



PHP APP EKS

[sub-page1](#)

<https://www.notion.so/PHP-Application-CICD-Steps-1feefe9912cb8027b834e4c44a2a81e1>

Hello everyone, This document covers the **end-to-end deployment** of a PHP-APP on Amazon EKS using **GitOps**, with integrated **DevSecOps** best practices for security, automation, and monitoring. I Hope this detailed guide is useful.

[CLICK HERE FOR GITLAB REPOSITORY](#)

Steps:-

Step 1 — Launch 3 Ubuntu (Latest Version) Instance on AWS.

Step 2 — Install Jenkins, Docker. Create a Sonarqube Container using Docker on Jenkins server.

Step 3 — Install Plugins like gitlab, docker, Sonarqube Scanner, and more in Jenkins.

step 4 — Configure jenkins, integrate sonarqube, and add credentials.

Step 5 — setup k8s server as jenkins agent and connecting them.

Step 6 — Create a Pipeline Project in Jenkins using a Declarative Pipeline

step 7 — setup mysql database on second server.

Step 8 — Create AWS EKS cluster

Step 9 — Install Helm

Step 10 — Install Prometheus and Grafana using Helm

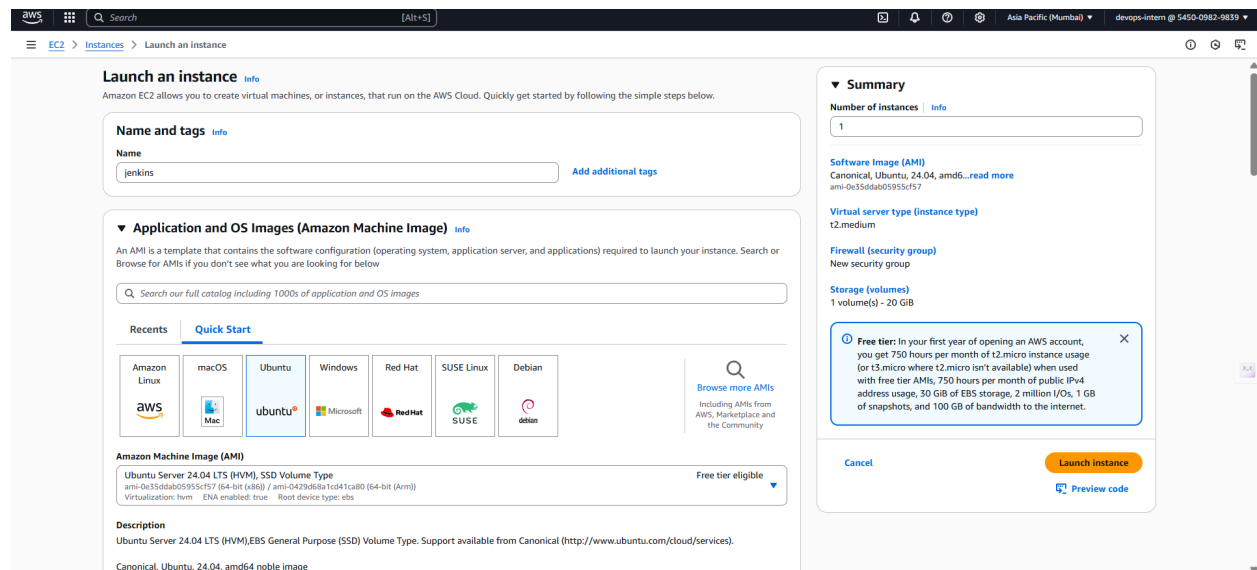
Step 11 — Install ArgoCD using Helm

Step 12 — Access the php-app on the Browser.

Step-by-Step Deployment

Step 1: Launch EC2 Instances

Launch an AWS t2.medium Instance. Use the image as Ubuntu. You can create a new key pair or use an existing one. Enable HTTP and HTTPS settings in the Security Group and open 8080 and 9000 port no for jenkins and sonarqube.



The screenshot displays the AWS Management Console interface for launching an EC2 instance. The 'Launch an instance' page is active, showing the 'Name and tags' section with the instance name 'jenkins'. The 'Application and OS Images' section shows 'Ubuntu' selected as the AMI. The 'Virtual server type' is set to 't2.medium'. The 'Firewall (security group)' is set to 'New security group'. The 'Storage (volumes)' section shows '1 volume(s) - 20 GiB'. A 'Free tier' notification is visible, stating that the user is eligible for the free tier. The 'Launch instance' button is highlighted in orange.

EC2

Instances

Launch an instance

▼ Instance type

Info

Get advice

Instance type

t2.medium

Family: t2

2 vCPU

4 GiB Memory

Current generation: true

On-Demand Linux base pricing: 0.0496 USD per Hour

On-Demand Ubuntu Pro base pricing: 0.0531 USD per Hour

On-Demand Windows base pricing: 0.0676 USD per Hour

On-Demand RHEL base pricing: 0.0784 USD per Hour

On-Demand SUSE base pricing: 0.1496 USD per Hour

All generations

Compare instance types

Additional costs apply for AMIs with pre-installed software

▼ Key pair (login)

Info

You can use a key pair to securely connect to your instance. Ensure that you have access to the selected key pair before you launch the instance.

Key pair name - required

todo

Create new key pair

▼ Network settings

Info

VPC - required

Info

vpc-0468c2f86b51ec6c (TODO-app-vpc)

Create new VPC

Subnet

Info

subnet-09b6fcb8ea07faa1a

VPC: vpc-0468c2f86b51ec6c

Owner: 541509829839

Availability Zone: ap-south-1a

Zone type: Availability Zone

IP addresses available: 4074

CIDR: 10.0.0.0/20

Create new subnet

Auto-assign public IP

Info

Enable

Additional charges apply when outside of free tier allowance

Firewall (security groups)

Info

A security group is a set of firewall rules that control the traffic for your instance. Add rules to allow specific traffic to reach your instance.

▼ Summary

Info

Number of instances

1

Software Image (AMI)

Canonical, Ubuntu, 24.04, amd64...read more

ami-0e35d6ab05955cf57

Virtual server type (instance type)

t2.medium

Firewall (security group)

New security group

Storage (volumes)

1 volume(s) - 20 GiB

Free tier: In your first year of opening an AWS account, you get 750 hours per month of t2.micro instance usage (or t3.micro where t2.micro isn't available) when used with free tier AMIs, 750 hours per month of public IPv4 address usage, 30 GiB of EBS storage, 2 million I/Os, 1 GB of snapshots, and 100 GB of bandwidth to the internet.

Cancel

Launch instance

Preview code

EC2

Instances

Launch an instance

▼ Description - required

Info

launch-wizard-5 created 2025-05-22T15:41:01.125Z

Inbound Security Group Rules

▼ Security group rule 1 (TCP, 22, 0.0.0.0/0)

Remove

Type

Info

ssh

Protocol

Info

TCP

Port range

Info

22

Source type

Info

Anywhere

Source

Info

0.0.0.0/0

Description - optional

Info

e.g. SSH for admin desktop

▼ Security group rule 2 (TCP, 80, 0.0.0.0/0)

Remove

Type

Info

HTTP

Protocol

Info

TCP

Port range

Info

80

Source type

Info

Anywhere

Source

Info

0.0.0.0/0

Description - optional

Info

e.g. SSH for admin desktop

▼ Security group rule 3 (TCP, 443, 0.0.0.0/0)

Remove

Type

Info

HTTPS

Protocol

Info

TCP

Port range

Info

443

Source type

Info

Anywhere

Source

Info

0.0.0.0/0

Description - optional

Info

e.g. SSH for admin desktop

▼ Summary

Info

Number of instances

1

Software Image (AMI)

Canonical, Ubuntu, 24.04, amd64...read more

ami-0e35d6ab05955cf57

Virtual server type (instance type)

t2.medium

Firewall (security group)

New security group

Storage (volumes)

1 volume(s) - 20 GiB

Free tier: In your first year of opening an AWS account, you get 750 hours per month of t2.micro instance usage (or t3.micro where t2.micro isn't available) when used with free tier AMIs, 750 hours per month of public IPv4 address usage, 30 GiB of EBS storage, 2 million I/Os, 1 GB of snapshots, and 100 GB of bandwidth to the internet.

Cancel

Launch instance

Preview code

Now create one more ec2 instance with same configuration

Now again create one more ec2 instance but this time in private subnet and open port 3306 for mysql database.

Step 2 — Install Jenkins, Docker. Create a Sonarqube Container using Docker on first server.

Connect to the instance via SSH:

```
ssh -i your-key.pem ubuntu@your-ec2-ip
```

2A: Install Jenkins.

Run below commands to Install Jenkins

#Installation of JAVA

```
sudo apt update
```

```
sudo apt install fontconfig openjdk-17-jre
```

```
java -version
```

#Installation of jenkins

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

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

```
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
```

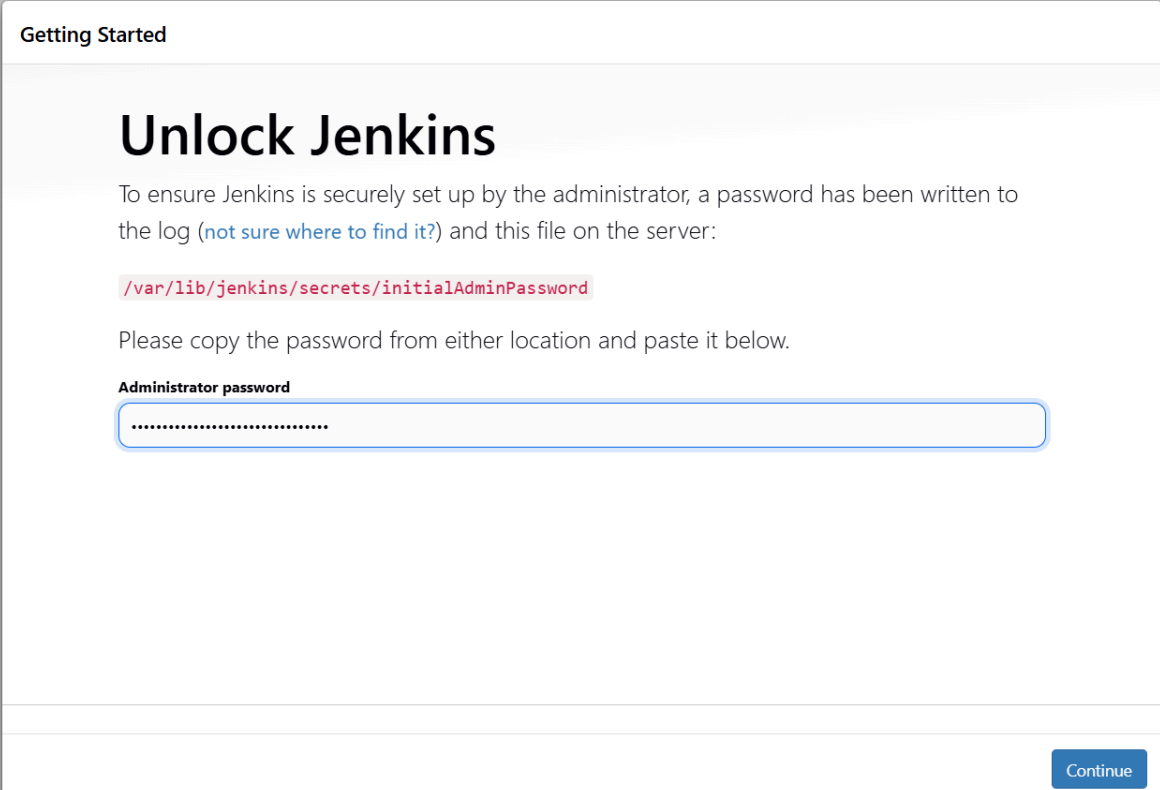
```
# Update the packages
sudo apt-get update
sudo apt-get install jenkins
jenkins --version
```

