Project: Deployment of Online Ticketing and Event Management Applications and Infrastructure

By: Harshika

LinkedIn: Harshika

Portfolio: harshika-devops.online

PHASE 1 (Foundation)

- Launch 3 EC2 instances
- (Server 1 “Jenkins → Build + Docker”, Server 2 “SonarQube → Code Analysis”, Server 3 “Amazon EKS → Deployment”)
- Setup:
    - Jenkins
    - Docker
    - Trivy

PHASE 2 (Pipeline)

- Jenkins pipeline (without Kubernetes)
- Sonarqube
- Build + Push Docker image
- Deploy container

PHASE 3 (Kubernetes / EKS)

- Create cluster
- Deploy via Jenkinsfile2

PHASE 4 (Monitoring)

- Prometheus
- Grafana

Phase 1:

STEP 1: Launch FIRST EC2 Instance (BMS-Server)

This will be the main server (Jenkins + Docker)

1. Go to EC2 Dashboard

- Login to AWS
- Go to EC2
- Click “Launch Instance”

2. Name the Instance

Name: Jenkins-bms-Server

3. Ubuntu Server 22.04 LTS (HVM), SSD Volume Type

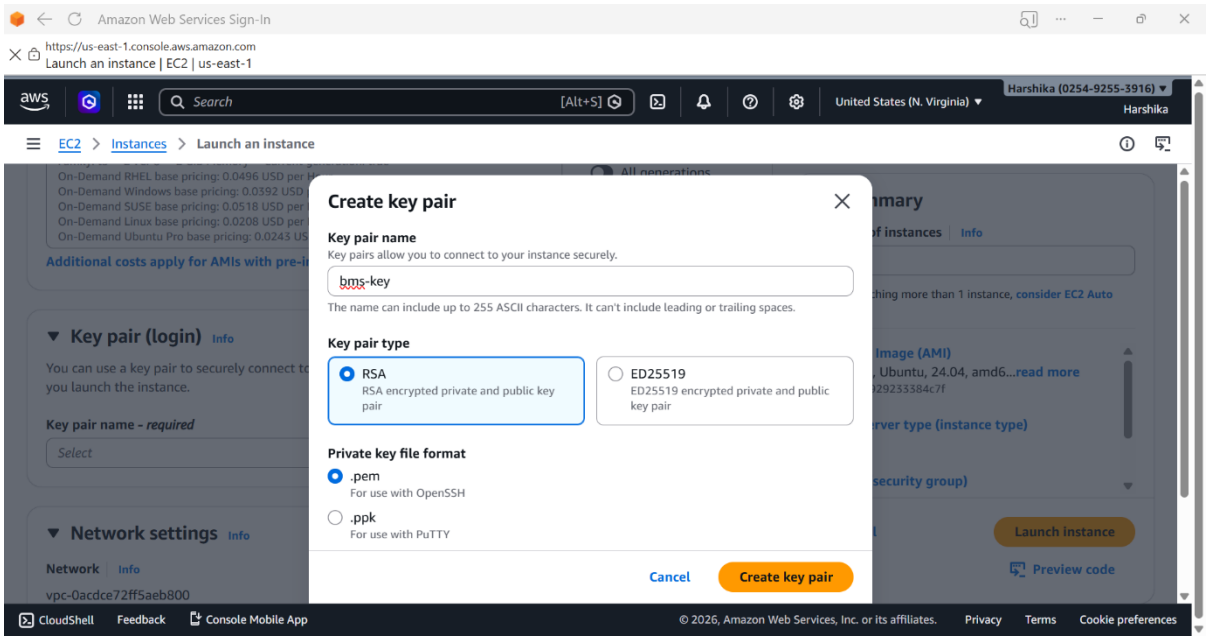
4. Choose Instance Type

t3.small

5. Under “Key pair”

- Click Create new key pair
- Name: bms-key

- Type: RSA
- Format: .pem
- Click Create



6. Network Settings (VERY IMPORTANT)

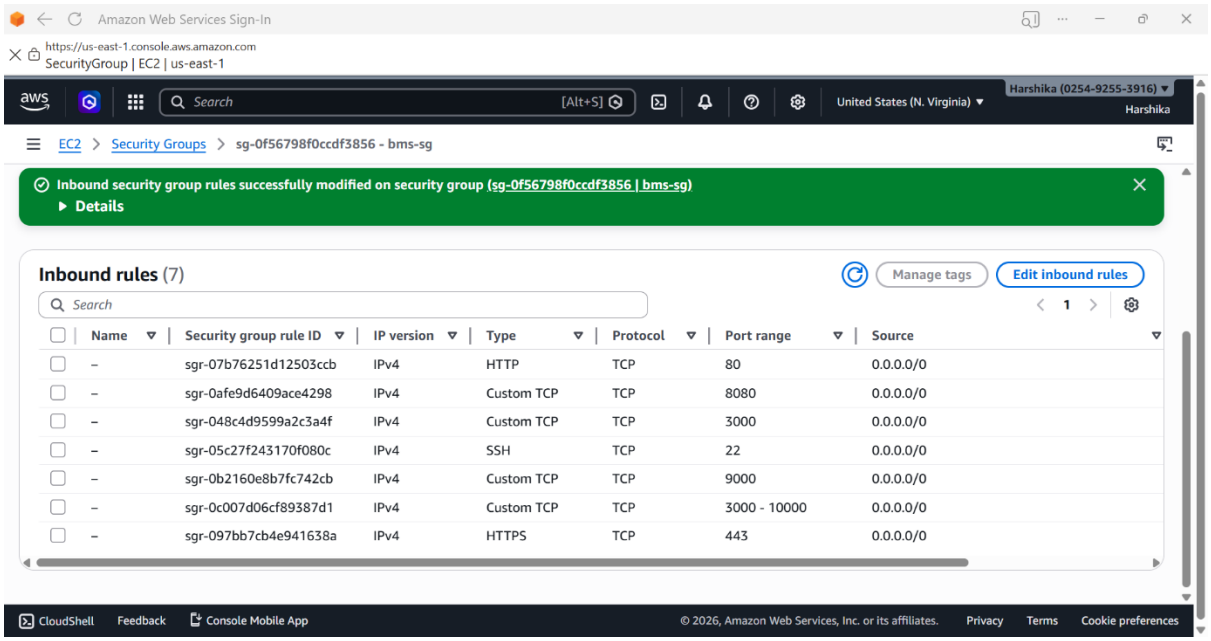
Add Inbound Rules:bms-sg

Click “Add security group rule” and add ALL below:

Type	Port	Source
SSH	22	Anywhere
HTTP	80	Anywhere
HTTPS	443	Anywhere
Custom TCP	8080	Anywhere
Custom TCP	3000	Anywhere

This is IMPORTANT for:

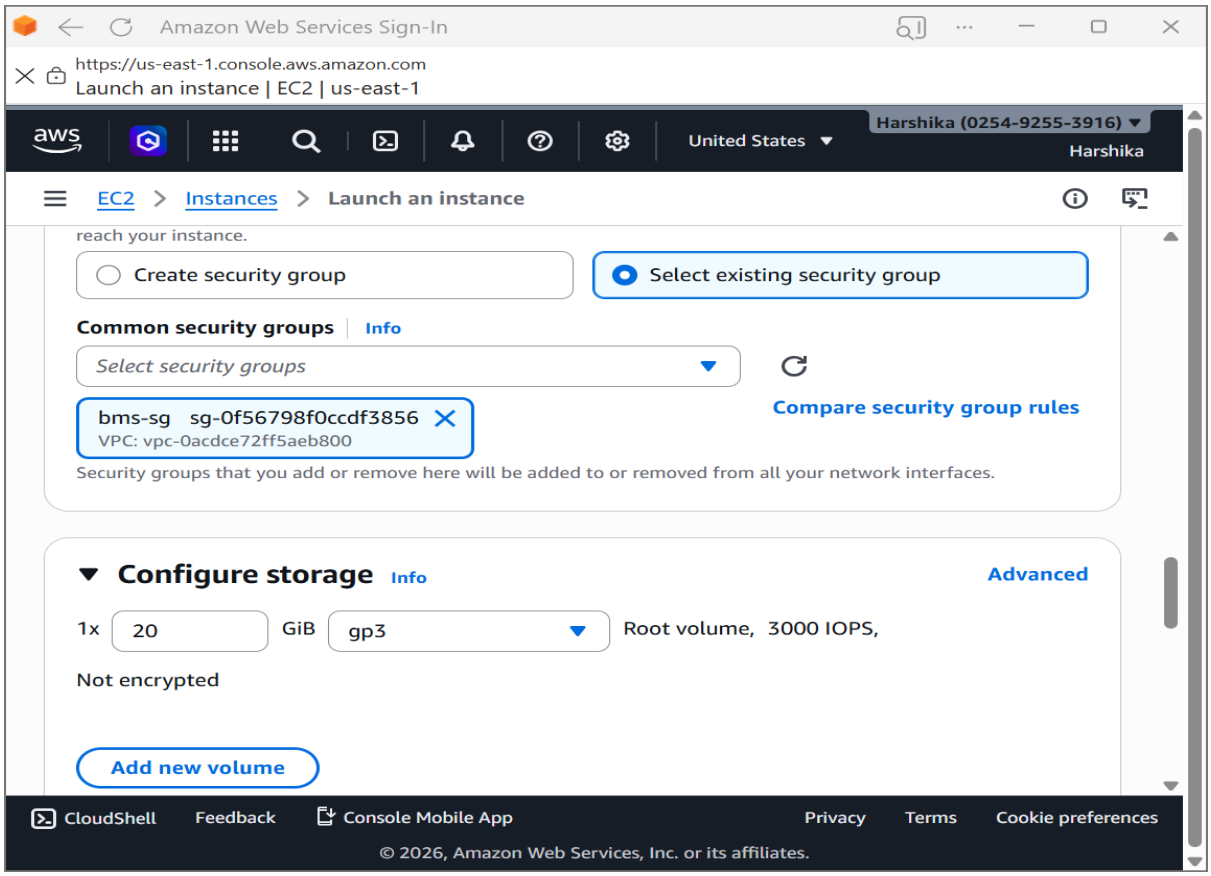
- Jenkins (8080)
- App (3000)
- Other services



7. Configure Storage

Size: 20 GB

(Default is 8 → increase it)



