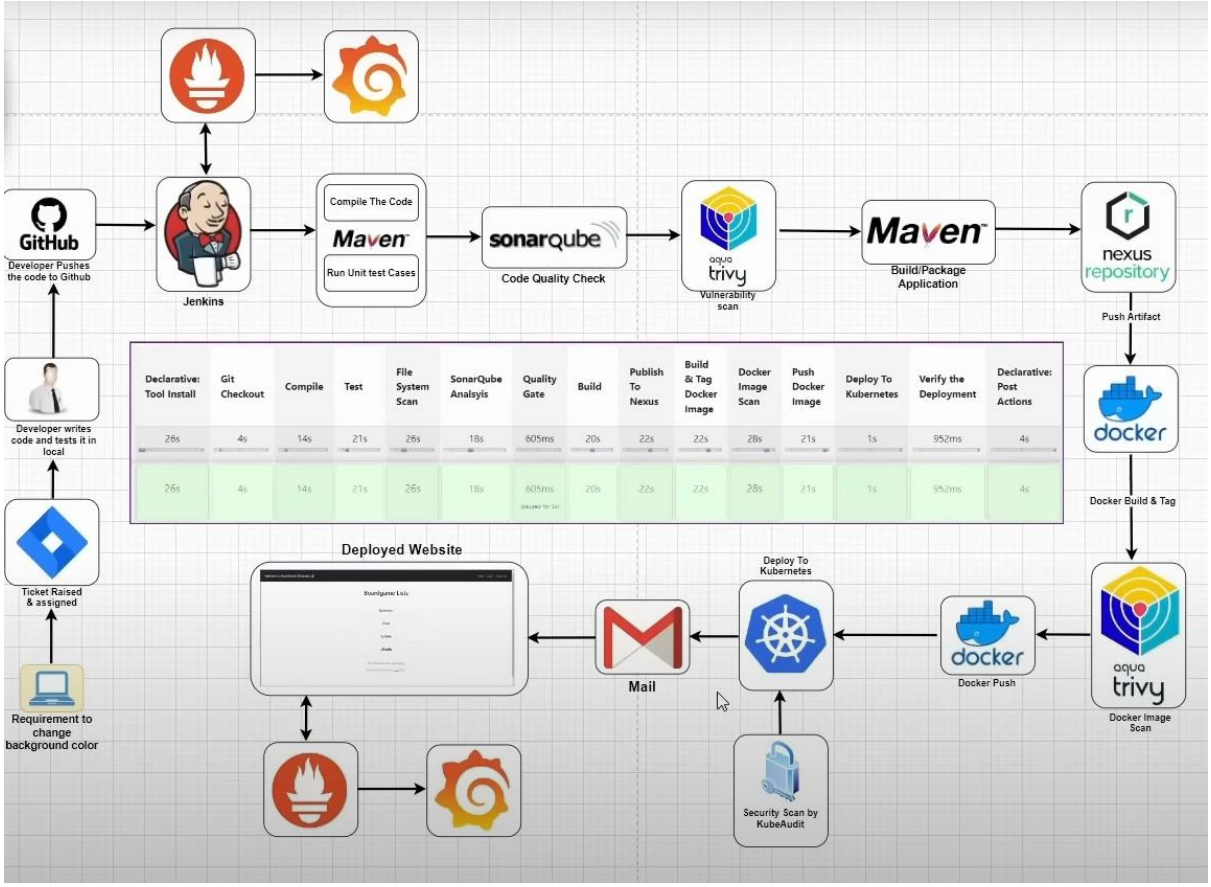
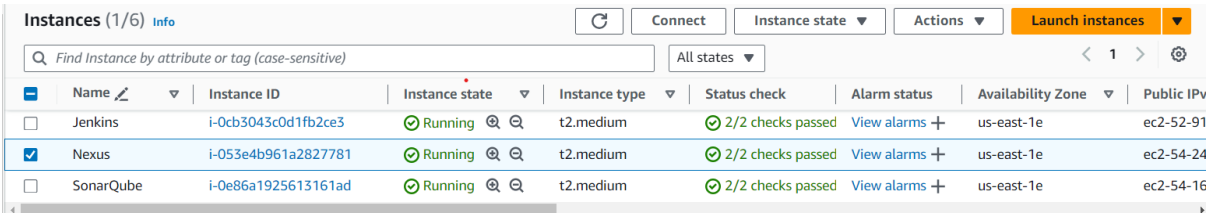
The Ultimate End-End CI/CD Project



Phase-1

Create 3 EC2 Instances with 30GB RAM and choose t2.medium



Instances (1/6) Info | Connect | Instance state | Actions | Launch instances

Find Instance by attribute or tag (case-sensitive) | All states

Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv
Jenkins	i-0cb3043c0d1fb2ce3	Running	t2.medium	2/2 checks passed	View alarms +	us-east-1e	ec2-52-91
Nexus	i-053e4b961a2827781	Running	t2.medium	2/2 checks passed	View alarms +	us-east-1e	ec2-54-24
SonarQube	i-0e86a1925613161ad	Running	t2.medium	2/2 checks passed	View alarms +	us-east-1e	ec2-54-16

Install Docker on All 3 VMs

Step-by-Step Installation

1. Install prerequisite packages:

sudo apt-get update

sudo apt-get install ca-certificates curl

2. Download and add Docker's official GPG key:

sudo install -m 0755 -d /etc/apt/keyrings

sudo curl -fsSL https://download.docker.com/linux/ubuntu/gpg -o /etc/apt/keyrings/docker.asc

sudo chmod a+r /etc/apt/keyrings/docker.asc

3. Add Docker repository to Apt sources:

echo "deb [arch=$(dpkg --print-architecture) signed by=/etc/apt/keyrings/docker.asc] https://download.docker.com/linux/ubuntu $(. /etc/os-release && echo "$VERSION_CODENAME") stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null

4. Update package index:

sudo apt-get update

5. Install Docker packages:

sudo apt-get install docker-ce docker-ce-cli containerd.io -y

6. Grant permission to Docker socket (optional, for convenience):

sudo chmod 666 /var/run/docker.sock

By following these steps, you should have successfully installed Docker on your Ubuntu system. You can now start using Docker to containerize and manage your applications.

Follow this official document if you find any errors:

Link: [Install Docker Engine on Ubuntu | Docker Docs]

Setting Up Jenkins on Ubuntu

Step-by-Step Installation

1. Update the system:

sudo apt-get update

sudo apt-get upgrade -y

2. Install Java (Jenkins requires Java):

sudo apt install -y fontconfig openjdk-17-jre

3. Add Jenkins repository key:

sudo wget -O /usr/share/keyrings/jenkins-keyring.asc

https://pkg.jenkins.io/debian-stable/jenkins.io-2023.key

4. Add Jenkins repository:

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

5. Update the package index:

sudo apt-get update

6. Install Jenkins:

sudo apt-get install -y jenkins

7. Start and enable Jenkins:

sudo systemctl start jenkins

sudo systemctl enable jenkins

8. Access Jenkins:

- Open a web browser and go to http://your_server_ip_or_domain:8080.
- You will see a page asking for the initial admin password. Retrieve it using:
- sudo cat /var/lib/jenkins/secrets/initialAdminPassword
- Enter the password, install suggested plugins, and create your first admin user.

or follow this official document

link: https://www.jenkins.io/doc/book/installing/linux/#debianubuntu

Installing Trivy on Jenkins Server

Step-by-Step Installation

1. Install prerequisite packages:

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

2. Add Trivy repository key:

wget -qO - https://aquasecurity.github.io/trivy-repo/deb/public.key |

sudo apt-key add -

3. Add Trivy repository to sources:

echo deb https://aquasecurity.github.io/trivy-repo/deb $(lsb_release -sc) main | sudo tee -a /etc/apt/sources.list.d/trivy.list

4. Update package index:

sudo apt-get update

5. Install Trivy:

sudo apt-get install trivy

or follow this official document

link: https://aquasecurity.github.io/trivy/v0.18.3/installation/

EKS-Setup:

First Create a user in AWS IAM with any name

Attach Policies to the newly created user

below policies

AmazonEC2FullAccess

AmazonEKS_CNI_Policy

AmazonEKSClusterPolicy

AmazonEKSWorkerNodePolicy

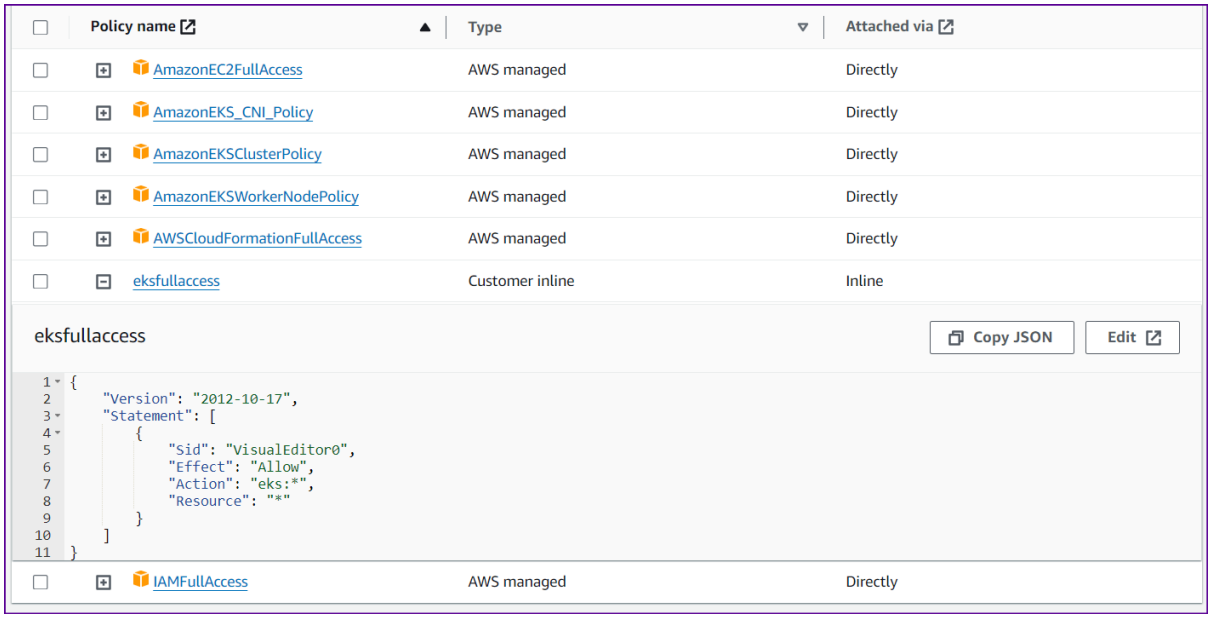
AWSCloudFormationFullAccess

IAMFullAccess

One more policy we need to create with content as below

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": "eks:*",
            "Resource": "*"
        }
    ]
}
```

Attach this policy to your user as well



AWSCLI