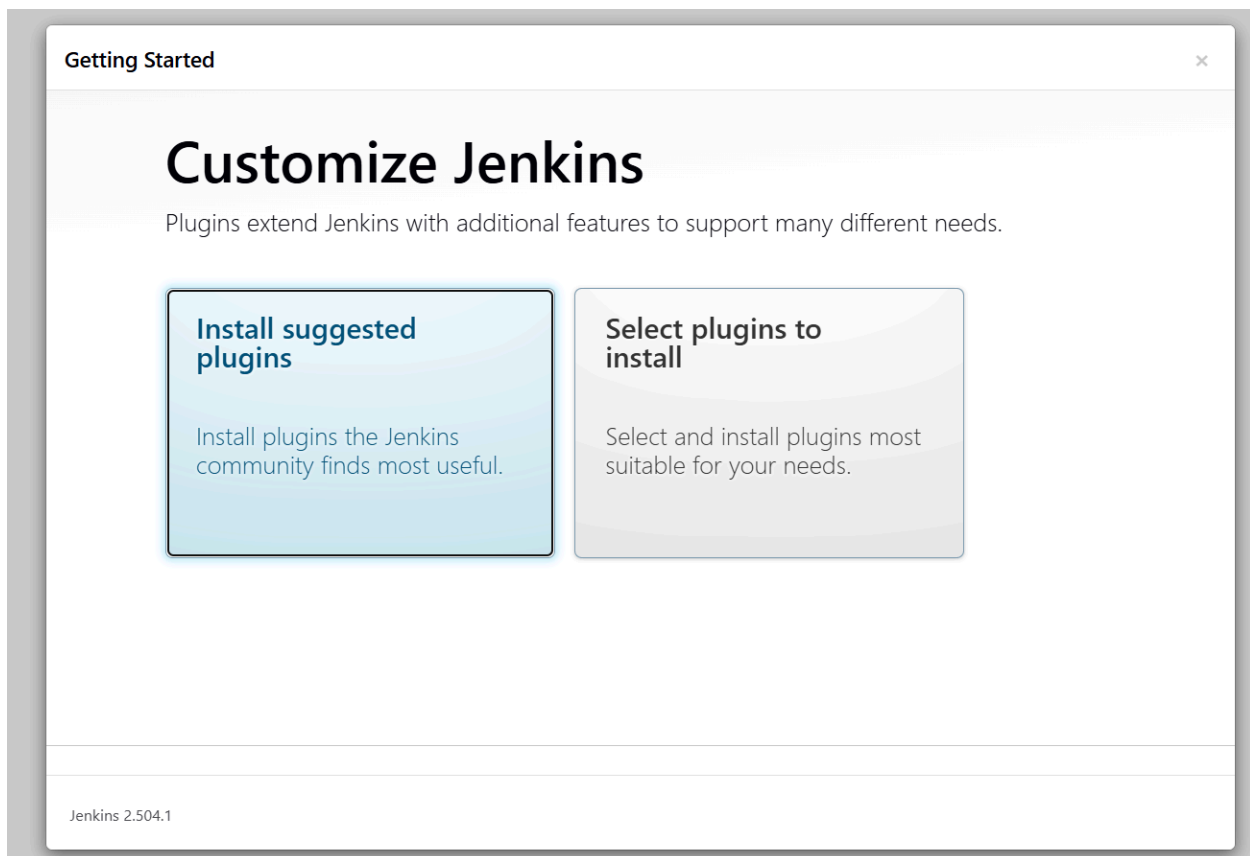
Once Jenkins is installed, now grab your Public IP Address and paste it to your browser like below.

<EC2 Public_IP_Address:8080>

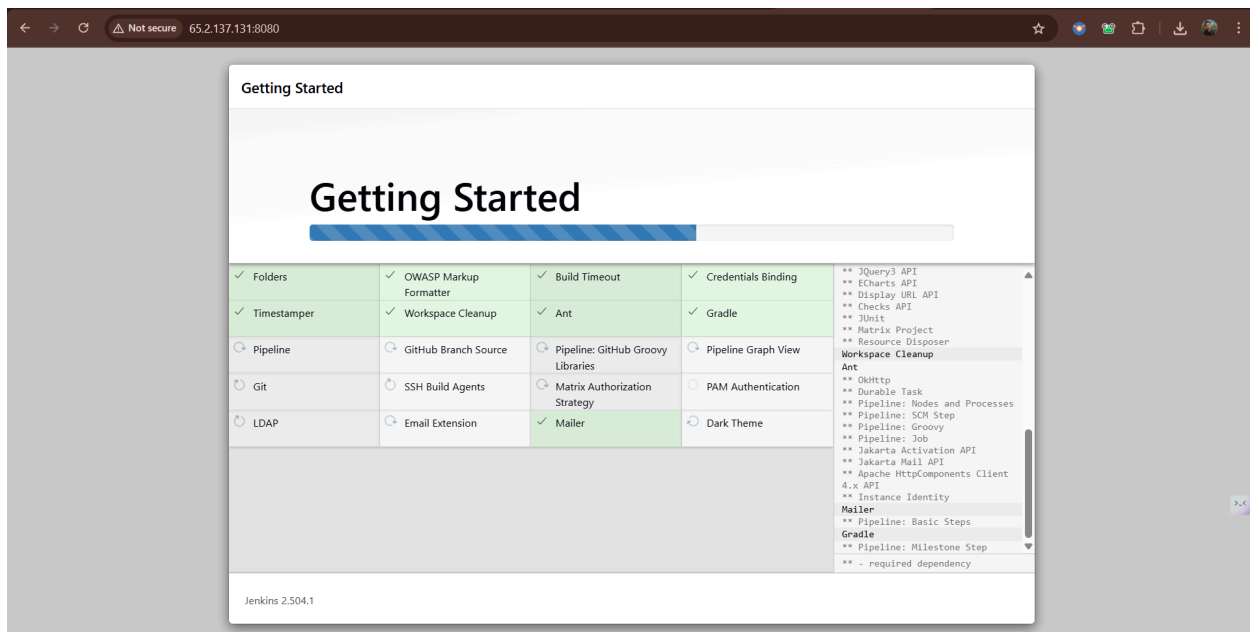
```
sudo cat /var/lib/jenkins/secrets/initialAdminPassword
```

Unlock Jenkins using an administrative password and install the suggested plugins.

The image shows the 'Getting Started' screen of the Jenkins web interface. The title 'Unlock Jenkins' is prominently displayed. Below it, a message explains that a password has been written to the log and a file on the server. The file path '/var/lib/jenkins/secrets/initialAdminPassword' is highlighted in red. A text prompt asks the user to copy the password from either location and paste it below. There is a text input field labeled 'Administrator password' with a blue border and a light blue background. The field contains a series of dots representing the password. At the bottom right of the screen, there is a blue button labeled 'Continue'.



Jenkins will now get install all the libraries.



Create a user click on save and continue.

Getting Started

Create First Admin User

Username

Jawaid

Password

.....

Confirm password

.....

Full name

Jawaid Akhtar

E-mail address

md.JawaidAkhtar@oyebusy.in

Jenkins 2.504.1

[Skip and continue as admin](#)

[Save and Continue](#)

