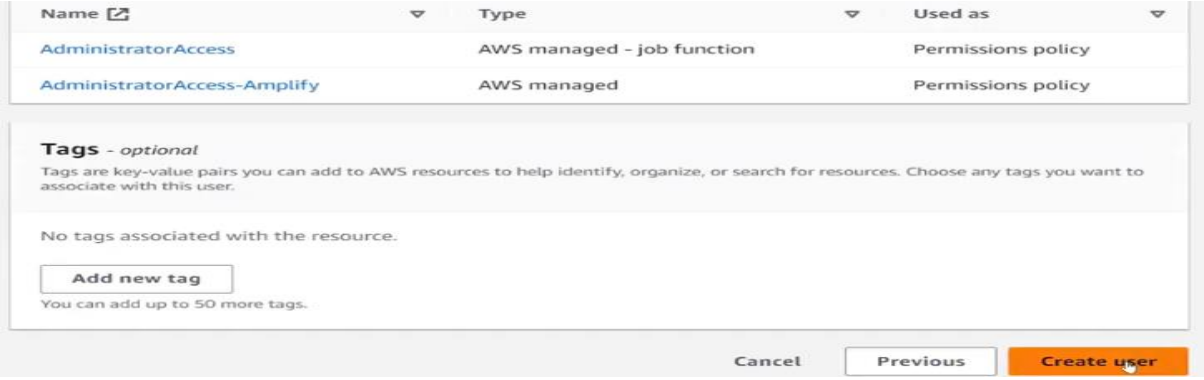
Set username



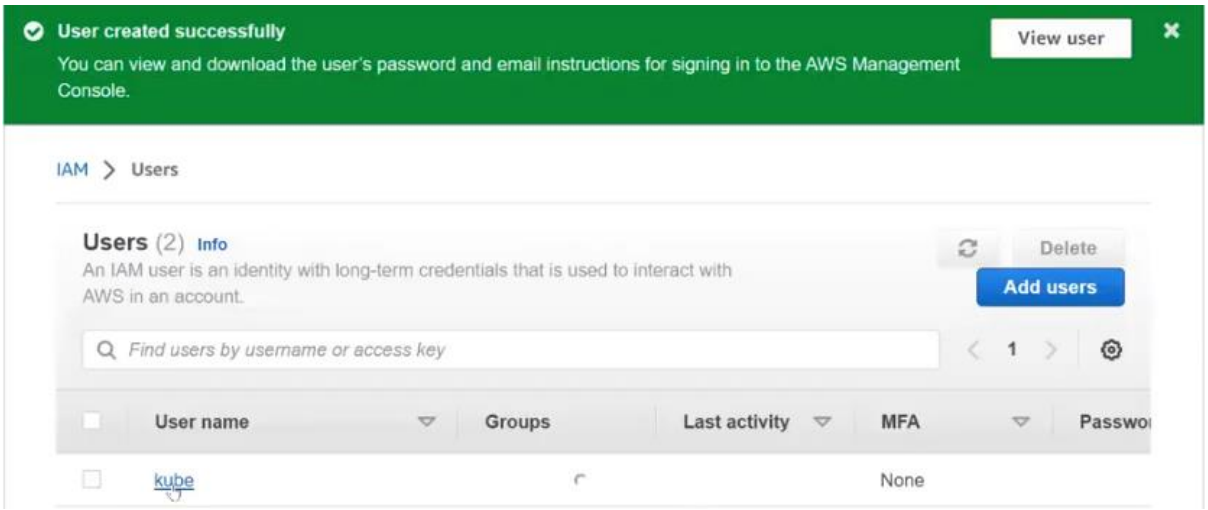


Policy name	Type	Attached entities
AccessAnalyzerServ...	AWS managed	0
AdministratorAccess	AWS managed - job function	1
AdministratorAcces...	AWS managed	1