```
curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o
"awscliv2.zip"
sudo apt install unzip
unzip awscliv2.zip
sudo ./aws/install
aws configure
```

KUBECTL

```
curl -o kubectl https://amazon-eks.s3.us-west-2.amazonaws.com/1.19.6/2021-
01-05/bin/linux/amd64/kubectl
chmod +x ./kubectl
sudo mv ./kubectl /usr/local/bin
kubectl version --short --client
```

EKSCTL

```
curl --silent --location
"https://github.com/weaveworks/eksctl/releases/latest/download/eksctl_$(una
me -s)_amd64.tar.gz" | tar xz -C /tmp
sudo mv /tmp/eksctl /usr/local/bin
eksctl version
```

Create EKS CLUSTER

```
eksctl create cluster --name=my-eks7 \
                      --region=ap-south-1 \
                      --zones= ap-south-1a, ap-south-1b \
                      --version=1.30 \
                      --without-nodegroup
```

```
eksctl utils associate-iam-oidc-provider \
    --region us-east-1 \
    --cluster my-eks2 \
    --approve

eksctl create nodegroup --cluster=my-eks7 \
                        --region= ap-south-1\
                        --name=node2 \
                        --node-type=t3.medium \
                        --nodes=3 \
                        --nodes-min=2 \
                        --nodes-max=4 \
                        --node-volume-size=20 \
                        --ssh-access \
                        --ssh-public-key=panduaws \
                        --managed \
                        --asg-access \
                        --external-dns-access \
                        --full-ecr-access \
                        --appmesh-access \
                        --alb-ingress-access
```

`Note: --ssh-public-key=panduaws → Give pem file name in AWS`

- Open INBOUND TRAFFIC IN ADDITIONAL Security Group

SonarQube Setup:

Ssh into sonarqube ec2 instance

docker run -d –name sonar -p 9000:9000 sonarqube:lts-comminity

access using <publicip:9000>

username: admin

password:admin

Nexus Setup:

Ssh into nexus ec2 instance

docker run -d –name nexus -p 8081:8081 sonatype/nexus3

access using <publicip:8081>

sign to nexus using the password, the password is stored in '**/nexus-data/admin.password**'

enter into the container using

docker exec -it <container-ID> /bin/bash

then run

cat **/nexus-data/admin.password**

you will get password

```
ubuntu@ip-172-31-59-116:~$ docker ps
CONTAINER ID   IMAGE             COMMAND                  CREATED
AMES
5559dd04aad5   sonatype/nexus3   "/opt/sonatype/nexus…"   About a minute ago
exus
ubuntu@ip-172-31-59-116:~$ docker exec -it 5559 /bin/bash
bash-4.4$ cat /nexus-data/admin.password
66db8137-b229-4682-a789-10655502bd3bbash-4.4$
```

Username: admin

Password: 66db8137-b229-4682-a789-10655502bd3b ###Replace your password

Phase-2

Close the repository and create your own repository and push those into your github repository

1.clone the repo:

git clone https://github.com/Madeep9347/cicd-project7.git

2. change the remote repo

git remote set-url origin https://github.com/Madeep9347/cicd-project7.git

→ replace with your github repo

git remote add new-origin https://github.com/Madeep9347/cicd-project7.git\

→replace with your github repo

3. Initialize Git Repository

git init

4. Add Files to Git:

Stage all files for the first commit:

git add .

5. Commit Files:

Commit the staged files with a commit message:

git commit -m "Initial commit"

6. Push to GitHub:

Push the local repository to GitHub:

git push -u origin main

Install Plugins in Jenkins

1. Eclipse Temurin installer → for jdk
2. Sonarqube scanner
3. Docker
4. Docker pipeline
5. Kubernetes
6. Kubernetes cli
7. Kubernetes credentials
8. Kubernetes clint api
9. Config file provider → for Nexus
10. Maven integration
11. Pipeline maven integration

Now we installed the tools and Now we need to configure them

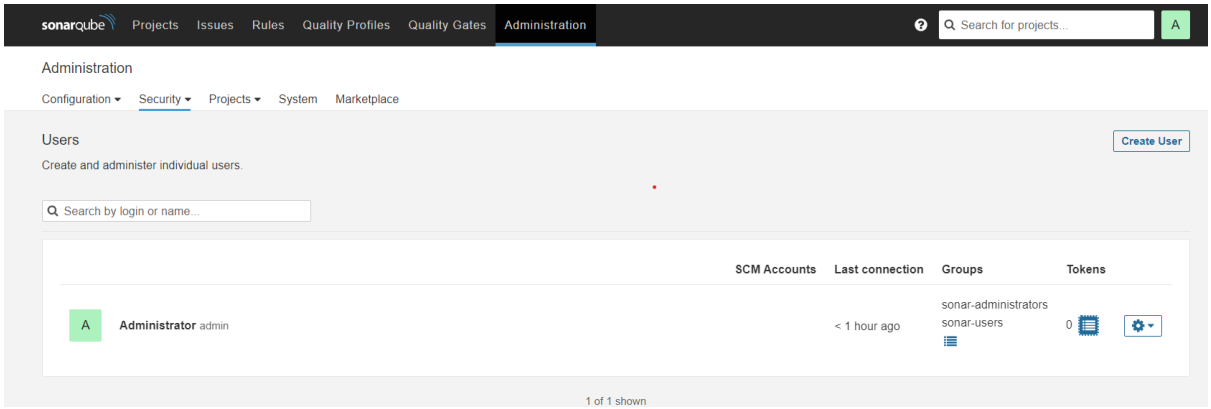
Go to → manage Jenkins→Tools→

1. Jdk→ name= jdk17 , install automatically from adoptium.net, version= jdk17 latest
2. Sonarqube scanner → name=sonar-scanner, Install automatically
3. Maven → name= maven3, version= 3.6.3
4. Docker→ name=docker, install automatically from docker.com

Now configure the sonarqube server in Jenkins

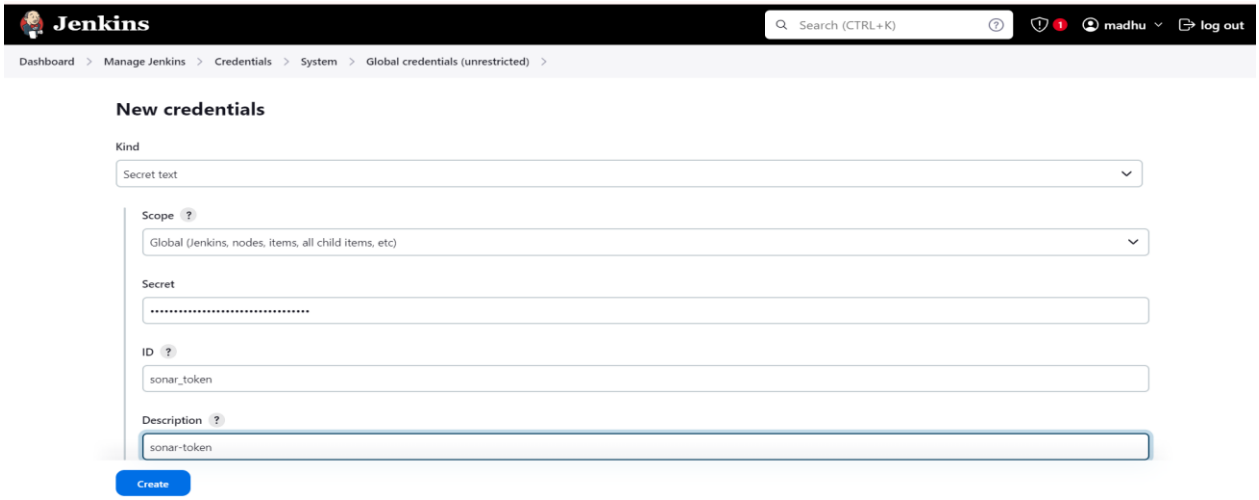
Firstly generate the token in sonarqube

Goto → Administaration→ security→ users→update token→ name= sonartoken and generate



add the token in jenkins

goto→ manage jenkins→credentials→ global→ kind= secret text→secret=<your-token>id=sonar-token, description=sonar=token



Go to → manage Jenkins→ system→sonarqube server→name=sonar, url=http://publicip:9000, token=sonar-token

Sonarqube scanner→ This is the tool that actually scans your code and sends the results to the SonarQube server.

Sonarqube server→ Displays analysis results.

Nexus Configuration:

Update your pom.xml file with your nexus repositories