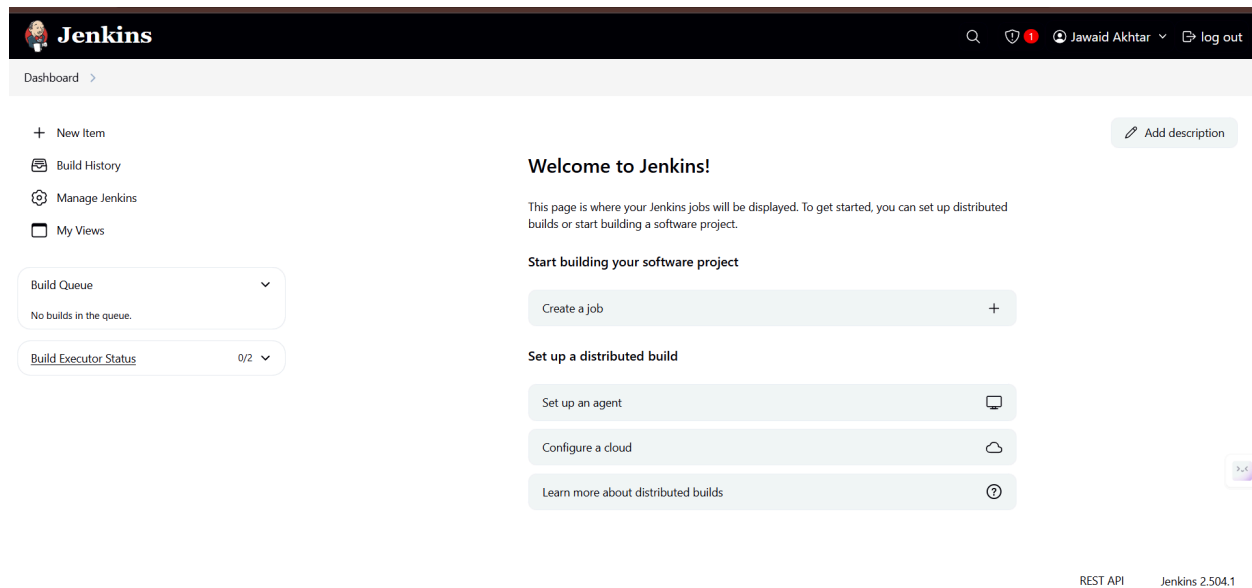
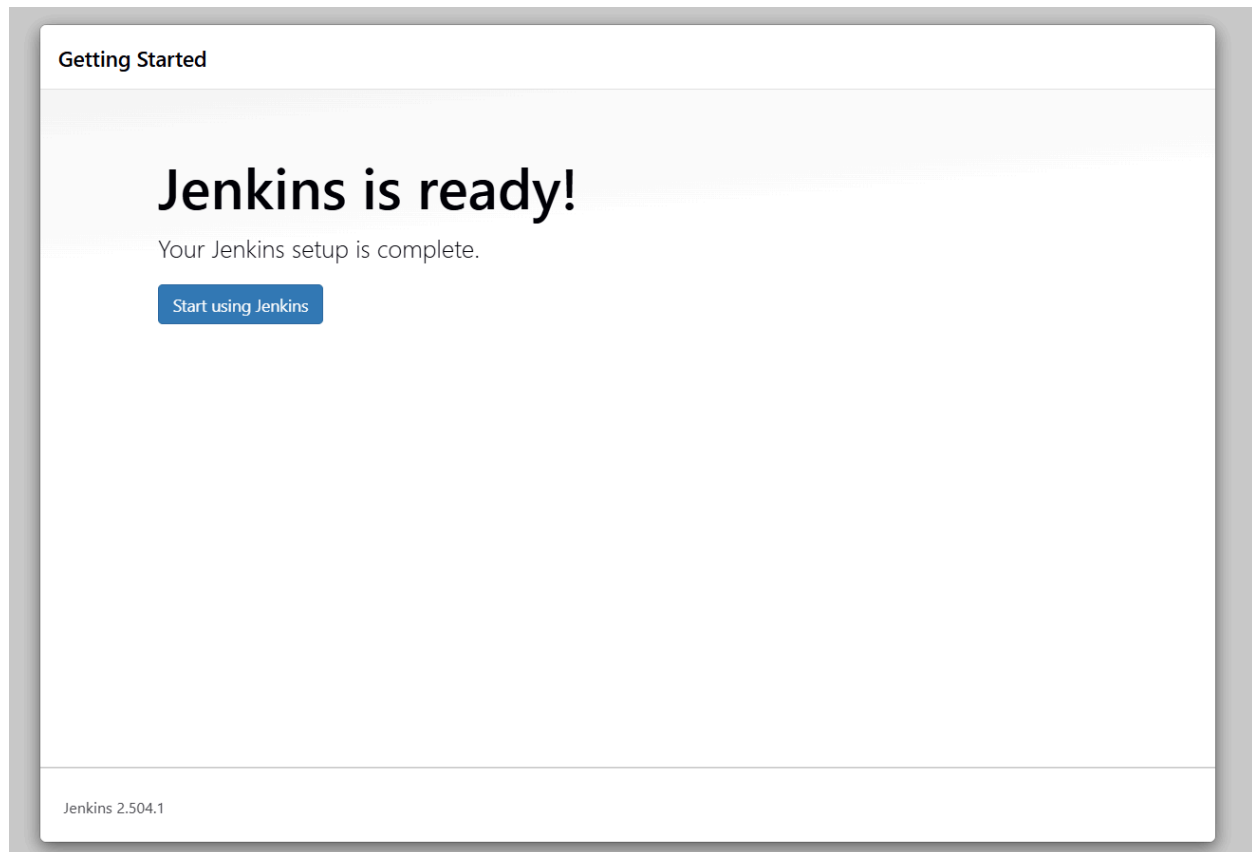
Instance Configuration

Jenkins URL:

The Jenkins URL is used to provide the root URL for absolute links to various Jenkins resources. That means this value is required for proper operation of many Jenkins features including email notifications, PR status updates, and the BUILD_URL environment variable provided to build steps.

The proposed default value shown is **not saved yet** and is generated from the current request, if possible. The best practice is to set this value to the URL that users are expected to use. This will avoid confusion when sharing or viewing links.

Jenkins Getting Started Screen.



2B : Install Docker.

Run below command to install docker.

```
# Run the following command to uninstall all conflicting packages:
for pkg in docker.io docker-doc docker-compose docker-compose-v2 podman-docker containerd runc; do sudo apt-get remove $pkg; done
```

```
# Add Docker's official GPG key:
```

```
sudo apt-get update
```

```
sudo apt-get install ca-certificates curl
```

```
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
```

```
# Add the repository to Apt sources:
```

```
echo \
```

```
"deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.asc] https://download.docker.com/linux/ubuntu \
```

```
$(. /etc/os-release && echo "${UBUNTU_CODENAME:-$VERSION_CODENAME}") stable" | \
```

```
sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
```

```
sudo apt-get update
```

```
# To install the latest version, run:
```

```
sudo apt-get install docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-plugin
```

```
docker --version
```

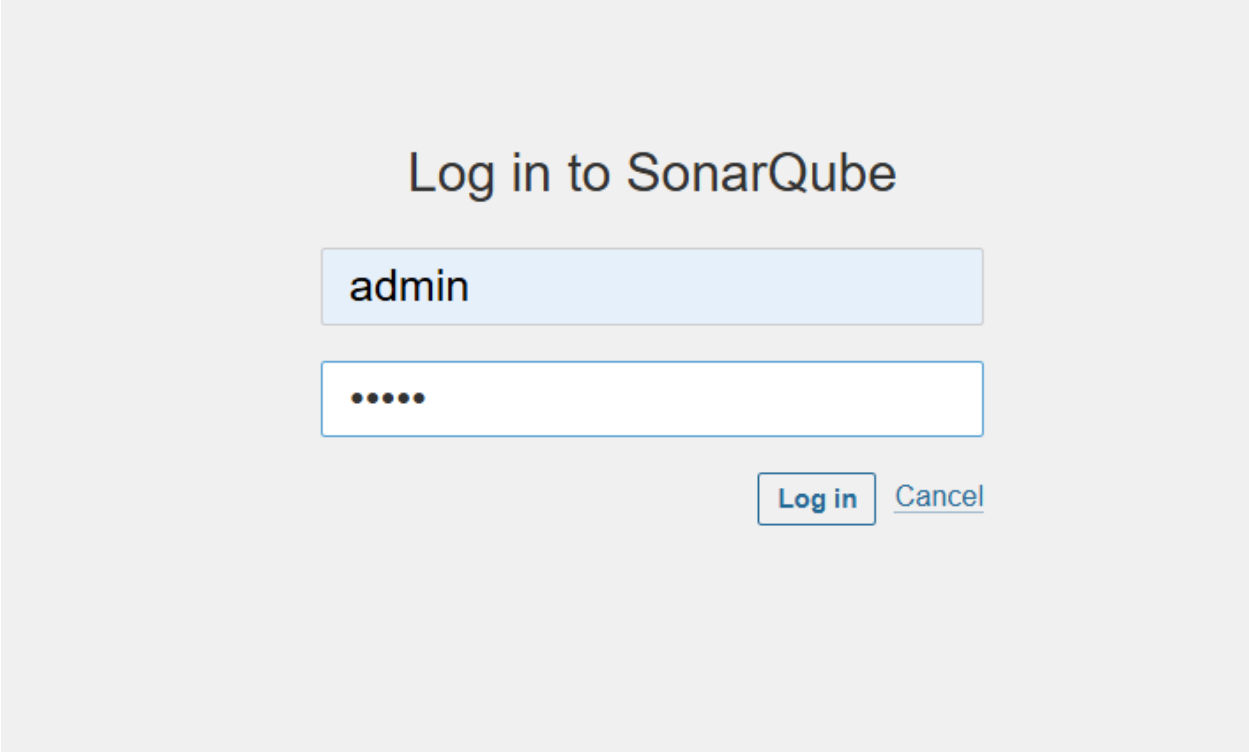
```
sudo usermod -aG docker $USER && newgrp docker #my case is ubuntu
```

After the docker installation, we will create a sonarqube container.

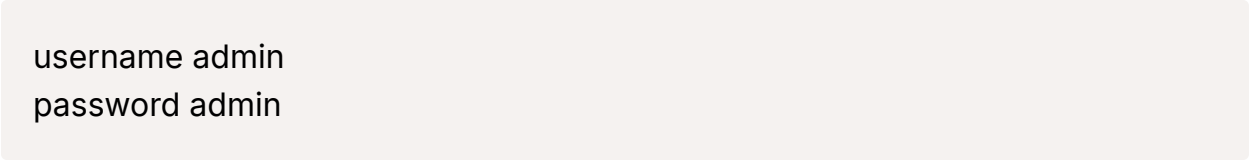
```
docker run -d --name sonar -p 9000:9000 sonarqube:lts-community
```

Now our sonarqube is up and running, just take the ip address and paste in browser as below:

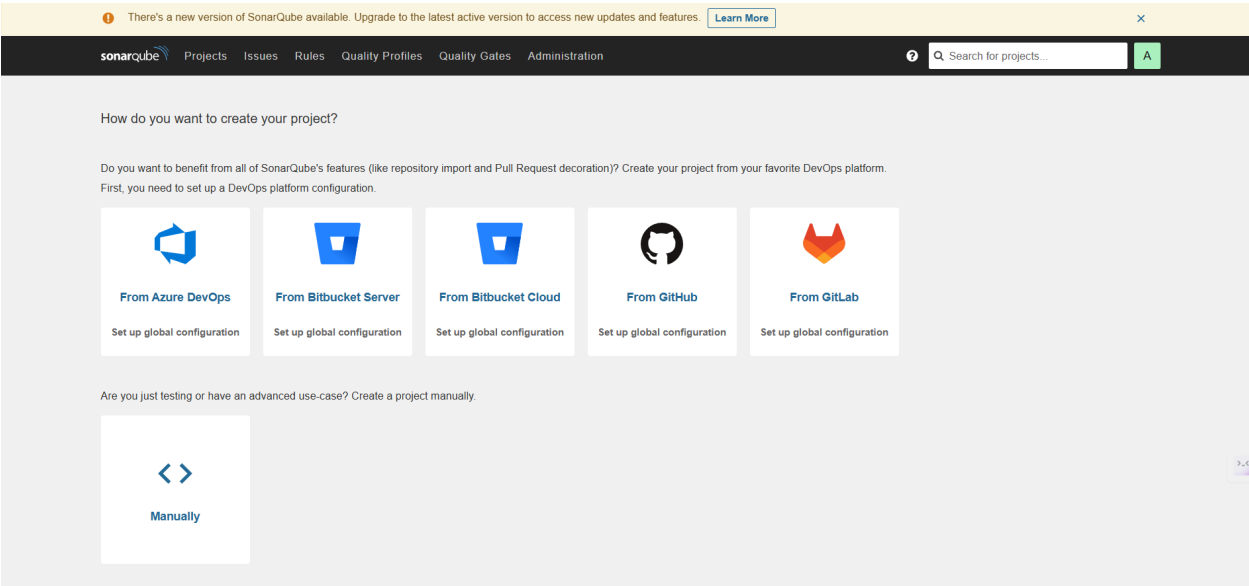
<EC2 Public IP Address:9000>



Enter username and password, click on login and change password



Update New password, This is Sonar Dashboard.



Step 3 — Install Plugins like gitlab, docker, Sonarqube Scanner, and more in Jenkins.