8. Launch Instance

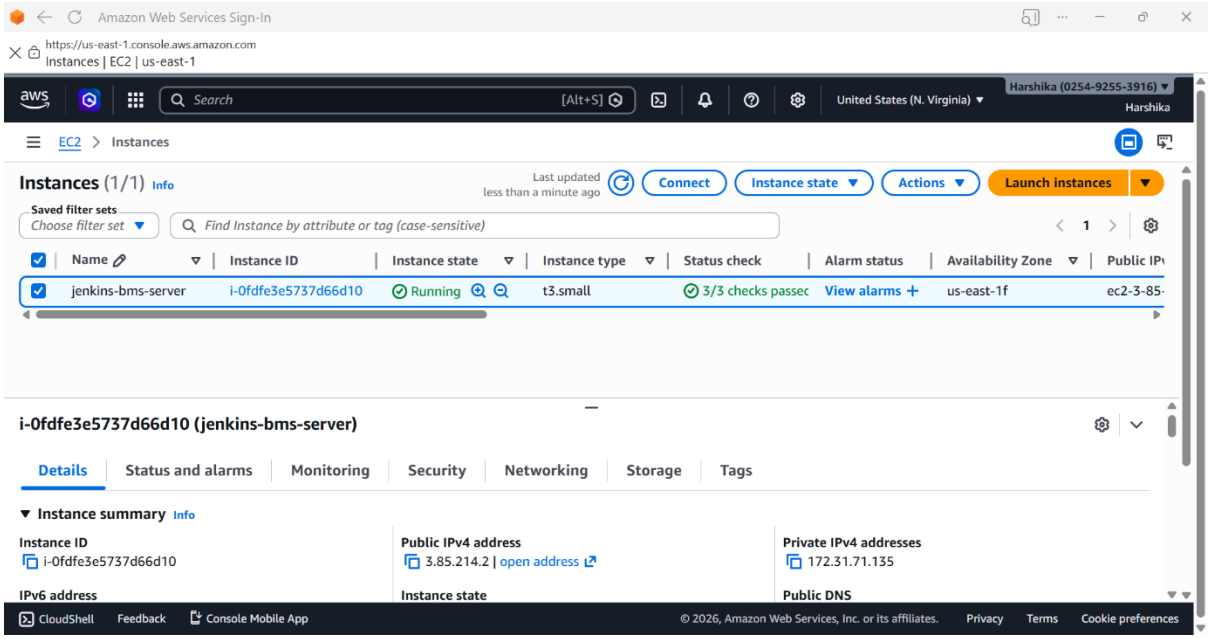
Click:
Launch Instance

9. Wait & Verify

After launching:

- Go to Instances
- Wait until:

Instance state = Running
Status check = 3/3 checks passed



STEP 2: Connect to EC2 using PowerShell

2.1 Connect to Server

Cd Downloads

Fix permission

Run this:

icacls bms-key.pem /inheritance:r
icacls bms-key.pem /grant:r "%username%:R"

This avoids SSH permission error (very common)

ssh -i bms-key.pem ubuntu@3.85.214.2

First-time prompt

You will see:

Are you sure you want to continue connecting (yes/no)?

Type:

yes

STEP 2.2: Add SWAP MEMORY (VERY IMPORTANT)

Run these commands

sudo fallocate -l 2G /swapfile

sudo chmod 600 /swapfile

sudo mkswap /swapfile

sudo swapon /swapfile

Make it permanent (IMPORTANT)

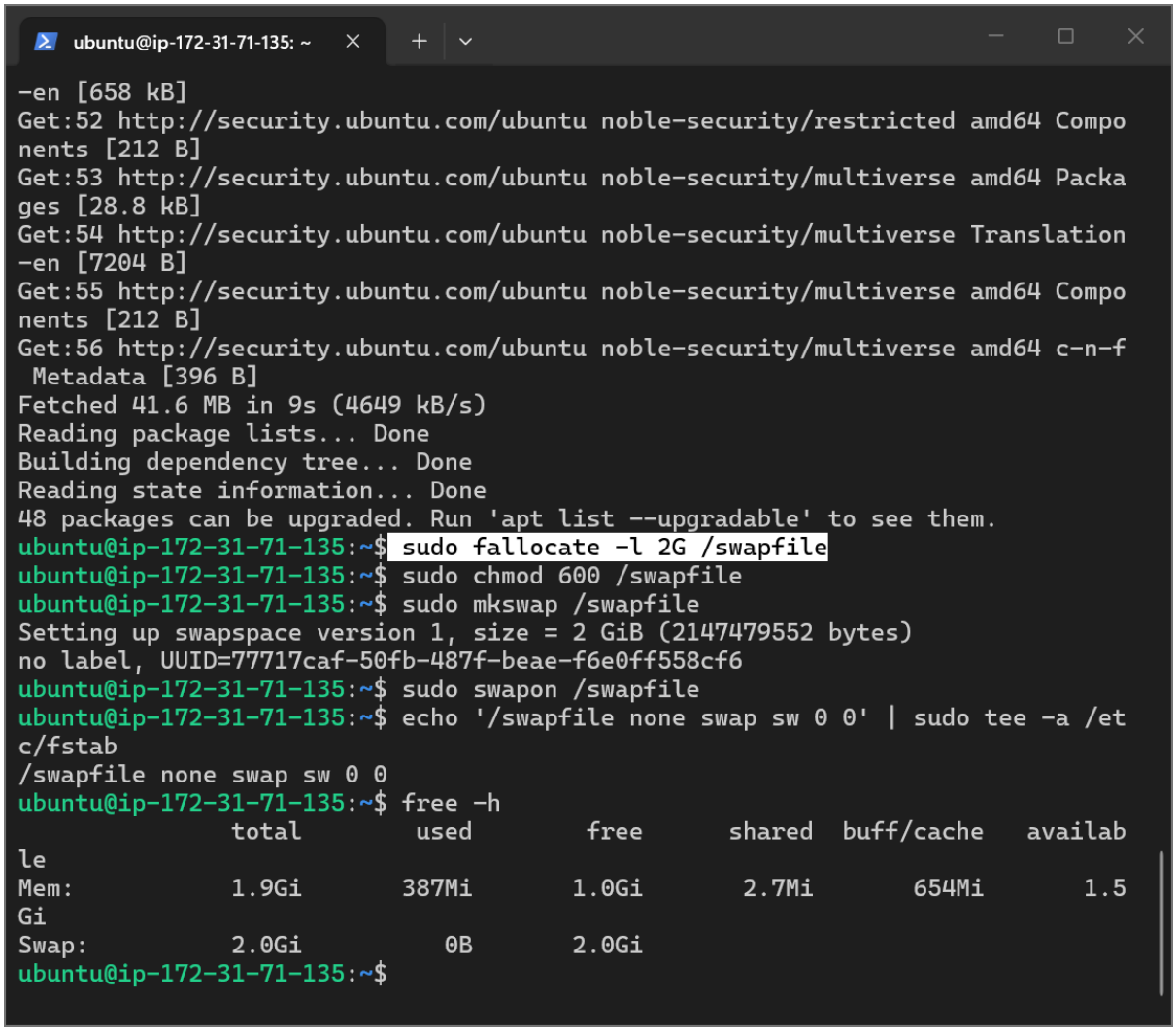
echo '/swapfile none swap sw 0 0' | sudo tee -a /etc/fstab