```
</dependency>
        </dependencies>

        <distributionManagement>
        <repository>
            <id>maven-releases</id>
            <url>http://54.242.176.54:8081/repository/maven-releases/</url>
        </repository>
        <snapshotRepository>
            <id>maven-snapshots</id>
            <url>http://54.242.176.54:8081/repository/maven-snapshots/</url>
        </snapshotRepository>
    </distributionManagement>
```

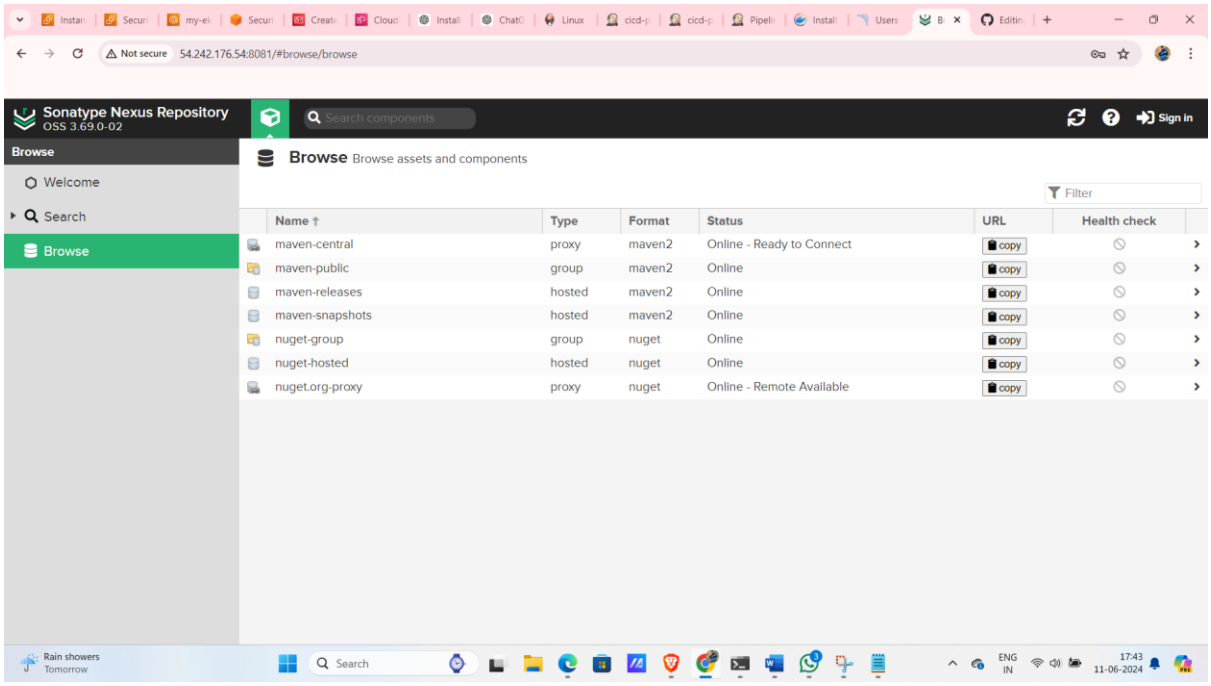
Copy the maven-releases URL , maven-snapshots URL and update in the pom.xml file

“”

<url>http://54.242.176.54:8081/repository/maven-releases/</url>

<url>http://54.242.176.54:8081/repository/maven-snapshots/</url>

“”

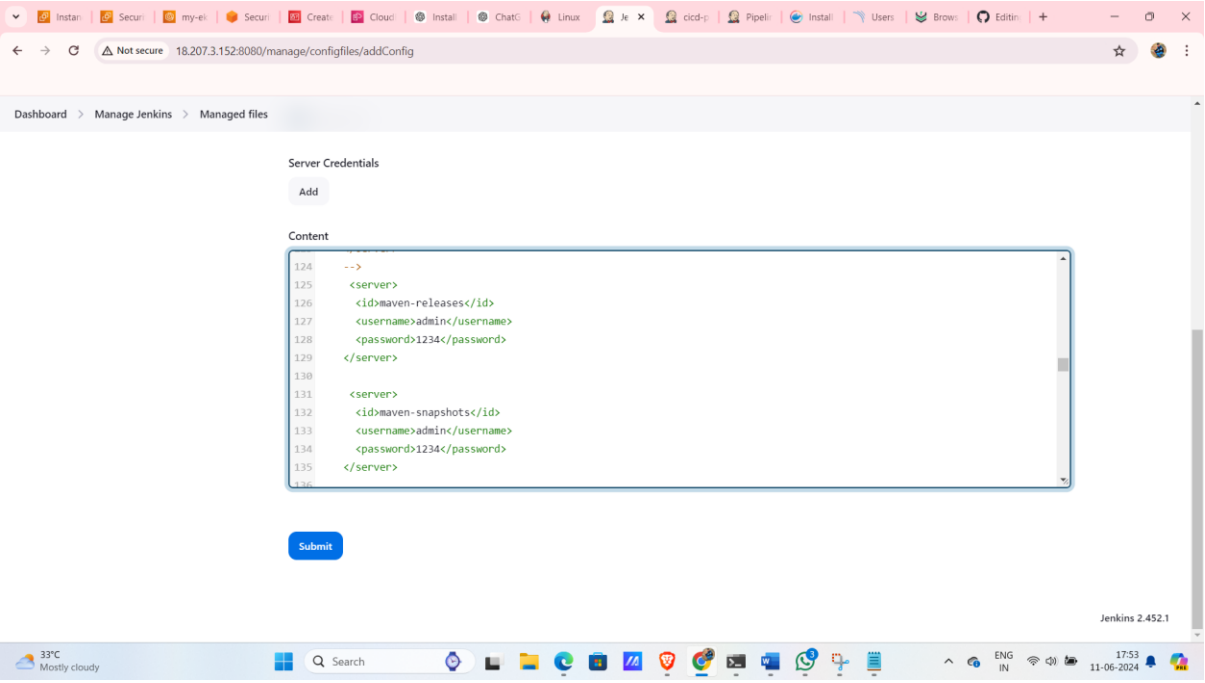


Nexus authentication with Jenkins:

Go to→ manage Jenkins→manage files→add new config→ select global mavensettings.xml, id=maven-setting → click on next

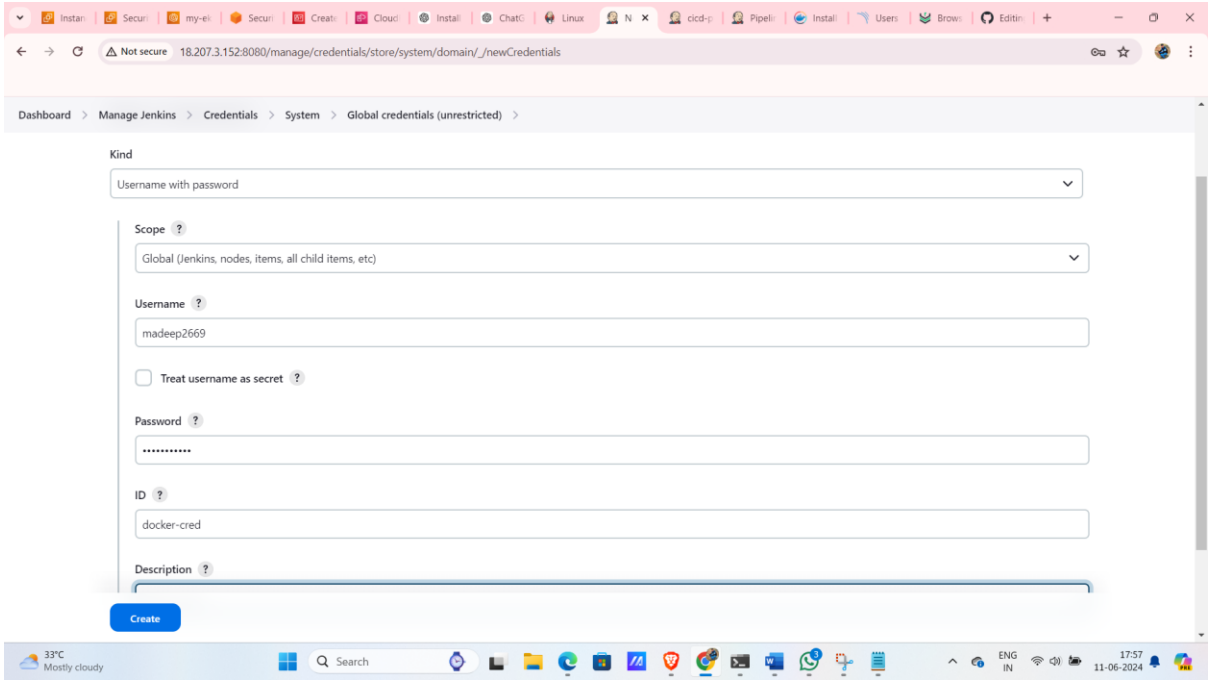
Go to content

Add the servers with name, username and password



Add DockerHub credentials in Jenkins:

Goto→ manage Jenkins→credentials→kind=username and password



Create Service Account, Role & Assign that role, And create a secret for Service Account and generate a Token in Jenkins server

Creating Service Account

First create the namespace using

Kubectl create namespace webapps

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: jenkins
  namespace: webapps
```

Create Role

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: app-role
  namespace: webapps
rules:
  - apiGroups:
```

```
        - ""
        - apps
        - autoscaling
        - batch
        - extensions
        - policy
        - rbac.authorization.k8s.io
    resources:
      - pods
      - secrets
      - componentstatuses
      - configmaps
      - daemonsets
      - deployments
      - events
      - endpoints
      - horizontalpodautoscalers
      - ingress
      - jobs
      - limitranges
      - namespaces
      - nodes
      - pods
      - persistentvolumes
      - persistentvolumeclaims
      - resourcequotas
      - replicasets
      - replicationcontrollers
      - serviceaccounts
      - services
    verbs: ["get", "list", "watch", "create", "update", "patch", "delete"]
```

Bind the role to service account

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: app-rolebinding
  namespace: webapps
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: app-role
subjects:
- namespace: webapps
  kind: ServiceAccount
  name: jenkins
```

Generate token using service account in the namespace:

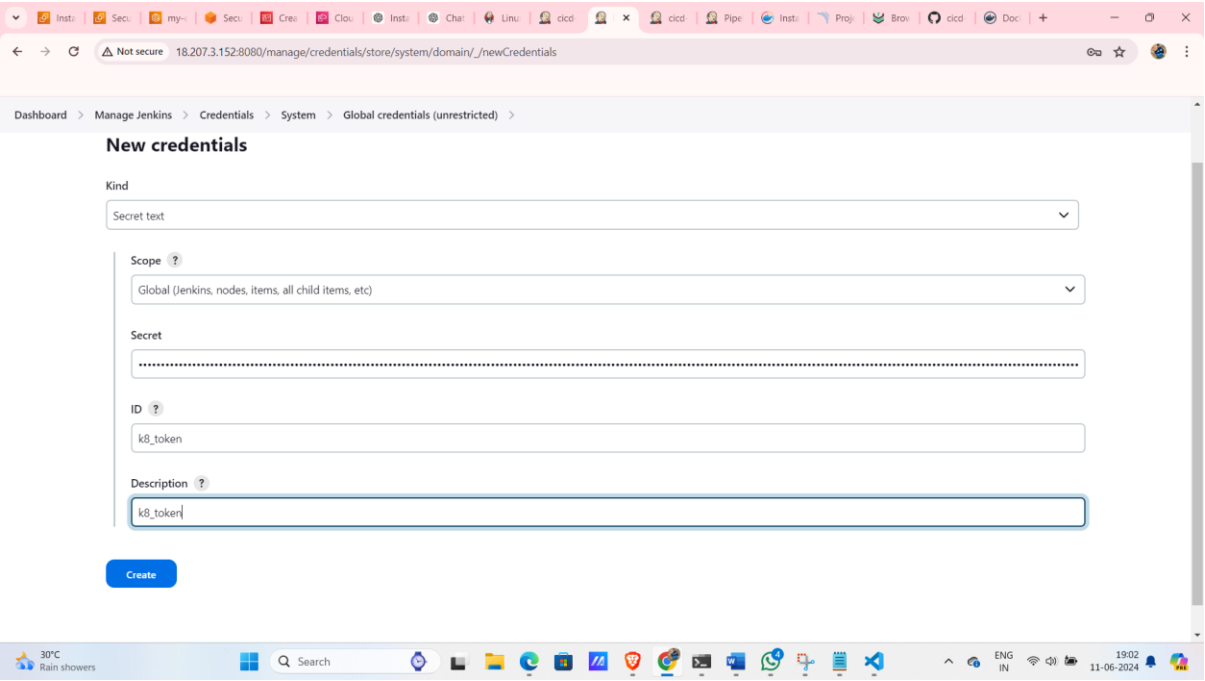
```yaml
apiVersion: v1
kind: Secret
type: kubernetes.io/service-account-token
metadata:
  name: mysecretname
  annotations:
    kubernetes.io/service-account.name: myserviceaccount
```

kubectl -n webapps describe secret mysecretname

```
ubuntu@ip-172-31-54-141:~$ kubectl create namespace webapps
namespace/webapps created
ubuntu@ip-172-31-54-141:~$ vim service_account.yaml
ubuntu@ip-172-31-54-141:~$ kubectl apply -f service_account.yaml
serviceaccount/jenkins created
ubuntu@ip-172-31-54-141:~$ vim role.yaml
ubuntu@ip-172-31-54-141:~$ kubectl apply -f role.yaml
role.rbac.authorization.k8s.io/app-role created
ubuntu@ip-172-31-54-141:~$ vim role_bind.yaml
ubuntu@ip-172-31-54-141:~$ kubectl apply -f role_bind.yaml
rolebinding.rbac.authorization.k8s.io/app-rolebinding created
ubuntu@ip-172-31-54-141:~$ vim secret.yaml
ubuntu@ip-172-31-54-141:~$ kubectl apply -f se
secret.yaml           service_account.yaml
ubuntu@ip-172-31-54-141:~$ kubectl apply -f se
secret.yaml           service_account.yaml
ubuntu@ip-172-31-54-141:~$ kubectl apply -f se
secret.yaml           service_account.yaml
ubuntu@ip-172-31-54-141:~$ kubectl apply -f secret.yaml -n webapps
secret/mysecretname created
ubuntu@ip-172-31-54-141:~$ kubectl -n webapps describe secret mysecretname
Name:         mysecretname
Namespace:    webapps
Labels:       <none>
Annotations:  kubernetes.io/service-account.name: jenkins
              kubernetes.io/service-account.uid: 345fc6d1-ff35-47bb-a3d3-cf85fec3a0d7

Type:  kubernetes.io/service-account-token

Data
====
namespace:  7 bytes
token:      eyJhbGciOiJSUzI1NiIsImtpZCI6ImRINUUzZHoxX1VpU1VONkpGQmljSEFPckt4bGpIeDhmaDVxdV9BbHYzb3cifQ.eyJpc3MiOiJrdWJlcm5ldGVzL3NlcnZpY2VhY2NvdW50Iiwia3ViZ
XJuZXRlcy5pby9zZXJ2aWNlYWNjb3VudC9uYW1lc3BhY2UiOiJ3ZWJhcHBzIiwia3ViZXJuZXRlcy5pby9zZXJ2aWNlYWNjb3VudC9zZWNyZXQubmFtZSI6Im15c2VjcmV0bmFtZSIsImt1YmVybmV0ZXMua
W8vc2VydmljZWFjY291bnQvc2VydmljZS1hY2NvdW50LmShbWUiOiJqZW5raW5zIiwia3ViZXJuZXRlcy5pby9zZXJ2aWNlYWNjb3VudC9zZXJ2aWNlLWFjY291bnQudWlkIjoiMzQ1ZmM2ZDEtZmYzNS00N
2JiLWEzZDMtY2Y4NWZlYzNhMGQ3Iiwic3ViIjoic3lzdGVtOnNlcnZpY2VhY2NvdW50OndlYmFwcHM6amVua2lucyJ9.s04GDXjsbB7P5YrWGSpNSNOv4kBiYLDuiVivaFUVtODYsge8v2L0TgdNdyK0Beur
xKLjECplBtruixRkvtJLY4zR9fWNv5M4P2TS0jsGafBXSzYIsDeDcmwOlR3YiP9s2f-KODYkjcHcTtLKaWyFGNRqUoOZdneRhxkndkL-S-gMHQ-zzd7ubyoU3w0HH3u2zW8NQae-3EOCVki-tpwhgauLU_Id
ALVA0cLP2gj2tMnba8XpLfFNA1hemQBt0cGha5PPAvjrJ27mlZ-LYe9ElvIglWsjkj-wvnGdv1q6GHgm7EQgALhSKJnBIMFWY8yhfAHQcsXjyLooOqz4wWq5lA
ca.crt:     1107 bytes
ubuntu@ip-172-31-54-141:~$
```

Add this token in Jenkins server

Goto→ manage Jenkins→ credentials→global→kind= secret text

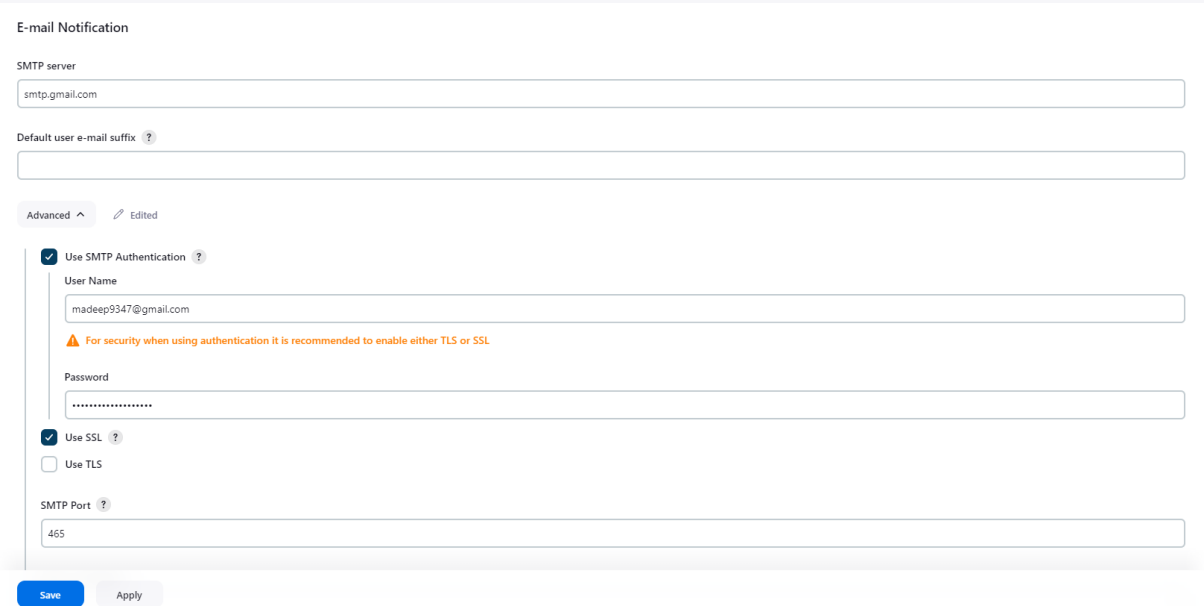


Email Notification Configurations:

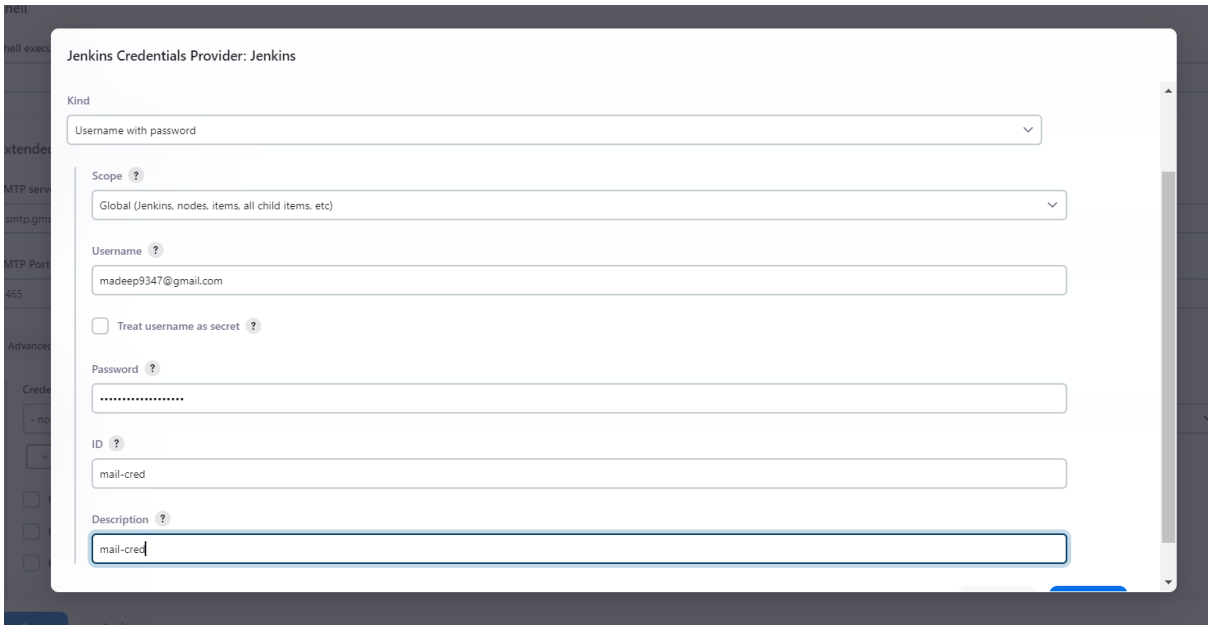
Goto this URL https://myaccount.google.com/apppasswords