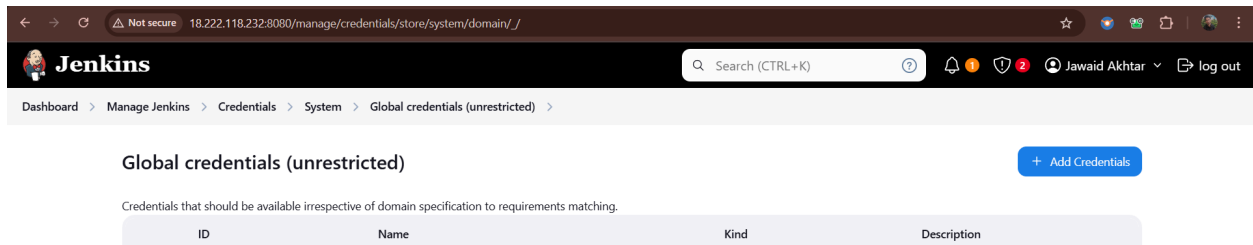
3A — Install Plugins

Go to Manage Jenkins → Plugins → Available Plugins →

Install below plugins (Install with restart)

→ GitLab, Generic WebHook Trigger, Docker, Docker Commons, Docker Pipeline, Docker API, docker-build-step, stageview pipeline, prometheus metrics, email extention templates, SonarQube Scanner, Kubernetes

Now go to jenkins dashboard ➡ manage jenkins ➡ credentials ➡ system ➡ global credentials ➡ add credentials. Use username and access token.



4b- Dockerhub credentials setup

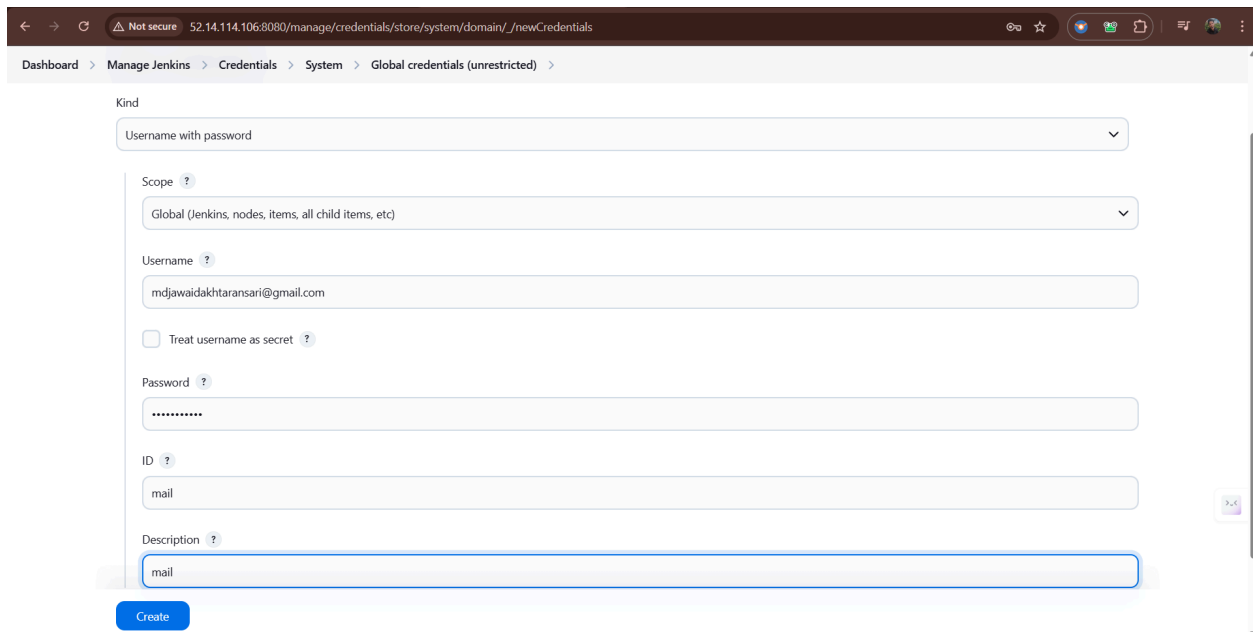
Now go to jenkins dashboard ➡ manage jenkins ➡ credentials ➡ system ➡ global credentials ➡ add credentials. Use username and access token.

4c- SonarQube credentials setup

Now go to jenkins dashboard ➡ manage jenkins ➡ credentials ➡ system ➡ global credentials ➡ add credentials. Use username and access token.

4d- Email credentials setup

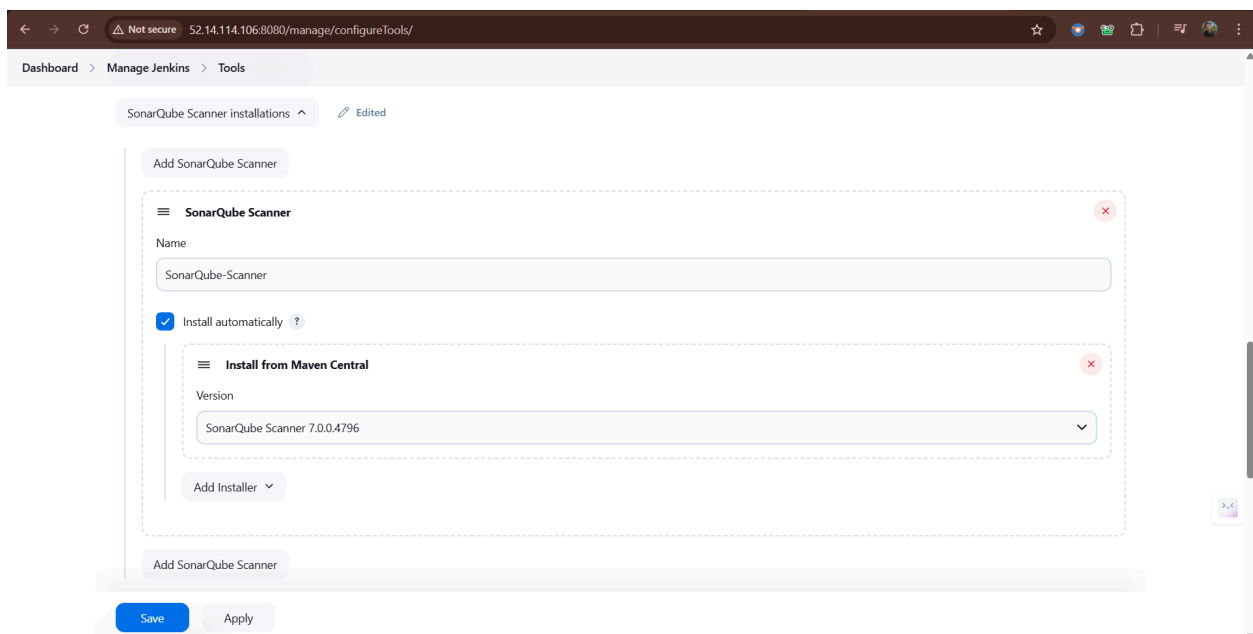
Now go to jenkins dashboard ➡ manage jenkins ➡ credentials ➡ system ➡ global credentials ➡ add credentials. Use username and access token/passwords.



The screenshot shows the Jenkins 'Global credentials (unrestricted)' configuration page. The browser address bar indicates the URL is 52.14.114.106:8080/manage/credentials/store/system/domain/_/newCredentials. The breadcrumb navigation is Dashboard > Manage Jenkins > Credentials > System > Global credentials (unrestricted). The form includes the following fields: 'Kind' set to 'Username with password', 'Scope' set to 'Global (Jenkins, nodes, items, all child items, etc)', 'Username' set to 'mdjawaidakhtaransari@gmail.com', a checkbox for 'Treat username as secret' which is unchecked, 'Password' masked with dots, 'ID' set to 'mail', and 'Description' set to 'mail'. A blue 'Create' button is at the bottom left.

4e — Configure SonarQube in Global Tool Configuration

Go to Manage Jenkins → Tools → Install SonarQube → Click on Save



The screenshot shows the Jenkins 'Tools' configuration page for 'SonarQube Scanner installations'. The browser address bar shows the URL 52.14.114.106:8080/manage/configureTools/. The breadcrumb navigation is Dashboard > Manage Jenkins > Tools. The page title is 'SonarQube Scanner installations' with an 'Edited' status. There is a button to 'Add SonarQube Scanner'. A configuration card for 'SonarQube Scanner' is shown with a red close button. It contains: 'Name' set to 'SonarQube-Scanner', 'Install automatically' checked, and an 'Install from Maven Central' section with 'Version' set to 'SonarQube Scanner 7.0.0.4796'. Below the card is an 'Add Installer' button. At the bottom, there are 'Save' and 'Apply' buttons.

Step 5 — setup k8s server as jenkins agent and connecting them

5a- Go to the k8s server using ssh:

SSH into the instance using the `.pem` file:

```
ssh -i "your-key.pem" ubuntu@your-k8s-ec2-public-ip
```

Install Java on the Agent:

```
sudo apt update
```

```
sudo apt install openjdk-21-jdk -y
```

```
java -version
```

5b- Establish SSH Connection Between Jenkins and k8s

Go to jenkins server and generate SSH Key Pair

```
ssh-keygen
```

Save the key pair in the default location (`~/.ssh/id_rsa`).

Copy the Public Key to the k8s server

You can copy the public-key and paste it into the `k8s server >.ssh>authorized_key`

Test the SSH connection from the Jenkins server to the k8s server:

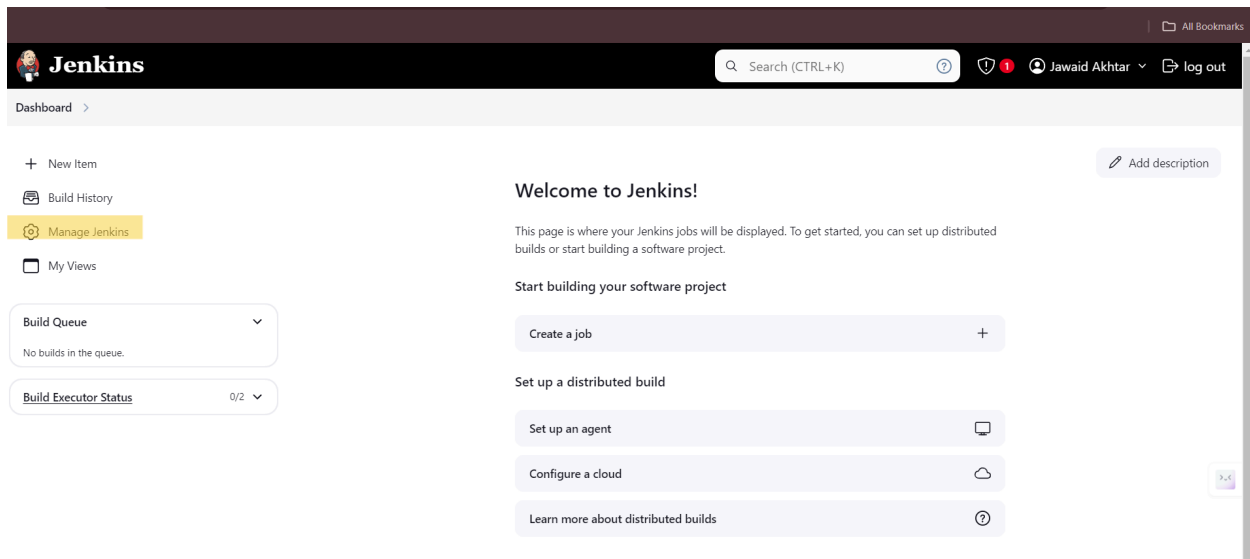
```
ssh -i <private_key> ubuntu@your-agent-ec2-public-ip
```

you should be able to SSH into the agent without being prompted for a password.