Verify swap

free -h

You should see something like:

Swap: 2.0Gi



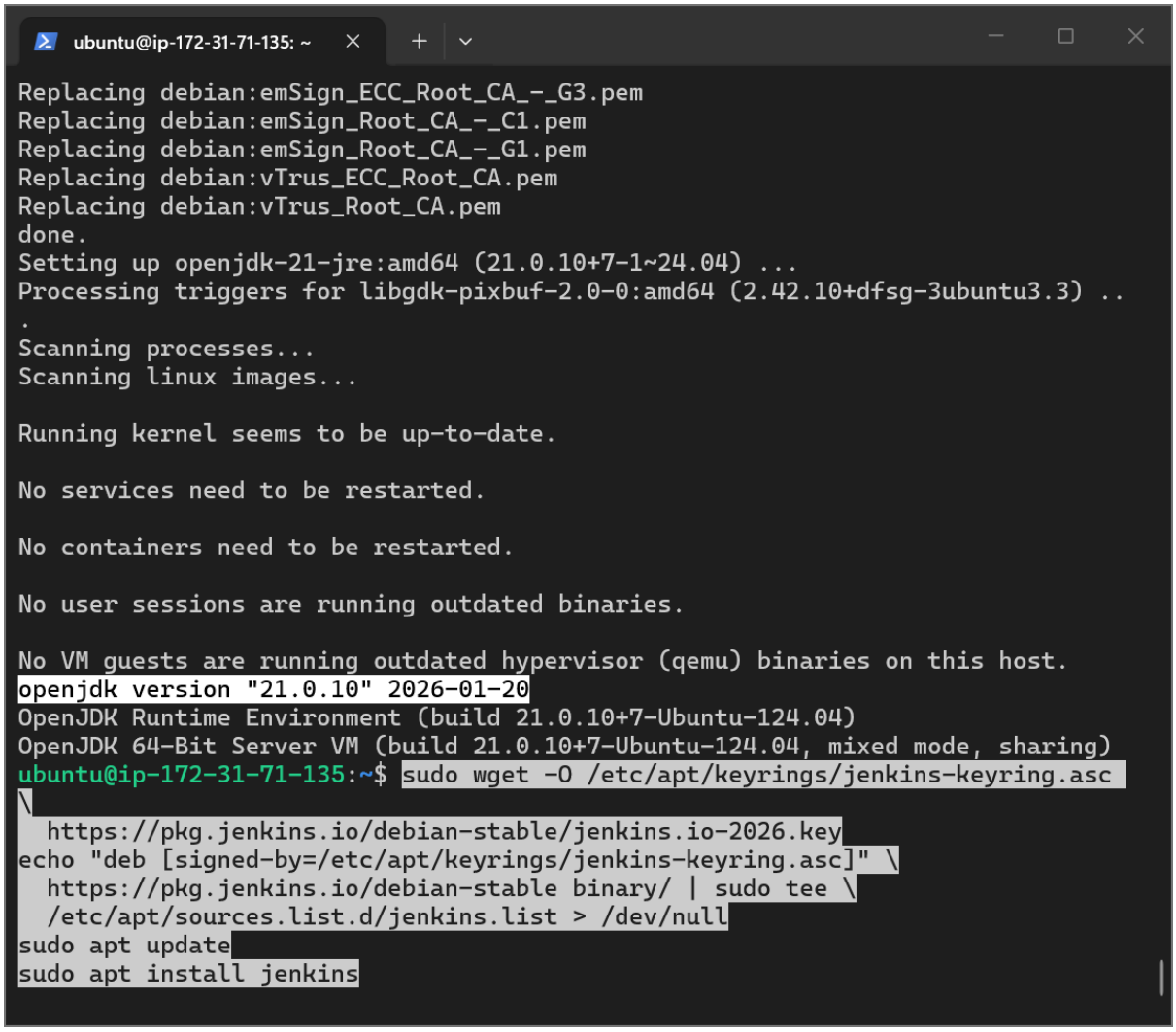
STEP 3: Install Java + Jenkins using the official link- Linux