generate apppassword and copy that password jijv akam wedd ujip

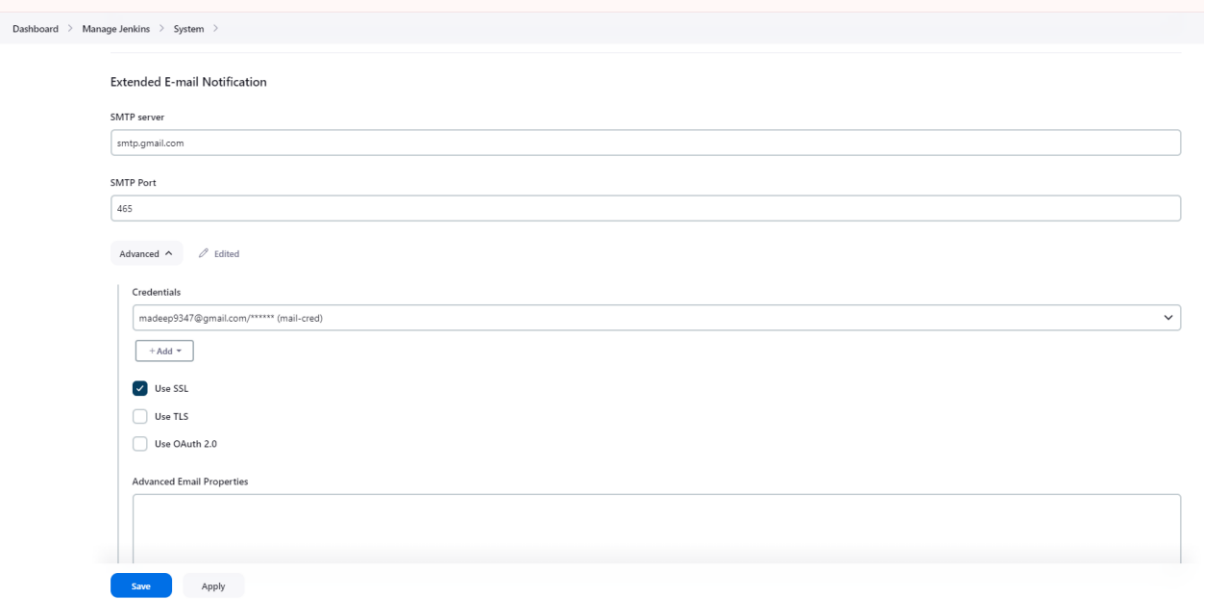
next go to Jenkins→manage Jenkins→system→E-mail Notification→smtp server= smtp.gmail.com, Advanced→ Use smtp Authentication→username="<yourgamilname>", password="<apppassword>", port= 465 and Test configuration by sending test e-mail.



Goto manage Jenkins→ credentials→ add the gmail and password



Now goto→ manage Jenkins→ system→ Extended E-mail Notification→ smtp-server=smtp.gmail.com, select the mail-cred



Now write the Jenkinsfile

```
pipeline {
    agent any

    tools {
        jdk 'jdk17'
        maven 'maven3'

    }

    environment {
        SCANNER_HOME = tool 'sonar-scanner'
    }

    stages {
        stage('Git Checkout') {
            steps {
                git branch: 'main', url: 'https://github.com/Madeep9347/cicd-project7.git'
            }
        }

        stage('Compile') {
            steps {
                sh "mvn compile"
            }
        }

        stage('Test') {
```

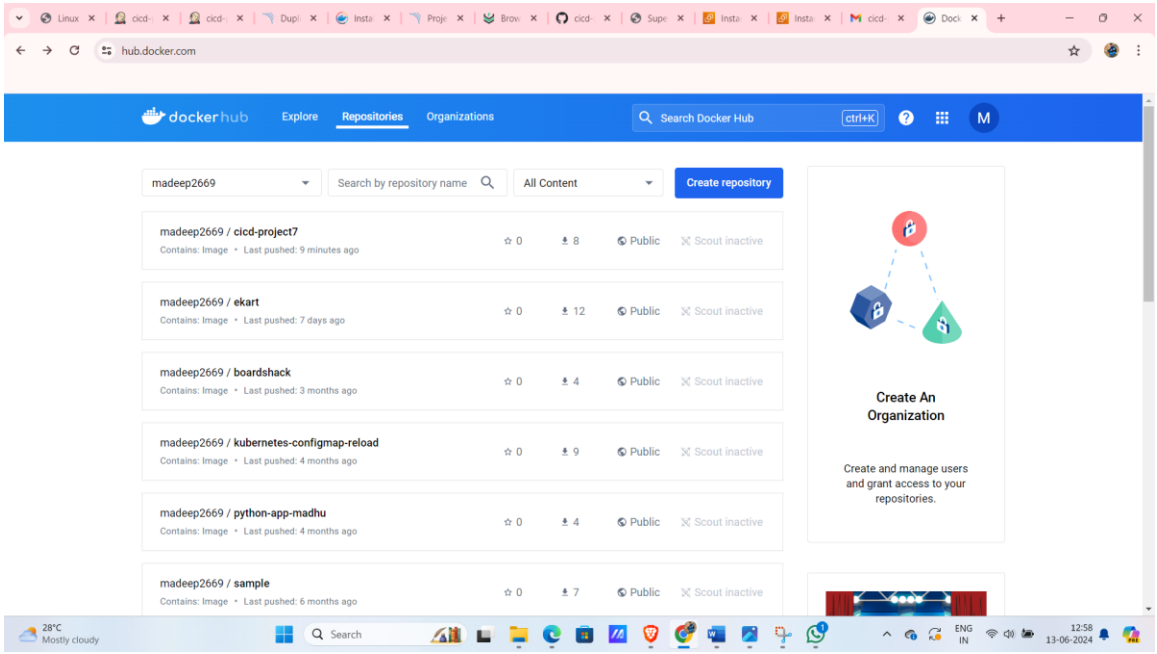
```
    steps {
        sh "mvn package -DskipTests=true"
    }
}

stage('Trivy Scan File System') {
    steps {
        sh "trivy fs --format table -o trivy-fs-report.html ."
    }
}

stage('SonarQube Analysis') {
    steps {
        withSonarQubeEnv('sonar') {
            sh '''$SCANNER_HOME/bin/sonar-scanner \
                -Dsonar.projectKey=Mission \
                -Dsonar.projectName=Mission \
                -Dsonar.java.binaries=.'''
        }
    }
}

stage('Build') {
    steps {
        sh "mvn package -DskipTests=true"
    }
}
 stage('Deploy Artifacts To Nexus') {
    steps {
```

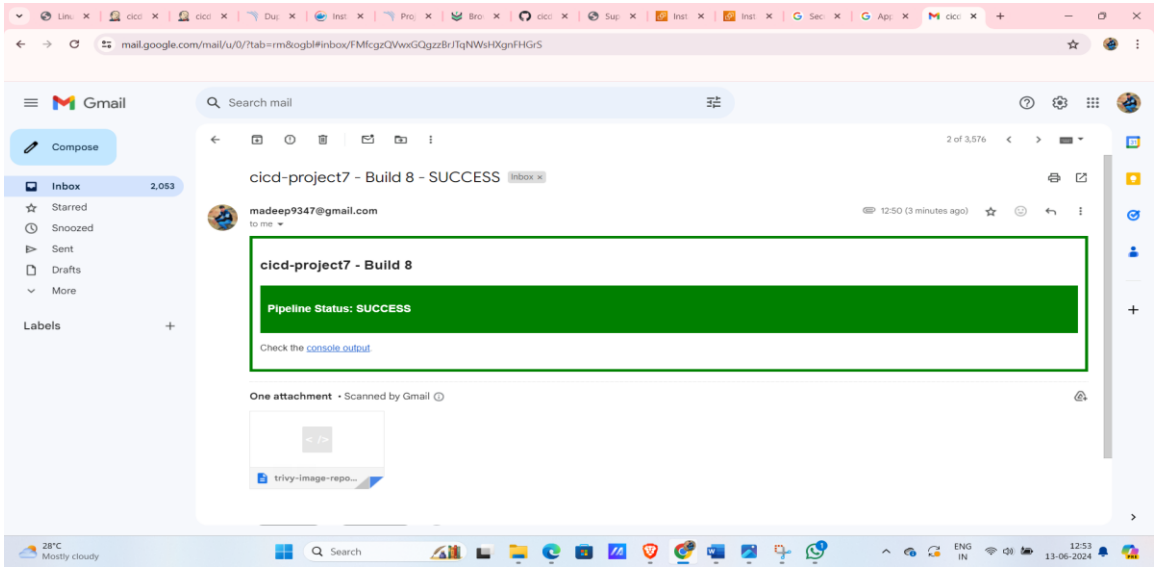
```
            withMaven(globalMavenSettingsConfig: 'maven-setting', jdk: 'jdk17', maven: 'maven3', mavenSettingsConfig: '', traceability: true) {
                sh "mvn deploy -DskipTests=true"
            }
        }
    }

    stage('Build & Tag Docker Image') {
        steps {
            script {
                withDockerRegistry(credentialsId: 'docker-cred', toolName: 'docker') {
                    sh "docker build -t madeep2669/cicd-project7:latest ."
                }
            }
        }
    }

    stage('Trivy Scan Image') {
        steps {
            sh "trivy image --format table -o trivy-image-report.html madeep2669/cicd-project7:latest"
        }
    }

    stage('Publish Docker Image') {
        steps {
            script {
                withDockerRegistry(credentialsId: 'docker-cred', toolName: 'docker') {
                    sh "docker push madeep2669/cicd-project7:latest"
                }
```