5c- Add the k8s to the Jenkins

Access Jenkins Dashboard:

Go to the Jenkins master web interface.



Navigate to Manage Jenkins > Manage Nodes and Clouds > New Node.

Nodes

S	Name	Architecture	Clock Difference	Free Disk Space	Free Swap Space	Free Temp Space	Response Time
	Built-In Node	Linux (amd64)	In sync	13.26 GiB	0 B	13.26 GiB	0ms
	Data obtained	7 min 45 sec	7 min 45 sec	7 min 45 sec	7 min 45 sec	7 min 45 sec	7 min 45 sec

Icon: S M L Legend

Provide a name for the agent, select Permanent Agent, and click OK.

New node

Node name

Type



Permanent Agent

Adds a plain, permanent agent to Jenkins. This is called "permanent" because Jenkins doesn't provide higher level of integration with these agents, such as dynamic provisioning. Select this type if no other agent types apply — for example such as when you are adding a physical computer, virtual machines managed outside Jenkins, etc.


Create

Configure the node:


Number of Executors: Set as required.

Remote root directory: Set this to `/home/ubuntu`` (or another directory on the agent).


Labels: Assign a label (e.g., `k8s``).

NAME 


jenkins_agent

Description 


Plain text [Preview](#)

Number of executors 

1

Remote root directory 

/home/ubuntu

Labels 

K8s

Launch Method: Choose Launch agent via SSH.

Launch method ?

Launch agents via SSH

Host ?

34.13.78.65

Credentials ?

- none -

+ Add ▾

! The selected credentials cannot be found

Host: Enter the public IP of the agent EC2 instance.

Credentials: Add the SSH credentials you created earlier.(username and private_key)

Jenkins Credentials Provider: Jenkins

Add Credentials

Domain

Global credentials (unrestricted) ▾

Kind

SSH Username with private key ▾

Scope ?

Global (Jenkins, nodes, items, all child items, etc) ▾

ID ?

agent-cred-um

Description ?

this is for agent-jenkins

Username

☐ Treat username as secret ?

Private Key

☒ Enter directly

Key

Enter New Secret Below

yourprivatekey

Passphrase

Host Key Verification Strategy: Choose a strategy (e.g., Non-Verifying Verification Strategy).

Click Save.

Launch method ?

Launch agents via SSH

Host ?

34.13.78.65

Credentials ?

jenkins (this is for agent-jenkins)

+ Add ▾

Host Key Verification Strategy ?

Non verifying Verification Strategy

Advanced ▾

Availability ?

Save

The Jenkins master will attempt to connect to the agent. Check the status under Manage Jenkins ➡ Manage Nodes and Clouds.

The agent should show as Connected.

Nodes

+ New Node

Configure Monitors



S	Name ↓	Architecture	Clock Difference	Free Disk Space	Free Swap Space	Free Temp Space	Response Time
	Built-In Node	Linux (amd64)	In sync	13.26 GiB	0 B	13.26 GiB	0ms
	jenkins_agent		N/A	N/A	N/A	N/A	N/A
Data obtained		3 ms	15 min	2 ms	1 ms	0 ms	15 min

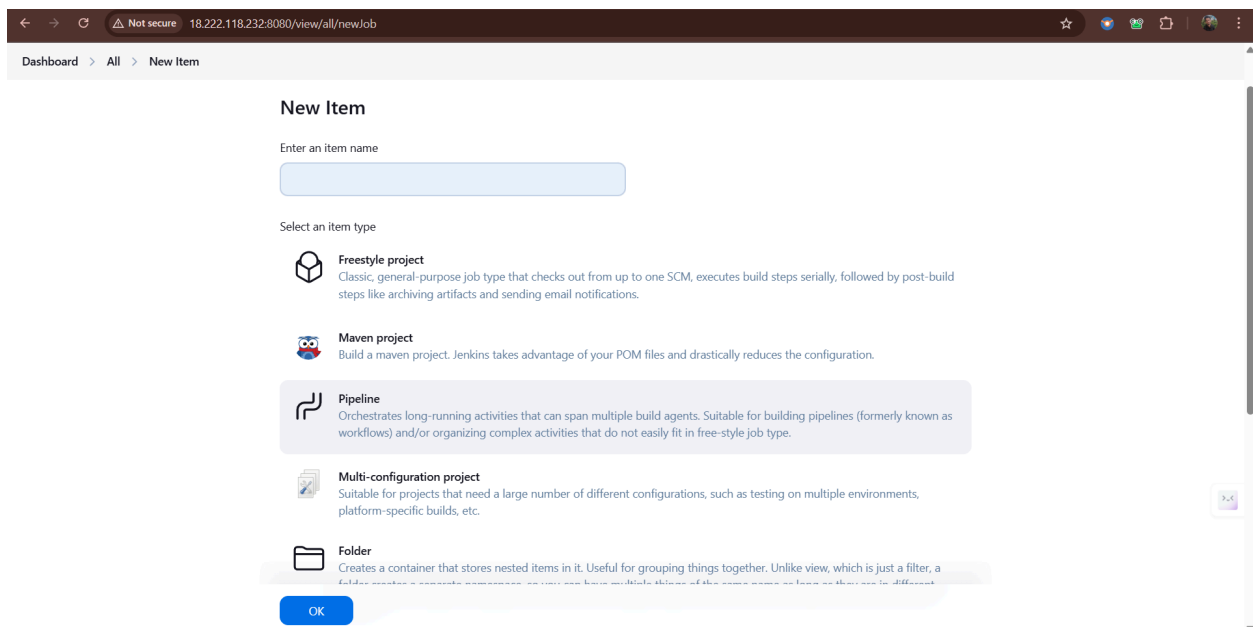
Icon: S M L

Legend

Step 6 — Create a Pipeline Project in Jenkins using a Declarative Pipeline

Now we will create a Jenkins complete pipeline for deploying our application.

Go to Jenkins dashboard and click on new item, give a name for your pipeline and select pipeline option and then click ok.



Now add below pipeline into script

```
pipeline {
  agent { label 'k8s' }

  environment {
    GIT_REPO_SSH = "git@gitlab.com:mdjawaidakhtaransari/php-crud-api.git"
    NAMESPACE = "staging"
    SCANNER_HOME = tool 'sonar-scanner'
  }

  stages {
    stage('Checkout') {
      steps {
        checkout([$class: 'GitSCM',
          branches: [[name: 'staging']],
```

```

        userRemoteConfigs: [[
            url: "${env.GIT_REPO_SSH}",
            credentialsId: 'gitlab-cred'
        ]]
    })
    sh 'git checkout -B staging origin/staging'
}
}
stage('SonarQube Analysis') {
    steps {
        withSonarQubeEnv('sonar-scanner') {
            sh '${SCANNER_HOME}/bin/sonar-scanner -Dsonar.projectKey=php-
crud-api -Dso
            r.sources=. -Dsonar.host.url=http://3.145.204.121:9000'}
        }
    }
}

stage('Docker Build') {
    steps {
        script {
            sh "docker build -t php-crud-api:latest ."
        }
    }
}

stage('Trivy Image Scan') {
    steps {
        sh "trivy image php-crud-api:latest || true"
    }
}

stage('Docker Login & Push') {
    steps {
        withCredentials([usernamePassword(credentialsId: 'DockerHubCred',
username
        ariable: 'DOCKER_USER', passwordVariable: 'DOCKER_PASS')]) {
            sh """
            echo "${DOCKER_PASS}" | docker login -u "${DOCKER_USER}" --p

```

```

password-stdin
    docker tag php-crud-api:latest ${DOCKER_USER}/php-crud-api:late
st
    docker push ${DOCKER_USER}/php-crud-api:latest
    """
    }
    }
    }
stage('Deploy to Staging with Helm') {
    steps {
        sh """
            helm upgrade --install php-crud-api ./helm -f ./values.yaml --name
space
            $NAMESPACE
            """
        }
    }
stage('Notify via Email') {
    steps {
        mail to: 'mdjawaidakhtaransari@gmail.com',
        subject: "✅ Jenkins Build #${env.BUILD_NUMBER} Successful - php
-crud-api
        ",
        body: "Build ${env.BUILD_NUMBER} completed successfully.\nImag
e: jawaid36
        5/php-crud-api:latest\nNamespace: ${NAMESPACE}"
    }
}
post {
    failure {
        mail to: 'mdjawaidakhtaransari@gmail.com',
        subject: "❌ Jenkins Build #${env.BUILD_NUMBER} Failed - php-crud-
api",
        body: "Build failed at stage: ${env.STAGE_NAME}\nCheck Jenkins logs
for detai

```



```
    ls."  
  }  
}  
}
```

step 7 — setup mysql database on second server

7a- Connect to your mysql server but with private ip

```
ssh -i <your-key.pem> ubuntu@<private-ip>
```

Now run below command to install and configure mysql database

```
sudo apt update  
sudo apt install mysql-server -y  
sudo systemctl restart mysql  
sudo mysql_secure_installation
```

7b- Create a new user and database

```
CREATE DATABASE todo;  
CREATE USER 'admin' IDENTIFIED BY 'your-password';  
GRANT ALL PRIVILEGES ON todo.* TO 'admin';  
USE DATABASE todo;  
CREATE TABLE tasks;  
FLUSH PRIVILEGES;  
EXIT;
```

```
INSERT INTO tasks (title, description, completed)  
VALUES  
( 'Complete Terraform Project', 'Finish writing the Terraform scripts', 0),  
( 'Deploy Web App', 'Deploy the Python app on EC2 instance', 0),  
( 'Write Blog Post', 'Document the Terraform workspace process', 1);
```

7c- Allow Remote Access

Edit MySQL config file:

```
sudo nano /etc/mysql/mysql.conf.d/mysqld.c
```

Change `bind-address = 127.0.0.1` to `bind-address = 0.0.0.0`

Restart MySQL

```
sudo systemctl restart mysql
```

Step 8 — Create aws EKS cluster

for creating EKS cluster we need below tools to be installed and configured, So let's start installing one by one