3.1 Install Java 21

sudo apt update
sudo apt install fontconfig openjdk-21-jdk -y

3.2 Verify Java

java -version



3.3 Create keyring directory

sudo mkdir -p /etc/apt/keyrings

3.4 Add Jenkins Key

curl -fsSL https://pkg.jenkins.io/debian-stable/jenkins.io-2026.key | sudo tee /etc/apt/keyrings/jenkins-keyring.asc > /dev/null

3.5 Add Jenkins Repo

echo "deb [signed-by=/etc/apt/keyrings/jenkins-keyring.asc] https://pkg.jenkins.io/debian-stable binary/" | sudo tee /etc/apt/sources.list.d/jenkins.list > /dev/null

3.6 Update packages

sudo apt update

3.7 Install Jenkins

sudo apt install jenkins -y

3.8 Start Jenkins

sudo systemctl start jenkins
sudo systemctl enable jenkins

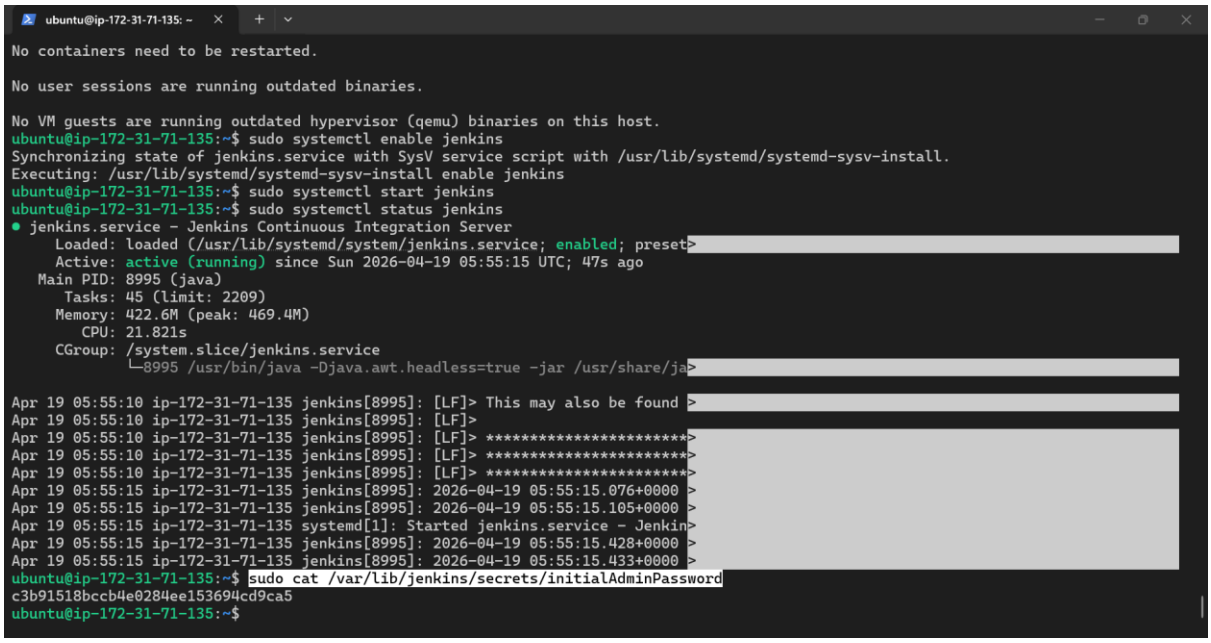
3.9 Check status

sudo systemctl status jenkins

Expect:

active (running)

Press q to exit



Unlocking Jenkins

Browse to http://localhost:8080 (or the port you configured for Jenkins during installation) and wait until the **Unlock Jenkins** page appears.

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

3.10 Plugin Installation

On your Jenkins UI:

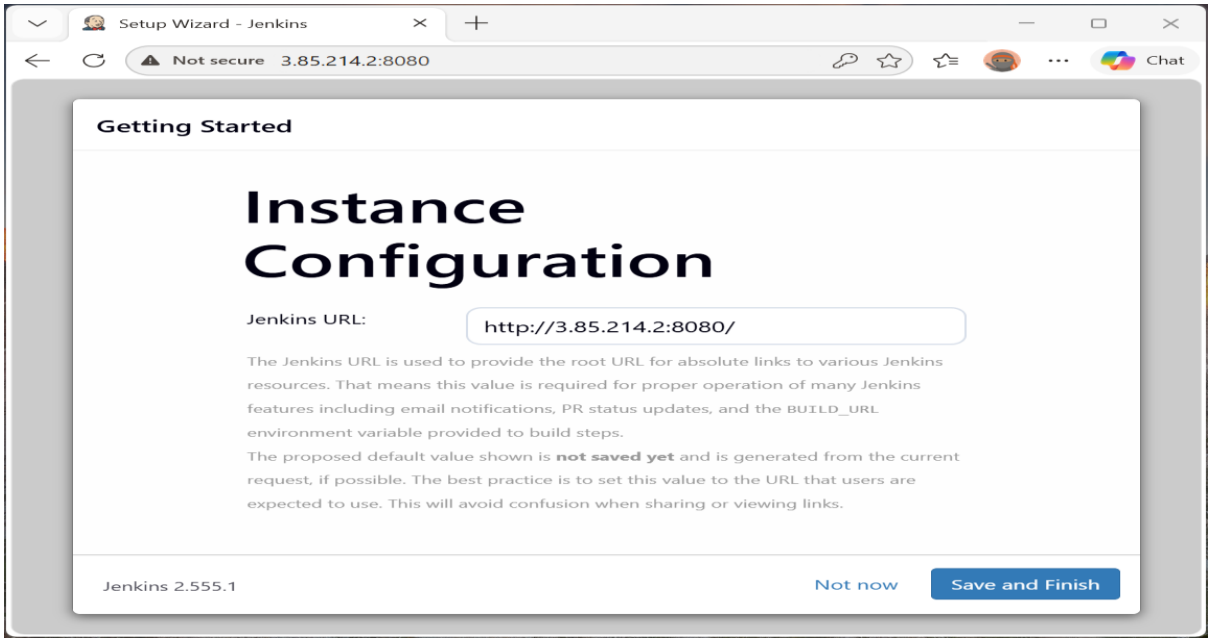
Click: **“Install suggested plugins”**

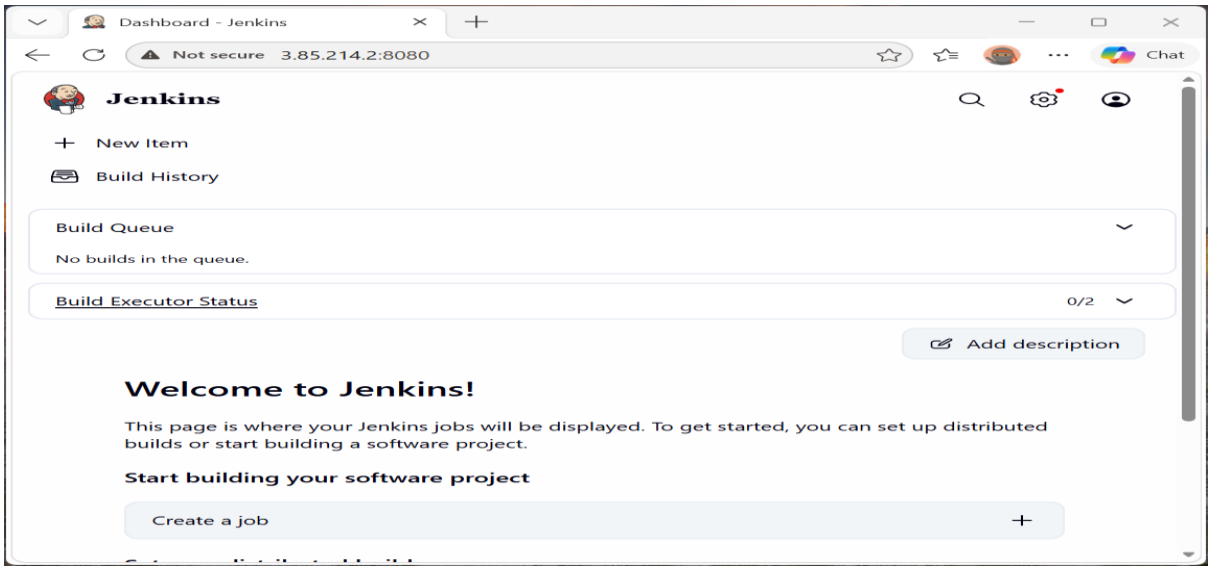
3.11 Create Admin User

Fill:

- Username → admin
- Password → (set something strong, remember it)
- Full name → anything
- Email → your email

Start Using Jenkins





Now open Jenkins and install plugins:

manage Jenkins-plugins-available plugins

Install below plugins;