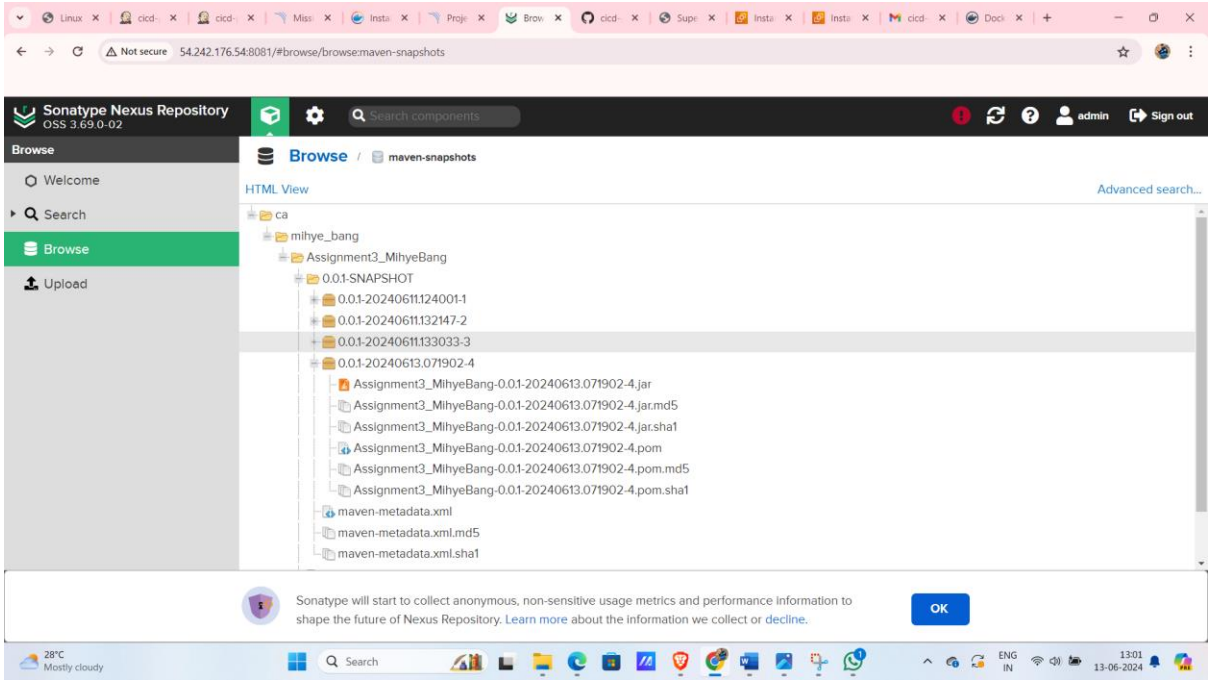
```
            }
        }
    }
    stage('Deploy to EKS') {
        steps {
            withKubeConfig(caCertificate: '', clusterName: 'my-eks22', contextName: '',
credentialsId: 'k8-token', namespace: 'webapps', restrictKubeConfigAccess: false, serverUrl:
'https://FE0E7FFC80B64E124F6F3EA8EDA2FE7E.sk1.ap-south-1.eks.amazonaws.com') {
                sh "kubectl apply -f ds.yml -n webapps"
                sleep 60
            }
        }
    }
    stage('Verify deployment') {
        steps {
            withKubeConfig(caCertificate: '', clusterName: 'my-eks22', contextName: '',
credentialsId: 'k8-token', namespace: 'webapps', restrictKubeConfigAccess: false, serverUrl:
'https://FE0E7FFC80B64E124F6F3EA8EDA2FE7E.sk1.ap-south-1.eks.amazonaws.com') {
                sh "kubectl get pods -n webapps"
                sh "kubectl get svc -n webapps"
            }
        }
    }
}
post {
    always {
        script {
            def jobName = env.JOB_NAME
            def buildNumber = env.BUILD_NUMBER
            def pipelineStatus = currentBuild.result ?: 'UNKNOWN'
            def bannerColor = pipelineStatus.toUpperCase() == 'SUCCESS' ? 'green' : 'red'
```

```
            def body = """
                <html>
                <body>
                <div style="border: 4px solid ${bannerColor}; padding: 10px;">
                <h2>${jobName} - Build ${buildNumber}</h2>
                <div style="background-color: ${bannerColor}; padding: 10px;">
                <h3 style="color: white;">Pipeline Status: ${pipelineStatus.toUpperCase()}</h3>
                </div>
                <p>Check the <a href="${BUILD_URL}">console output</a>.</p>
                </div>
                </body>
                </html>
            """

            emailext (
                subject: "${jobName} - Build ${buildNumber} - ${pipelineStatus.toUpperCase()}",
                body: body,
                to: 'madeep9347@gmail.com',
                from: 'jenkins@example.com',
                replyTo: 'jenkins@example.com',
                mimeType: 'text/html',
                attachmentsPattern: 'trivy-image-report.html'
            )
        }
    }
  }
}
```

```
[Pipeline] sh
+ kubectl get pods -n webapps
NAME                                    READY   STATUS    RESTARTS   AGE
cicd-project7-deployment-79465f874d-t8bgk   1/1     Running   0          41h
cicd-project7-deployment-79465f874d-wgpjq   1/1     Running   0          41h
[Pipeline] sh
+ kubectl get svc -n webapps
NAME                TYPE           CLUSTER-IP       EXTERNAL-IP                                                              PORT(S)          AGE
cicd-project7-ssvc  LoadBalancer   10.100.179.140   ac7b1a92512c243848be2a7df6fcee96-328134965.ap-south-1.elb.amazonaws.com   8080:30875/TCP   41h
[Pipeline] }
[kubernetes-cli] kubectl configuration cleaned up
[Pipeline] // withKubeConfig
[Pipeline] }
[Pipeline] // withEnv
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (Declarative: Post Actions)
[Pipeline] script
[Pipeline] {
[Pipeline] emailext
Sending email to: madeep9347@gmail.com
[Pipeline] }
[Pipeline] // script
[Pipeline] }
```



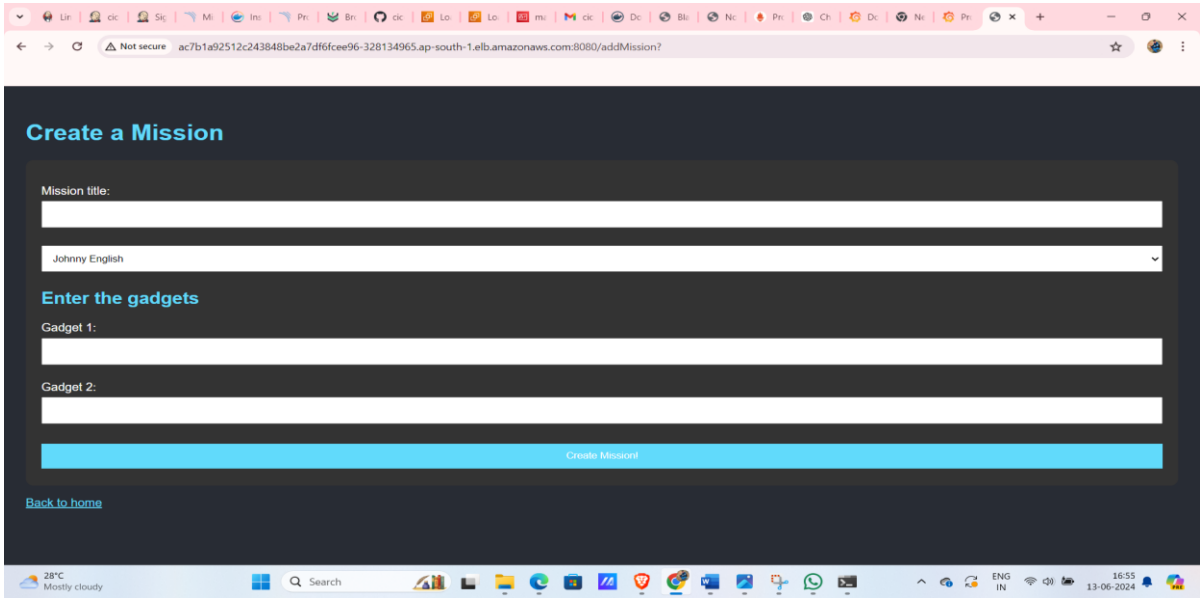





```
ubuntu@ip-172-31-54-141:~$ kubectl get all -n webapps
NAME                                                READY   STATUS    RESTARTS   AGE
pod/cicd-project7-deployment-79465f874d-t8bgk       1/1     Running   0          42h
pod/cicd-project7-deployment-79465f874d-wgpjq       1/1     Running   0          42h

NAME                                 TYPE           CLUSTER-IP       EXTERNAL-IP
                PORT(S)          AGE
service/cicd-project7-ssvc   LoadBalancer   10.100.179.140   ac7b1a92512c243848be2a7df6fcee96-328134965.ap-south-1.elb.a
mazonaws.com   8080:30875/TCP   42h

NAME                                        READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/cicd-project7-deployment    2/2     2            2           42h

NAME                                                   DESIRED   CURRENT   READY   AGE
replicaset.apps/cicd-project7-deployment-79465f874d    2         2         2       42h
ubuntu@ip-172-31-54-141:~$
```

Access the Application using the External-ip

http://ac7b1a92512c243848be2a7df6fcee96-328134965.ap-south-1.elb.amazonaws.com:8080/addMission?