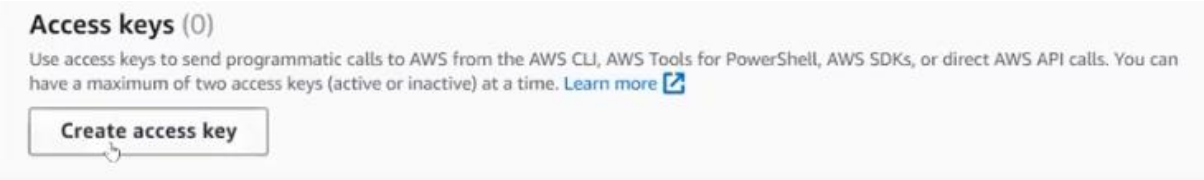


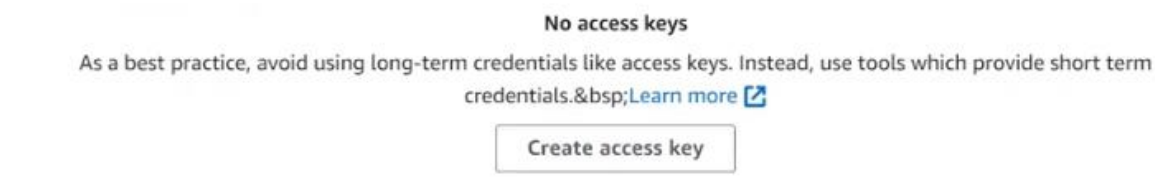


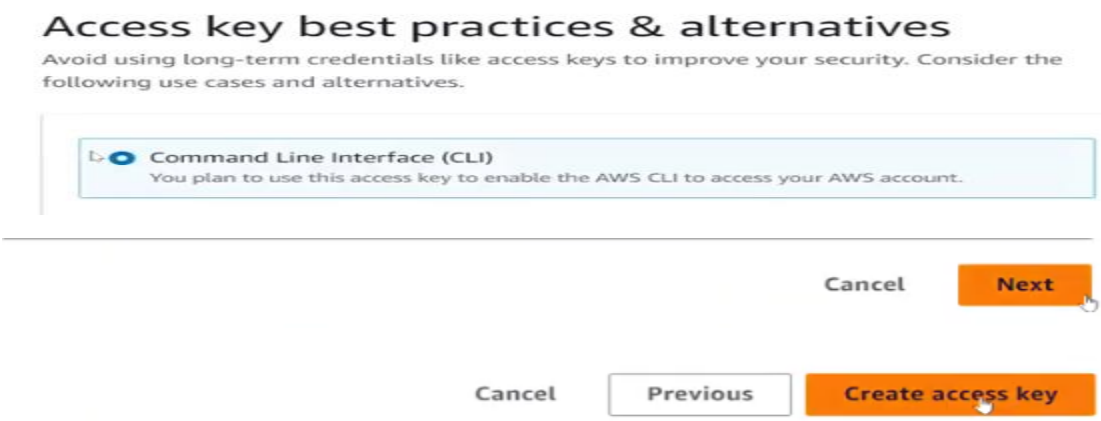
After username creation. Click on user name