Eclipse Temurin Installer,

SonarQube scanner,

NodeJS,

Docker,

Docker Commons,

Docker Pipeline,

Docker API,

docker-build-step,

OWASP dependency check,

Pipeline stage view,

Email Extension Template,

Kubernetes,

Kubernetes CLI,

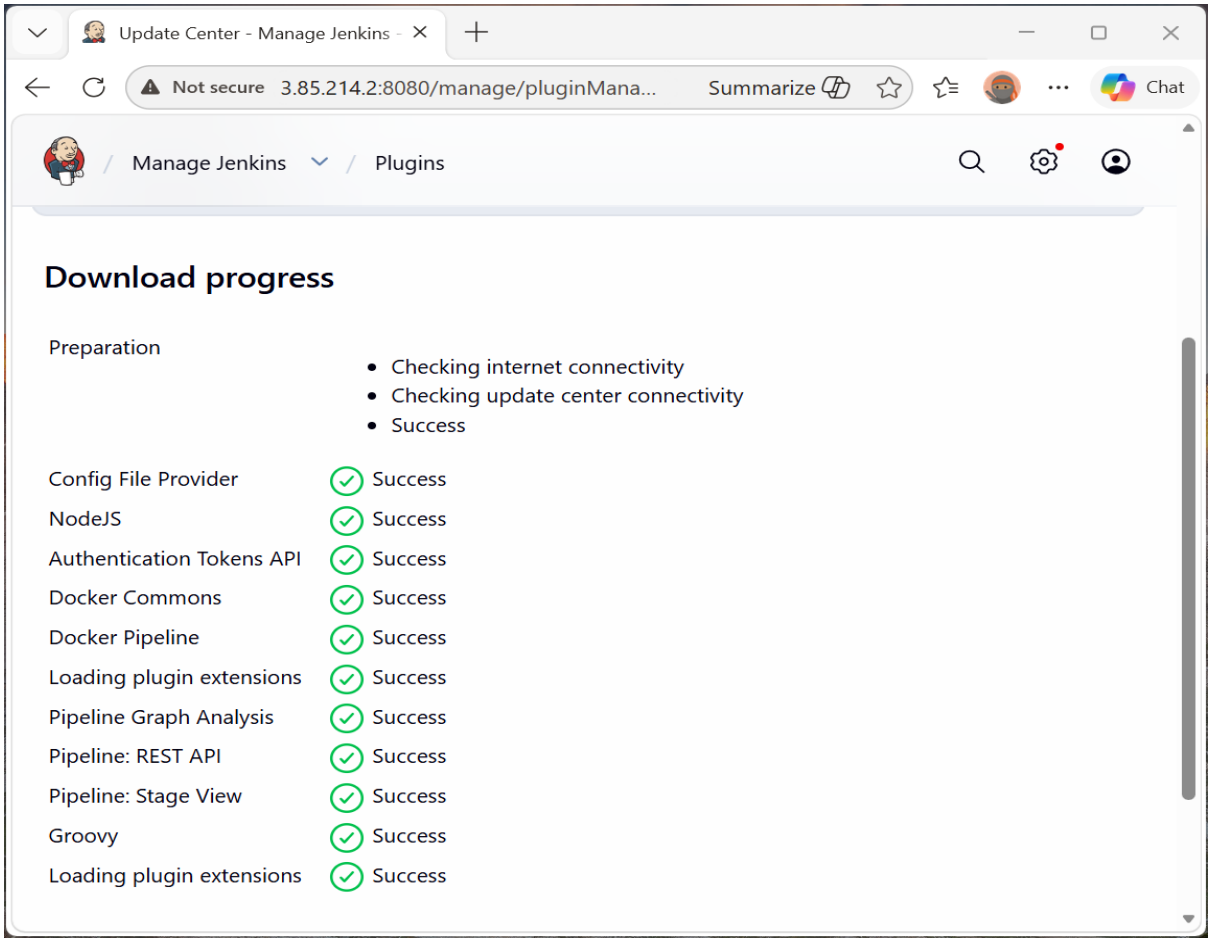
Kubernetes Client API,

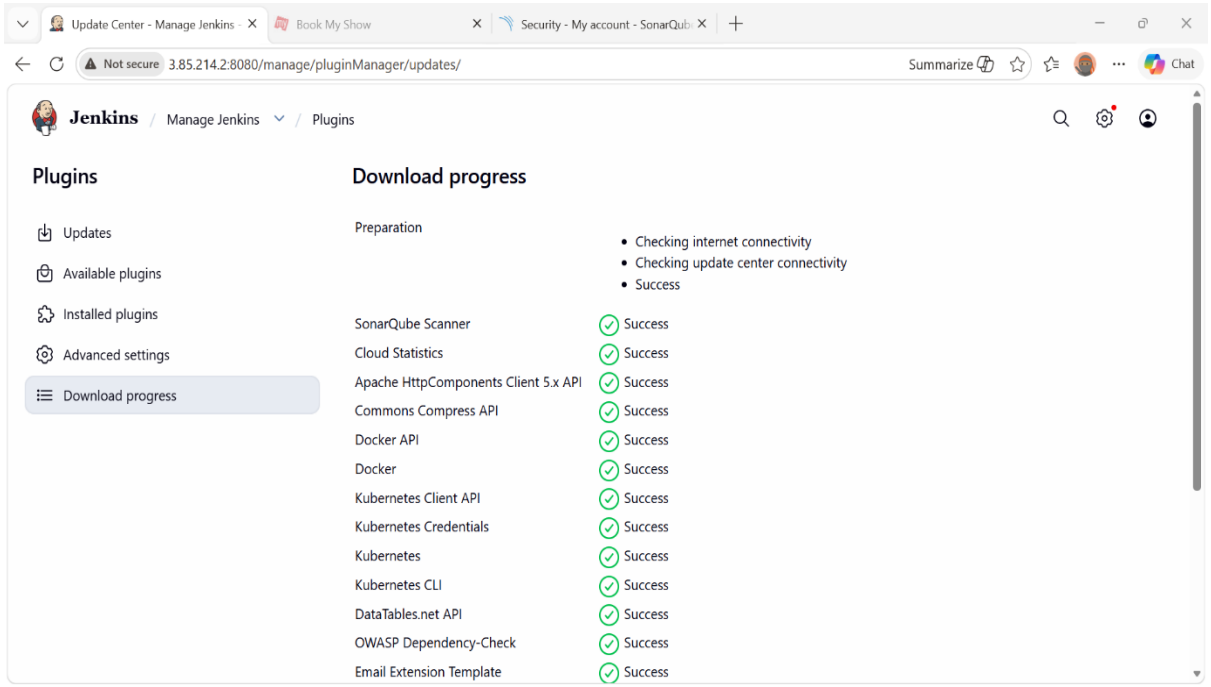
Kubernetes Creden als,

Config File Provider,

Prometheus metrics.

Prometheus metrics requires restart check the restart Jenkins box and re-login into Jenkins.