Setup Prometheus,Grafana,node-exporter,blackbox-exporter

Install Node Exporter in Jenkins server

1. Download Node Exporter:

wget https://github.com/prometheus/node_exporter/releases/download/v1.8.1/node_exporter-1.8.1.linux-amd64.tar.gz

2. Extract the Tarball:

tar -xzvf node_exporter-1.8.1.linux-amd64.tar.gz

3. Move to the Extracted Directory:

cd node_exporter-1.8.1.linux-amd64

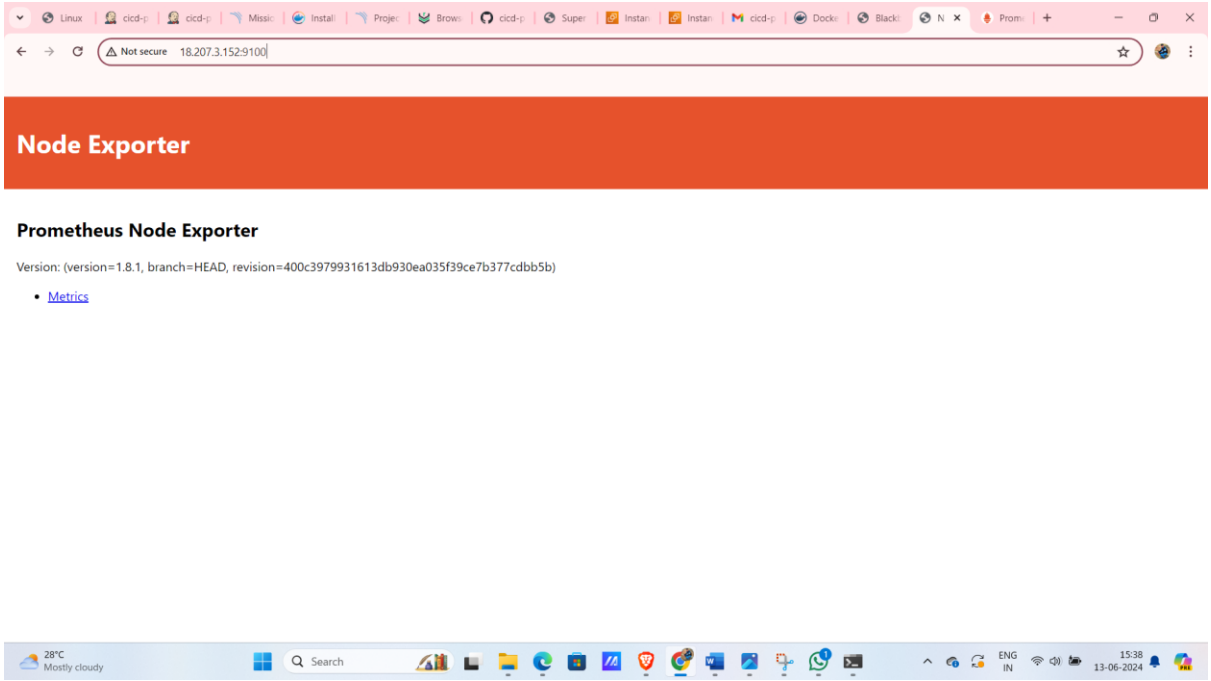
mv node_exporter-1.8.1.linux-amd64 node_exporter

4. Run Node Exporter:

./node_exporter &

5. Verify Node Exporter is Running:

Open a web browser and navigate to http://18.207.3.152:9100/metrics.



2. Install Blackbox Exporter in Jenkins server

1. Download Blackbox Exporter:

Wget https://github.com/prometheus/blackbox_exporter/releases/download/v0.25.0/blackbox_exporter-0.25.0.linux-amd64.tar.gz

2. Extract the Tarball:

tar -xzvf blackbox_exporter-0.25.0.linux-amd64.tar.gz

3. Move to the Extracted Directory:

cd blackbox_exporter-0.25.0.linux-amd64

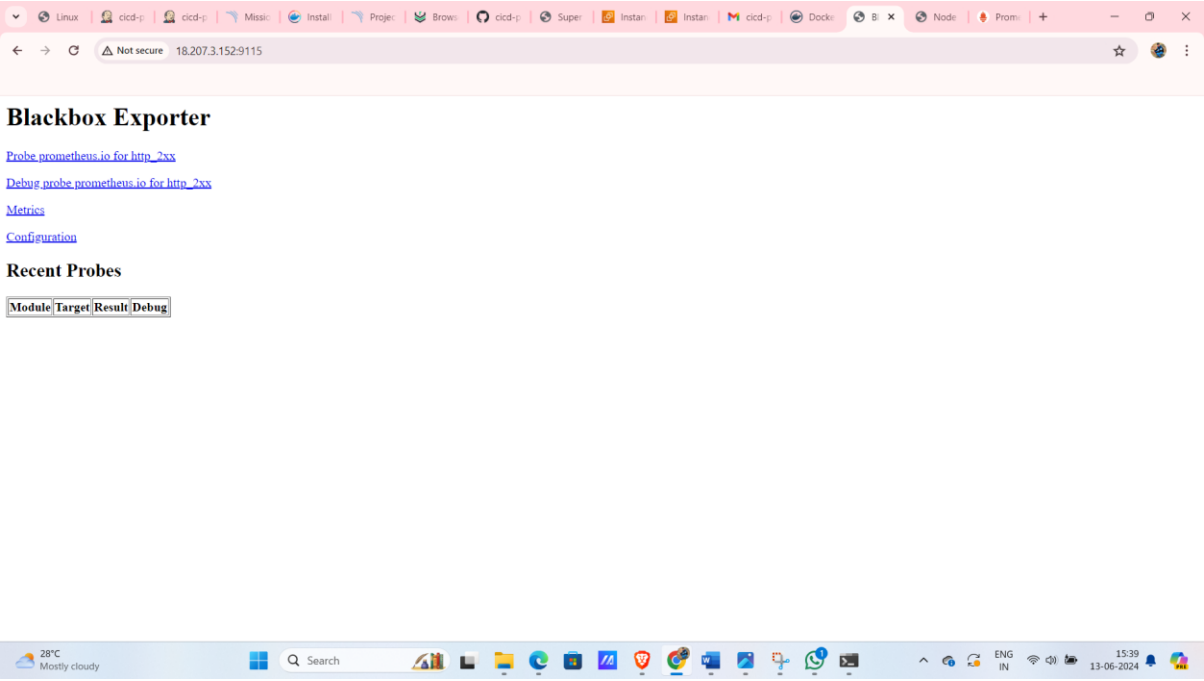
mv blackbox_exporter-0.25.0.linux-amd64 blackbox_exporter

4. Run Blackbox Exporter:

./blackbox_exporter &

5. Verify Blackbox Exporter is Running:

Open a web browser and navigate to http://18.207.3.152:9115/metrics.



Install Prometheus in Jenkins server

1. Download Prometheus:

wget

https://github.com/prometheus/prometheus/releases/download/v2.52.0/prometheus-2.52.0.linux-amd64.tar.gz

2. Extract the Tarball:

tar -xzvf prometheus-2.52.0.linux-amd64.tar.gz

3. Move to the Extracted Directory:

cd prometheus-2.52.0.linux-amd64

mv prometheus-2.52.0.linux-amd64 prometheus

Configuration

Prometheus Configuration

To scrape metrics from Node Exporter and Blackbox Exporter, you need to configure Prometheus.

1. Edit the Prometheus Configuration File (prometheus.yml):

```
global:
  scrape_interval: 15s
scrape_configs:
  - job_name: 'prometheus'
    static_configs:
      - targets: ['localhost:9090']


  - job_name: 'node_exporter'
    static_configs:
      - targets: ['18.207.3.152:9100']          # replace with your public-ip
  - job_name: 'blackbox_exporter'
    metrics_path: /probe
    params:
      module: [http_2xx]
    static_configs:
      - targets:
        - http://localhost:9115
    relabel_configs:
      - source_labels: [__address__]
```

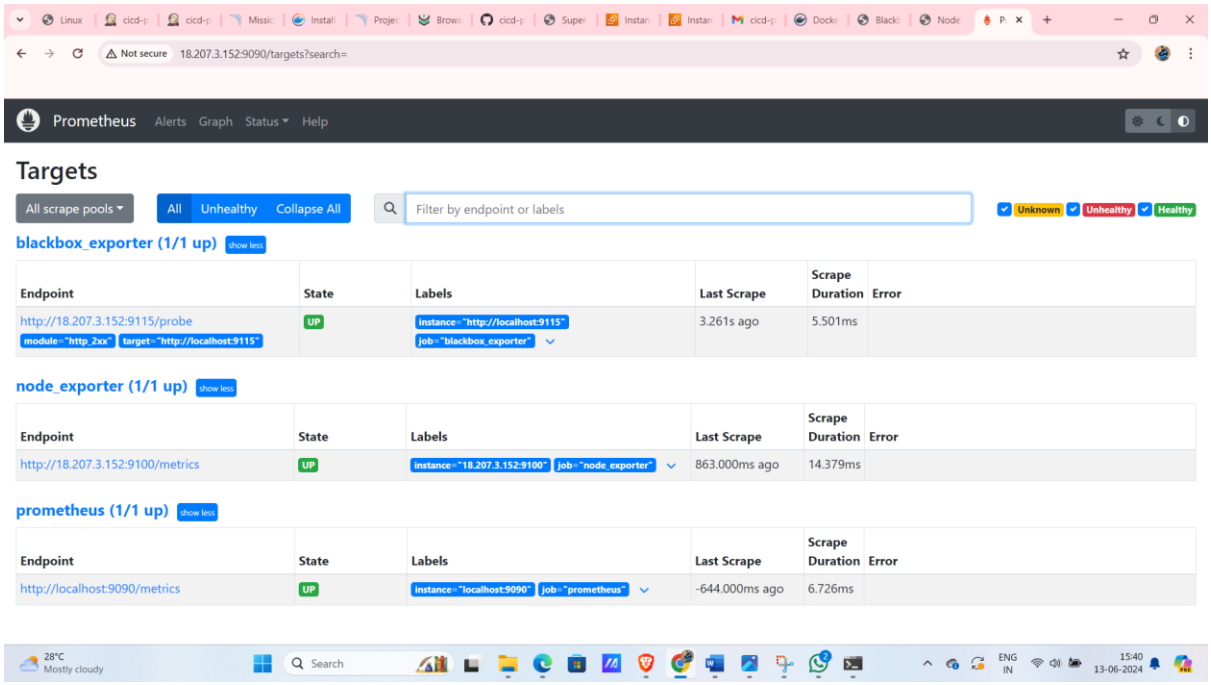
```
    target_label: __param_target
  - source_labels: [__param_target]
    target_label: instance
  - target_label: __address__
    replacement: 18.207.3.152:9115          # replace with your public-ip
```

4. Run Prometheus:

./prometheus &

5. Verify Prometheus is Running:

Open a web browser and navigate to http://18.207.3.152:9090.



Installation and Setup of Grafana

This guide will walk you through the steps to download, install, and set up Grafana on a Linux-based system.