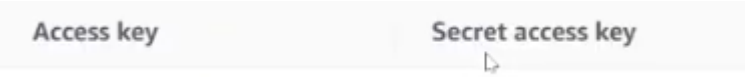


Click on create access key





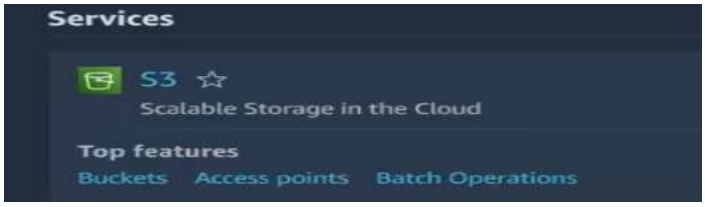


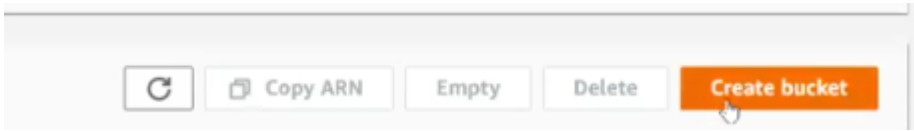


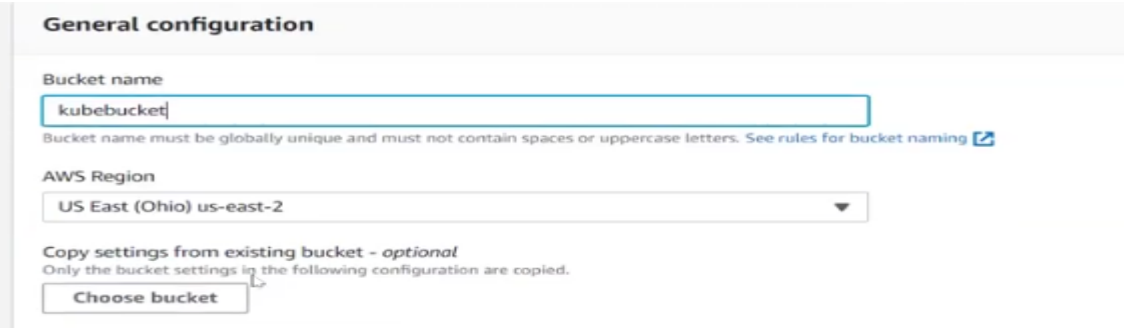
Copy the access key and secret access key and paste on Ubuntu linux

```
root@ip-172-31-11-179:~# aws configure

AWS Access Key ID [None]:

AWS Secret Access Key [None]:

Default region name [None]: us-east-2
Default output format [None]: json
```

To create one S3 bucket on AWS







Block Public Access settings for this bucket

Public access is granted to buckets and objects through access control lists (ACLs), bucket policies, access point policies, or all. In order to ensure that public access to this bucket and its objects is blocked, turn on Block all public access. These settings apply only to this bucket and its access points. AWS recommends that you turn on Block all public access, but before applying any of these settings, ensure that your applications will work correctly without public access. If you require some level of public access to this bucket or objects within, you can customize the individual settings below to suit your specific storage use cases. Learn more

- [ ] **Block *all* public access**
  Turning this setting on is the same as turning on all four settings below. Each of the following settings are independent of one another.
  - [ ] **Block public access to buckets and objects granted through *new* access control lists (ACLs)**
    S3 will block public access permissions applied to newly added buckets or objects, and prevent the creation of new public access ACLs for existing buckets and objects. This setting doesn't change any existing permissions that allow public access to S3 resources using ACLs.
  - [ ] **Block public access to buckets and objects granted through *any* access control lists (ACLs)**
    S3 will ignore all ACLs that grant public access to buckets and objects.
  - [ ] **Block public access to buckets and objects granted through *new* public bucket or access point policies**
    S3 will block new bucket and access point policies that grant public access to buckets and objects. This setting doesn't change any existing policies that allow public access to S3 resources.
  - [ ] **Block public and cross-account access to buckets and objects through *any* public bucket or access point policies**
    S3 will ignore public and cross-account access for buckets or access points with policies that grant public access to buckets and objects.

Bucket Versioning

Versioning is a means of keeping multiple variants of an object in the same bucket. You can use versioning to preserve, retrieve, and restore every version of every object stored in your Amazon S3 bucket. With versioning, you can easily recover from both unintended user actions and application failures. Learn more

Bucket Versioning

- ( ) Disable
- (•) Enable

[x] I acknowledge that the current settings might result in this bucket and the objects within becoming public.

Default encryption Info

Server-side encryption is automatically applied to new objects stored in this bucket.

Encryption key type Info

- (•) Amazon S3 managed keys (SSE-S3)
- ( ) AWS Key Management Service key (SSE-KMS)

Bucket Key

When KMS encryption is used to encrypt new objects in this bucket, the bucket key reduces encryption costs by lowering calls to AWS KMS. Learn more

- (•) Disable
- ( ) Enable

▸ **Advanced settings**

After creating the bucket you can upload files and folders to the bucket, and configure additional bucket settings.

Cancel | Create bucket

S3 bucket has been successfully created

```
root@ip-172-31-11-179:~# export KOPS_STATE_STORE=s3://<S3 Bucker name>
```

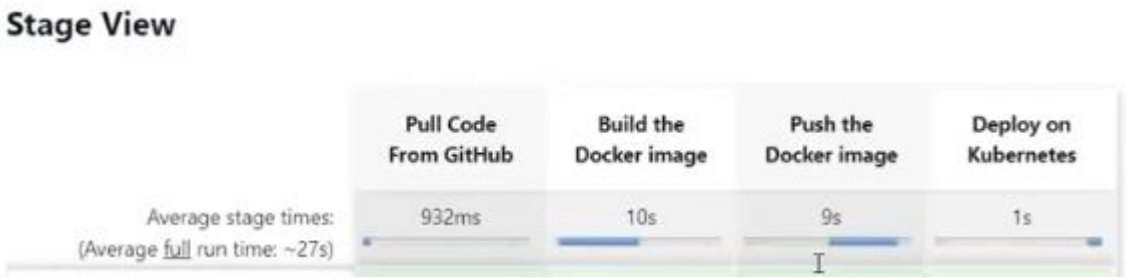
```
root@ip-172-31-11-179:~# kops create cluster --zones us-east-2a ${NAME}
```