8a- Install Docker

Run below command to install docker.

```
# Run the following command to uninstall all conflicting packages:
for pkg in docker.io docker-doc docker-compose docker-compose-v2 podman-docker containerd runc; do sudo apt-get remove $pkg; done
```

```
# Add Docker's official GPG key:
```

```
sudo apt-get update
```

```
sudo apt-get install ca-certificates curl
```

```
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
```

```
# Add the repository to Apt sources:
```

```
echo \
```

```
"deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.asc] https://download.docker.com/linux/ubuntu \
```

```
$(. /etc/os-release && echo "${UBUNTU_CODENAME:-$VERSION_CODENAME}") stable" | \
sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
sudo apt-get update

# To install the latest version, run:
sudo apt-get install docker-ce docker-ce-cli containerd.io docker-buildx-plugin
docker-compose-plugin
docker --version
sudo usermod -aG docker $USER && newgrp docker #my case is ubuntu
```

8b- Install Trivy

```
#Add repository to /etc/apt/sources.list.d
sudo apt-get install wget apt-transport-https gnupg lsb-release
wget -qO - https://aquasecurity.github.io/trivy-repo/deb/public.key | sudo apt-key
echo deb https://aquasecurity.github.io/trivy-repo/deb $(lsb_release -sc) main |
sudo apt-get update
sudo apt-get install trivy
```

8c- Install java

```
#Installation of JAVA
sudo apt update
sudo apt install fontconfig openjdk-21-jre
java -version
```

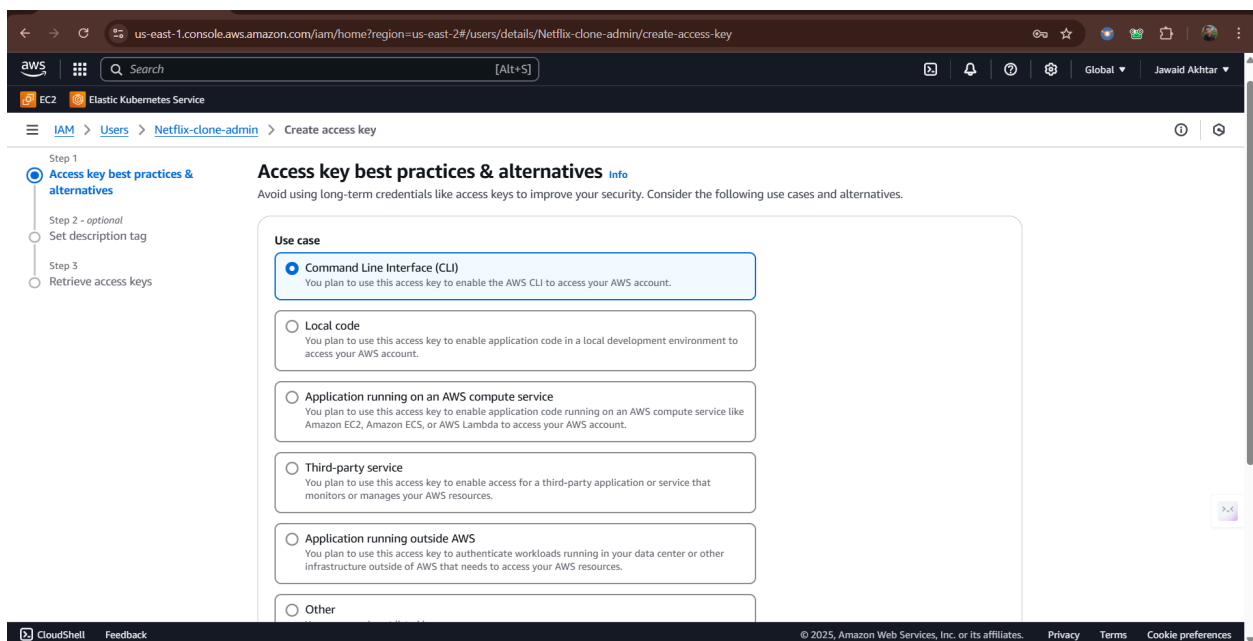
8d- Install aws cli and configure it

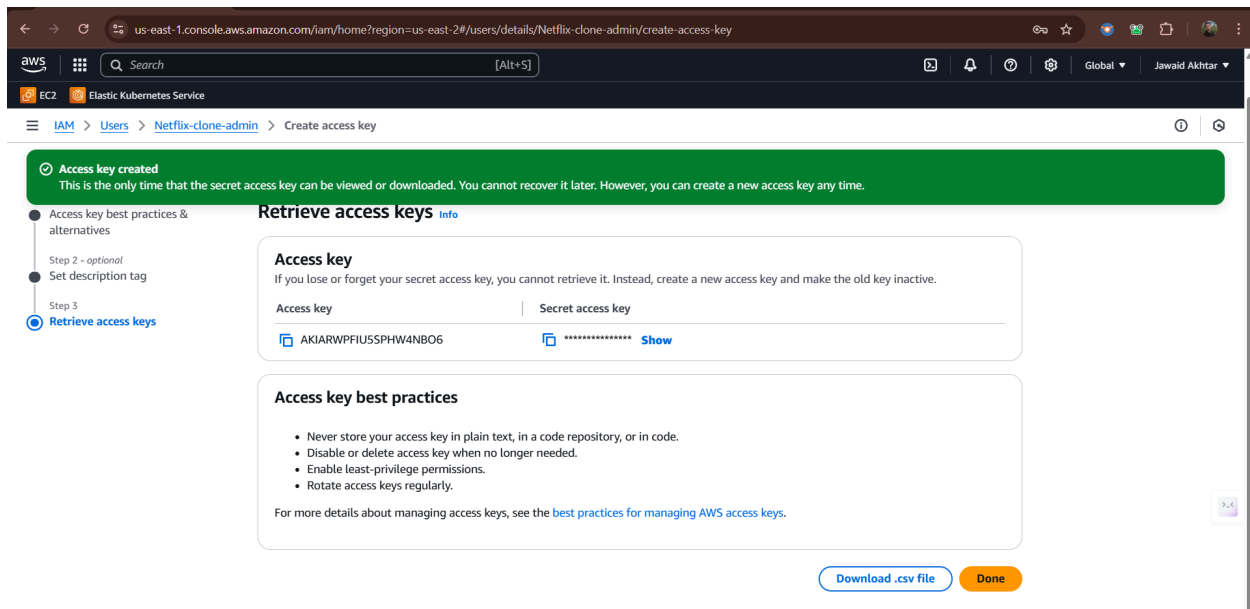
```
#!/bin/bash
#Downloading aws-cli zip file
echo "Downloading AWS_CLI..."
curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o "awscliv2.zip"
#Installing unzip tool
echo "Installing unzip tool..."
sudo apt install unzip
```

```
#Unzip AWS_CLI zip file
unzip awscliv2.zip
#Installing AWS_CLI
echo "Installing AWS_CLI..."
sudo ./aws/install
```

Now create access key for user.

Go to IAM → Users → your user name → Create access key → follow the steps shown as below





Now we will configure aws cli using access key and secret key

Run below command

```
aws configure
```

```
AWS Access Key ID [*****YEAC]: AKIARWPFIU5SPHW4NBO6 #Repl
```

```
AWS Secret Access Key [*****VTJY]: #Enter you secret ac
```

```
Default region name [us-east-2]: ap-south-1
```

```
Default output format [json]: json
```

8e- Install kubectl

```
#!/bin/bash
```

```
#Download the latest release with the command:
```

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

```
#Download the kubectl checksum file:
```

```
curl -LO "https://dl.k8s.io/release/$(curl -L -s https://dl.k8s.io/release/stable.t
xt)/bin/linux/amd64/kubectl.sha256"
```

```
#Validate the kubectl binary against the checksum file:
echo "$(cat kubectl.sha256) kubectl" | sha256sum --check
```

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

```
#Test to ensure the version you installed is up-to-date:
kubectl version --client
```

8f — install eksctl

```
curl -sLO "https://github.com/eksctl-io/eksctl/releases/latest/download/eksctl_
_$PLATFORM.tar.gz"
```

```
# (Optional) Verify checksum
curl -sL "https://github.com/eksctl-io/eksctl/releases/latest/download/eksctl_
checksums. xt" | grep $PLATFORM | sha256sum --check
```

```
tar -xzf eksctl_$PLATFORM.tar.gz -C /tmp && rm eksctl_$PLATFORM.tar.gz
```

```
sudo mv /tmp/eksctl /usr/local/bin
eksctl version
```

Now we have all required tools for creating EKS cluster on AWS.
now run below command to create EKS cluster.

```
eksctl create cluster --name php-todo-cluster --region ap-south-1
```

Step9 — Install Helm for kubernetes

Now we will install helm for simplifying deployment process and it also reduce deployment time.

For installing helm run below command on k8s server.

```
#!/bin/bash
curl https://baltocdn.com/helm/signing.asc | gpg --dearmor | sudo tee /usr/share/keyrings/helm.gpg > /dev/null
sudo apt-get install apt-transport-https --yes
echo "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/helm.gpg] https://baltocdn.com/helm/stable/debian/ all main" | sudo tee /etc/apt/sources.list.d/helm-stable-debian.list
sudo apt-get update
sudo apt-get install helm
```

Step 10 — Install Prometheus and Grafana using Helm

We will use helm for deploying monitoring tools like prometheus and grafana for dashboard.