1. Update your package list:

sudo apt-get update

2. Install necessary packages:

sudo apt-get install -y adduser libfontconfig1 musl

3. Download the Grafana Enterprise package:

Wget https://dl.grafana.com/enterprise/release/grafana-enterprise_11.0.0_amd64.deb

4. Install Grafana using dpkg:

sudo dpkg -i grafana-enterprise_11.0.0_amd64.deb

5. Start and Enable Grafana

1. Start the Grafana service:

sudo systemctl start grafana-server

2. Enable the Grafana service to start on boot:

sudo systemctl enable grafana-server

6. Access Grafana

1. Open a web browser and navigate to:

2. http://18.207.3.152:3000 # replace with your publicip

3. Log in to Grafana:

The default username is admin.

The default password is admin.

4. Change the default password:

Upon first login, you will be prompted to change the default password. Enter a new password and confirm it.

Configure Grafana

1. Add a Data Source:

Navigate to Configuration > Data Sources.

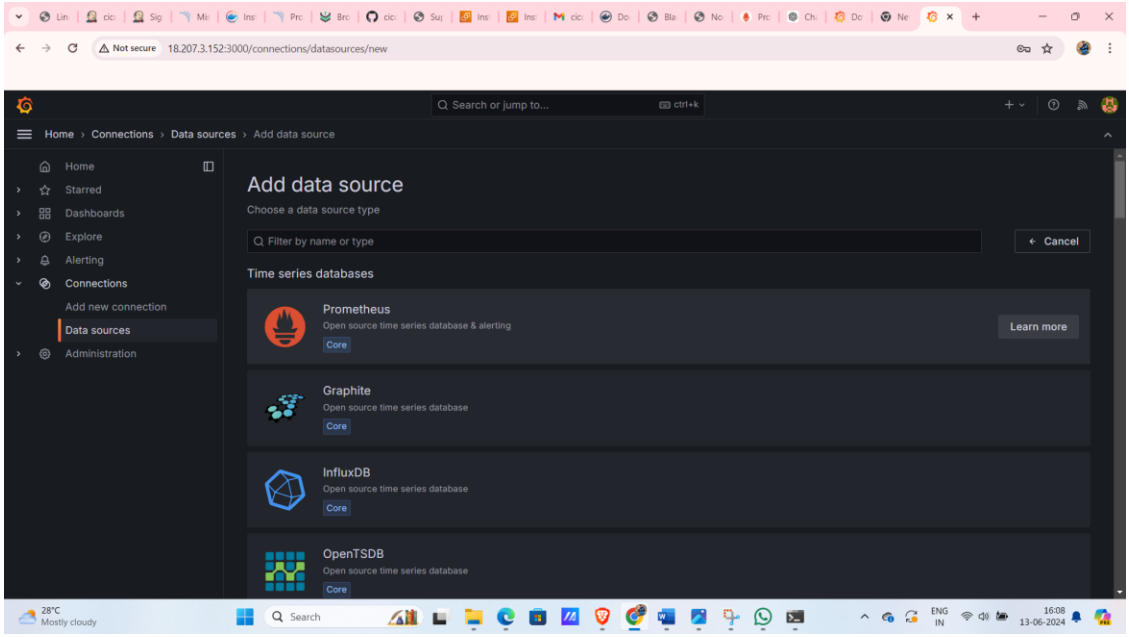
Click Add data source.

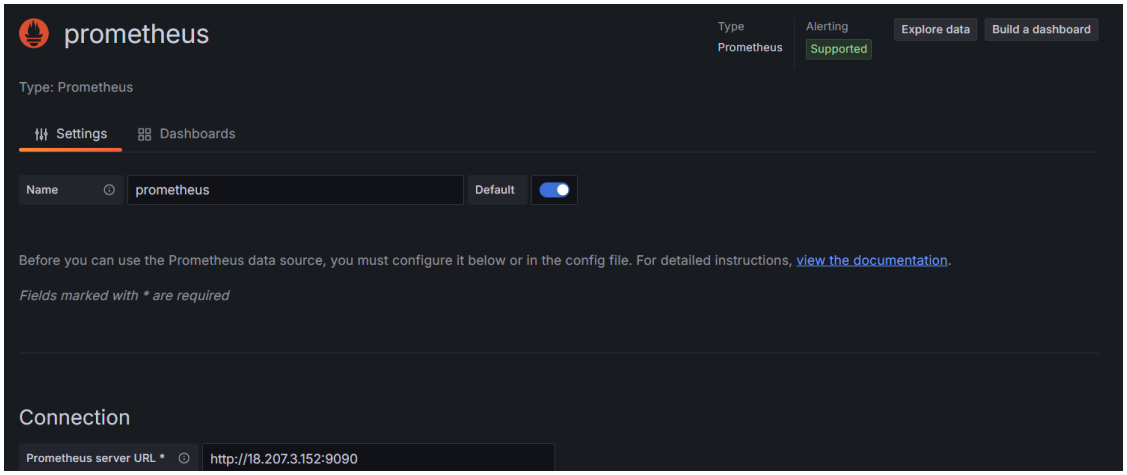
Choose your desired data source type (e.g., Prometheus).

Configure the data source with the appropriate URL

(e.g., http://localhost:9090 for Prometheus).

Click Save & Test.

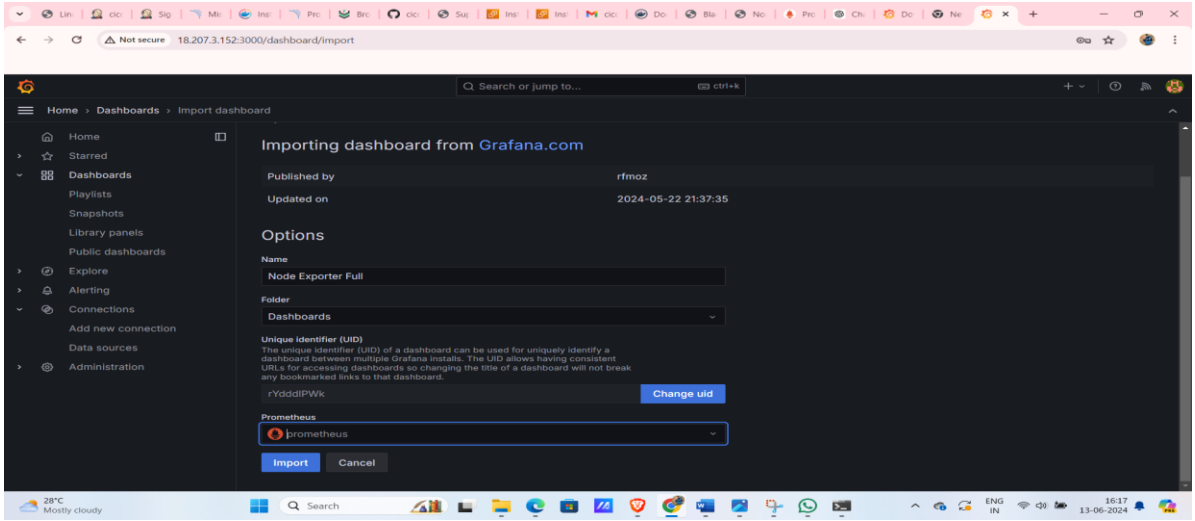




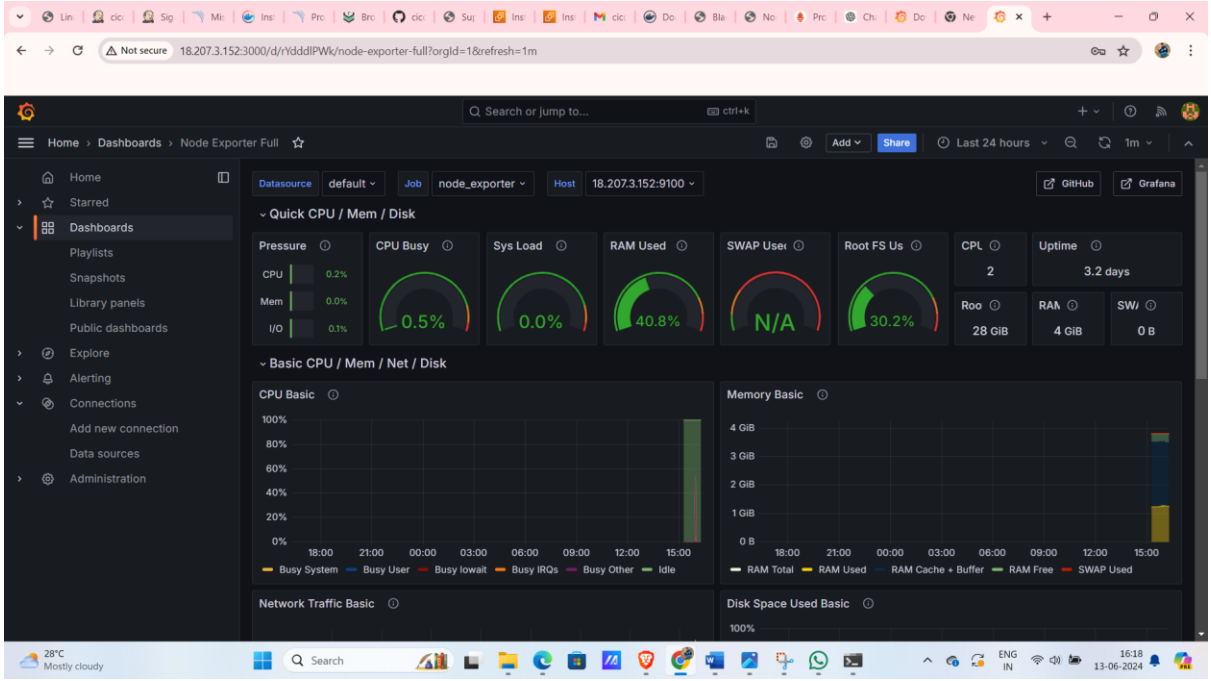
Save and Test.

Next goto→ dashboards→Create Dashboard→ Import dashboard

For node-exporter dashboard-id is 1860 import that and select datasource



You will get visualization dashboard for Jenkins server



Similarly create a dashboard for Monitoring the Website

For Blackbox-exporter dashboard-id is 7587