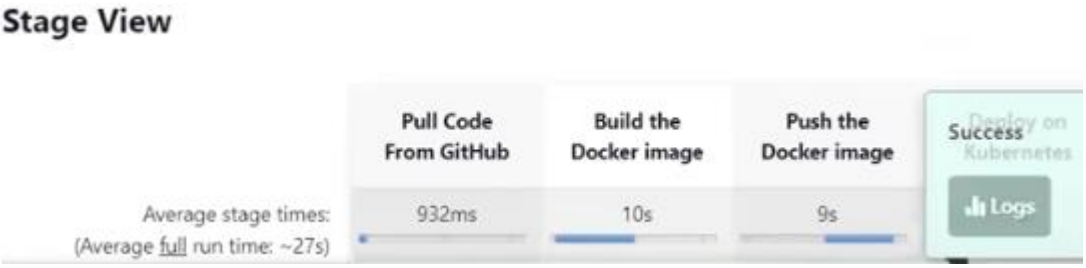
```
root@ip-172-31-11-179:~# kops update cluster --name

root@ip-172-31-11-179:~# kops validate cluster

Validating cluster
```

Now the cluster is ready.

Stage 4: Deploy on kubernetes

```
        stage('Deploy on kubernetes') {
            steps {
                sh 'kubectl apply -f /var/lib/jenkins/workspace/pod.yaml
                sh 'kubectl rollout restart deployment loadbalancer-pod'
            }
        }
    }
}
```

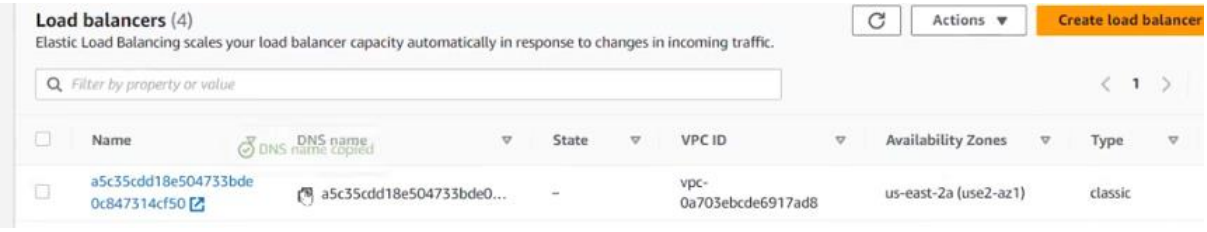




Four stages of Jenkins Declarative pipeline script:

```
pipeline {
    agent any
    stages {
        stage('pull code from GitHub') {
            steps {
                git <git hub remote repository URL>
             }
        }
        stage('Build the Docker image')
            steps {
                sh 'sudo docker build -t <dockerhub username>/image name: latest /var/lib/jenkins/workspace/kubeprojects'
                sh 'sudo docker tag <dockerhub username>/image name: latest <dockerhub username>/image name:${BUILD_NUMBER}'
            }
        }
        stage('push the Docker image') {
            steps {
                sh 'sudo docker image push <dockerhub username>/image name: latest
                sh 'sudo docker image push <dockerhub username>/image name:${BUILD_NUMBER}'
            }
        }
        stage('Deploy on kubernetes') {
            steps {
                sh 'kubectl apply -f /var/lib/jenkins/workspace/pod.yaml
                sh 'kubectl rollout restart deployment loadbalancer-pod'
            }
        }
    }
}
```

```
root@ip-172-31-11-179:~# kubectl get svc
NAME                TYPE           CLUSTER-IP     EXTERNAL-IP    PORT(S)        AGE
kubernetes          ClusterIP      100.64.0.1     <none>         443/TCP        10m
loadbalancer-svc    LoadBalancer   100.64.208.9   <pending>      80:30421/TCP   63s
root@ip-172-31-11-179:~# kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
loadbalancer-pod-c68658d-6kx4b      1/1     Running   0          4m32s
loadbalancer-pod-c68658d-7b46v      1/1     Running   0          4m51s
loadbalancer-pod-c68658d-wg2hx      1/1     Running   0          4m30s
```



Now webpage has been successfully triggered