Run below command for monitoring tools setup.

```
helm repo add prometheus-community https://prometheus-community.github.io/helm-charts
helm repo update
helm install prometheus prometheus-community/kube-prometheus-stack
kubectl get deployment -n default
```

You will see prometheus and grafana deployment is running

Now Expose your Prometheus and Grafana service to access its UI

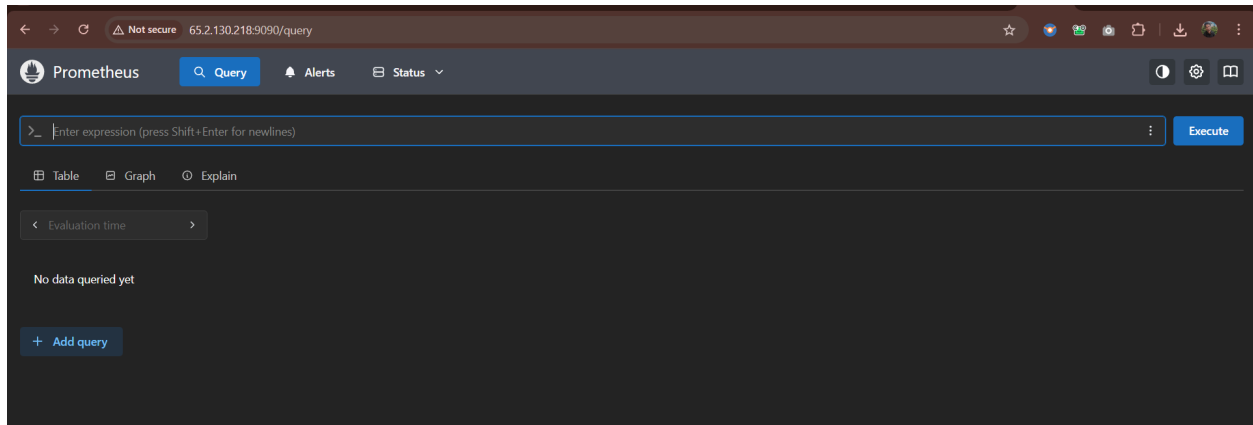
```
#Exposing prometheus service on port 9090
kubectl port-forward svc/prometheus-operated -n default 9090:9090 --address=0.0.0.0 &

#Exposing grafana service on port 8081
kubectl port-forward svc/prometheus-grafana -n default 8081:80 --address=0.0.0.0 &
```

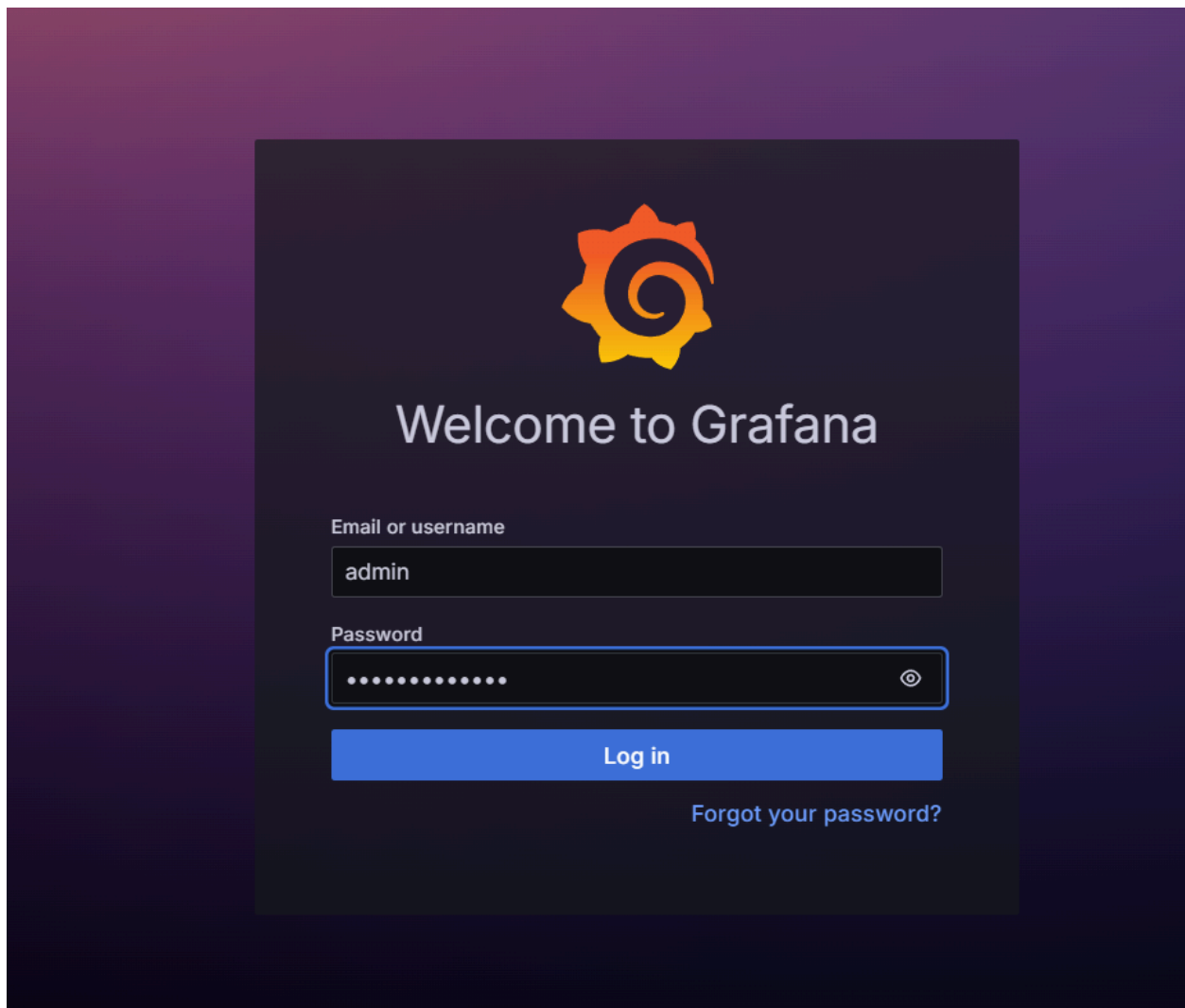
Now go to your browser and paste you public_ip following port number 9090 and you will be see login page of prometheus.

Initial username and password is admin.

After successfully login you will be asked to change initial password.

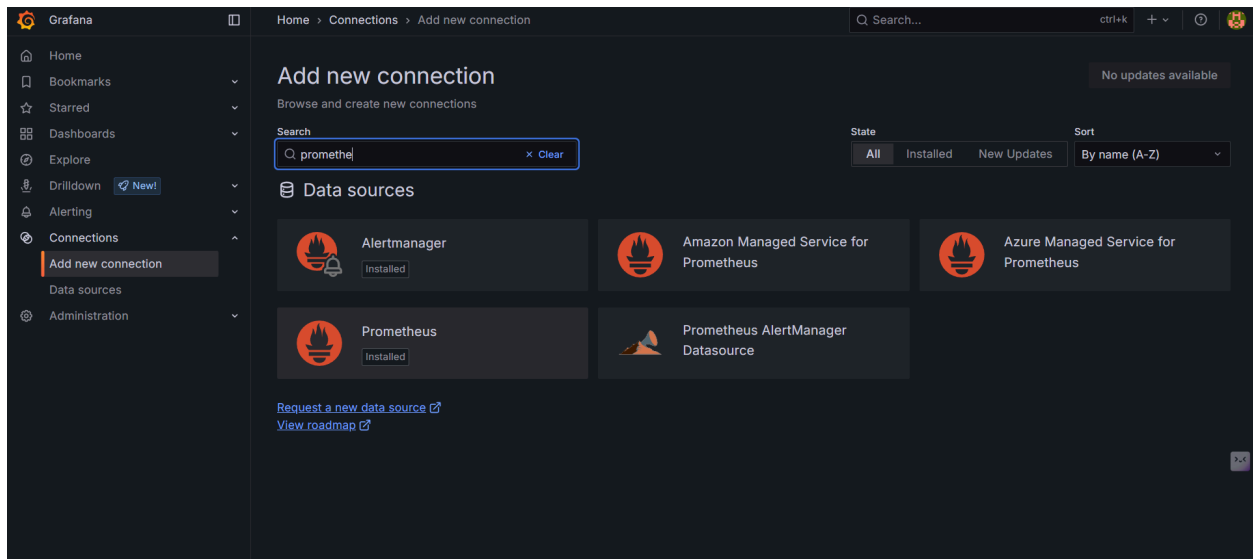


Now go to your browser and paste you public_ip following port number 8081 and you will see login page of Grafana. (Username= admin, Password= prom-operator)

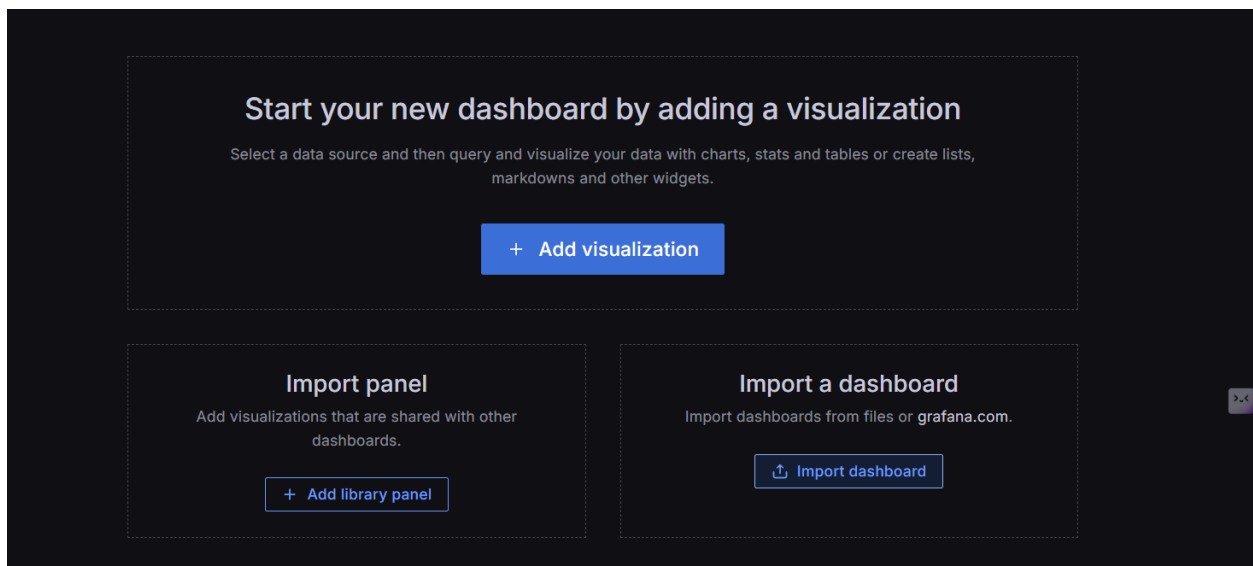


Now we will create a dashboard

Now got to connection add new data source select prometheus and provide your prometheus url and saved.



Now we are all set to create a dashboard to our grafana
Go to dashboard Import dashboard and type 1860 and click on load



Import dashboard

Import dashboard from file or Grafana.com



Upload dashboard JSON file

Drag and drop here or click to browse

Accepted file types: .json, .txt

Find and import dashboards for common applications at grafana.com/dashboards

Load

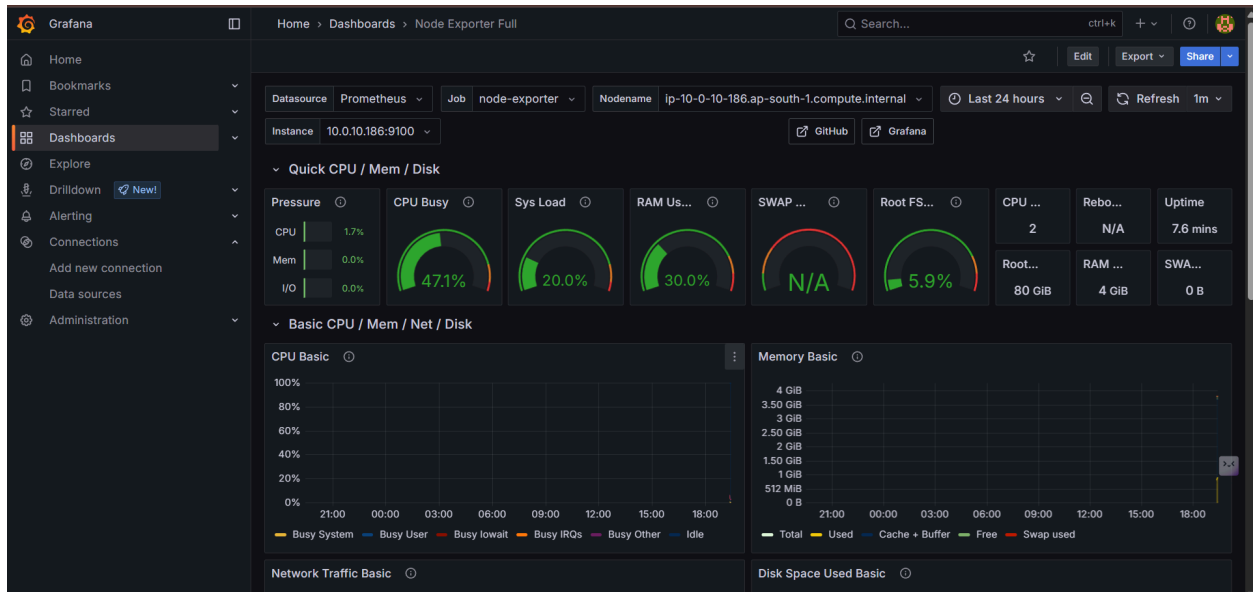
Import via dashboard JSON model

```
{
  "title": "Example - Repeating Dictionary variables",
  "uid": "_0HnEoN4z",
  "panels": [...]
  ...
}
```

Load

Cancel

You will see dashboard like below



Step 11 — Install ArgoCD using Helm

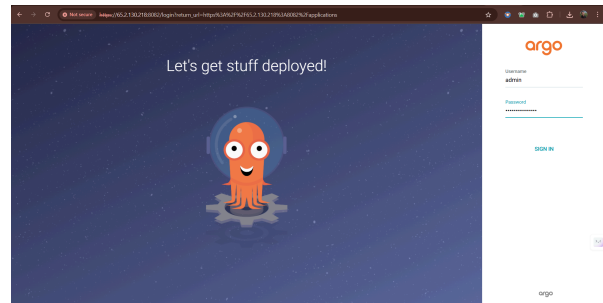
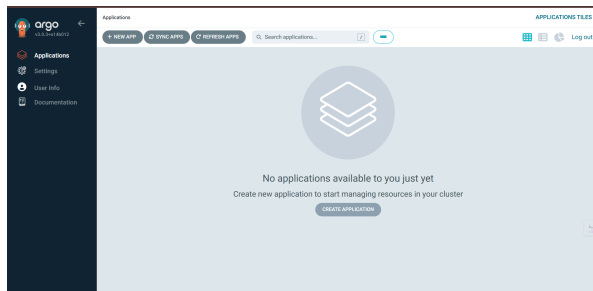
Now we will install ArgoCD using Helm to automatically deploy our application.

Run below command to install ArgoCD and access its UI.

```
helm repo add argo https://argoproj.github.io/argo-helm
helm repo update
helm install argocd argo/argo-cd
kubectl get svc -n default
kubectl port-forward svc/argocd-server -n default 8082:443 --address=0.0.0.0 &
```

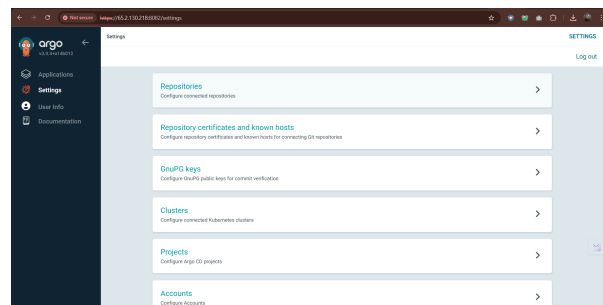
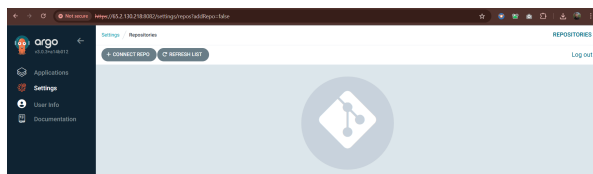
Now go to your browser and paste your public_ip following port number 8082 and you will see the login page of ArgoCD. Username is **admin** and for password run the below command to see the initial password.

```
kubectl -n default get secret argocd-initial-admin-secret -o jsonpath="{.data.password}"
```

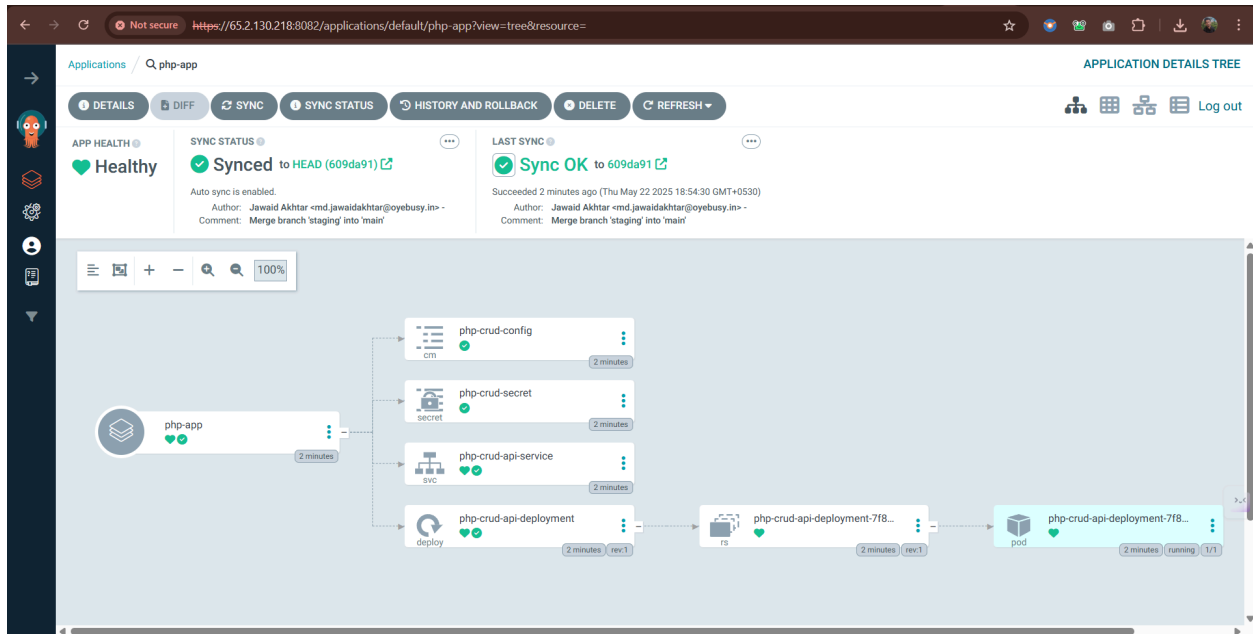


Our gitlab repo is private so we need to add our repo with credentials in argocd first

Now go to settings repositories and add repo now provide url, username and password of gitlab.

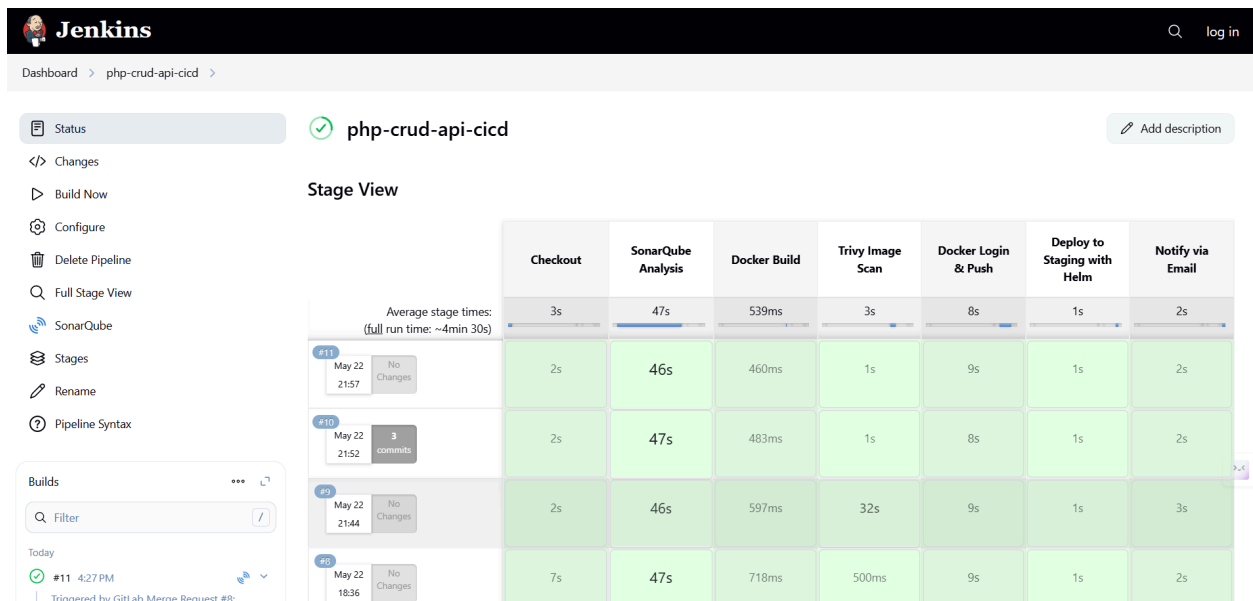


now go to application create application and fill all required details like name of app, gitlab repo url, path of manifest file, kubernetes cluster url and at the end namespace where you want to deploy your application. your application will be deployed like below.



Now push any changes into development branch and create a merge request to staging branch.

Once dev branch successfully merged into staging your webhook will trigger your pipeline and you will see below stages.



Step 12 — Access the PHP-APP on the Browser

Now Take load balancer DNS and paste on browser to access the application.

```
Not secure a8554b5104aa346aeb69ce59bf051364-1234405022.ap-south-1.elb.amazonaws.com/api.php/records/tasks
Pretty print
{"records":[{"id":1,"title":"Complete Terraform Project","description":"Finish writing the Terraform scripts","completed":false,"created_at":"2025-05-22 12:27:36"}, {"id":2,"title":"Deploy Web App","description":"Deploy the Python app on EC2 instance","completed":false,"created_at":"2025-05-22 12:27:36"}, {"id":3,"title":"Write Blog Post","description":"Document the Terraform workspace process","completed":true,"created_at":"2025-05-22 12:27:36"}]}
```