Update Center - Manage Jenkins
Not secure 3.85.214.2:8080/manage/pluginMana...
Summarize
Chat
Manage Jenkins / Plugins
Download progress
Preparation
Checking internet connectivity
Checking update center connectivity
Success
Config File Provider Success
NodeJS Success
Authentication Tokens API Success
Docker Commons Success
Docker Pipeline Success
Loading plugin extensions Success
Pipeline Graph Analysis Success
Pipeline: REST API Success
Pipeline: Stage View Success
Groovy Success
Loading plugin extensions Success


Update Center - Manage Jenkins
Book My Show
Security - My account - SonarQub
Not secure 3.85.214.2:8080/manage/pluginManager/updates/
Summarize
Chat
Jenkins / Manage Jenkins / Plugins
Plugins
Updates
Available plugins
Installed plugins
Advanced settings
Download progress
Download progress
Preparation
Checking internet connectivity
Checking update center connectivity
Success
SonarQube Scanner Success
Cloud Statistics Success
Apache HttpComponents Client 5.x API Success
Commons Compress API Success
Docker API Success
Docker Success
Kubernetes Client API Success
Kubernetes Credentials Success
Kubernetes Success
Kubernetes CLI Success
DataTables.net API Success
OWASP Dependency-Check Success
Email Extension Template Success

STEP 5: Install Git + Docker (on Jenkins server)

We'll install:

- Git → for pulling code
- Docker → for building & running apps
- Give Jenkins permission to use Docker

5.1 Install Git

sudo apt install git -y

Verify:

git --version

```
ubuntu@ip-172-31-71-135:~$ sudo apt install git -y
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
git is already the newest version (1:2.43.0-1ubuntu7.3).
git set to manually installed.
0 upgraded, 0 newly installed, 0 to remove and 48 not upgraded.
ubuntu@ip-172-31-71-135:~$ git --version
git version 2.43.0
ubuntu@ip-172-31-71-135:~$
```

5.2 Install Docker

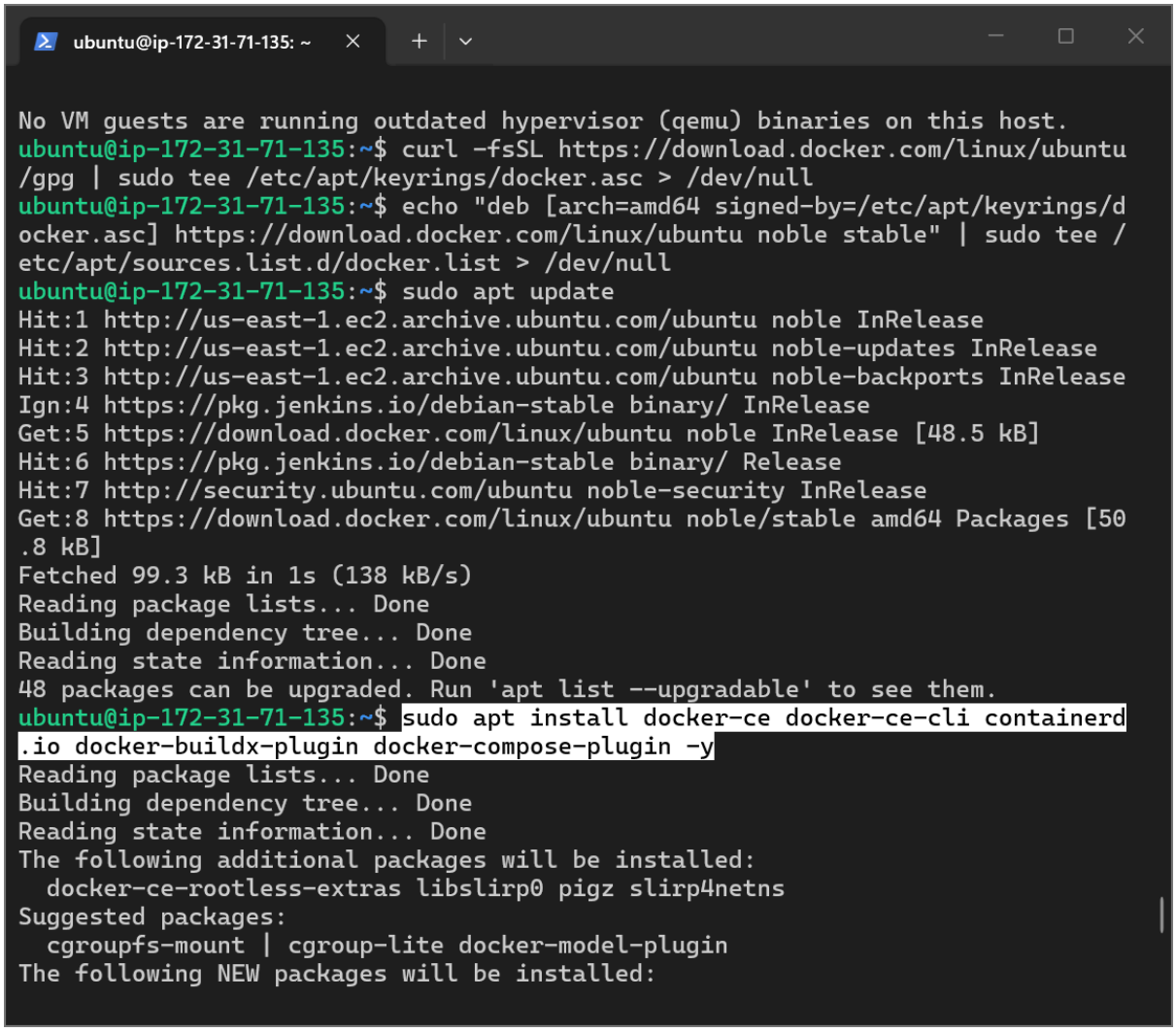
sudo apt install apt-transport-https ca-certificates curl software-properties-c

curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo tee /etc/apt/keyrings/docker.asc > /dev/null

echo "deb [arch=amd64 signed-by=/etc/apt/keyrings/docker.asc] https://download.docker.com/linux/ubuntu noble stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null

sudo apt update

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



5.3 Start Docker

sudo systemctl start docker
sudo systemctl enable docker

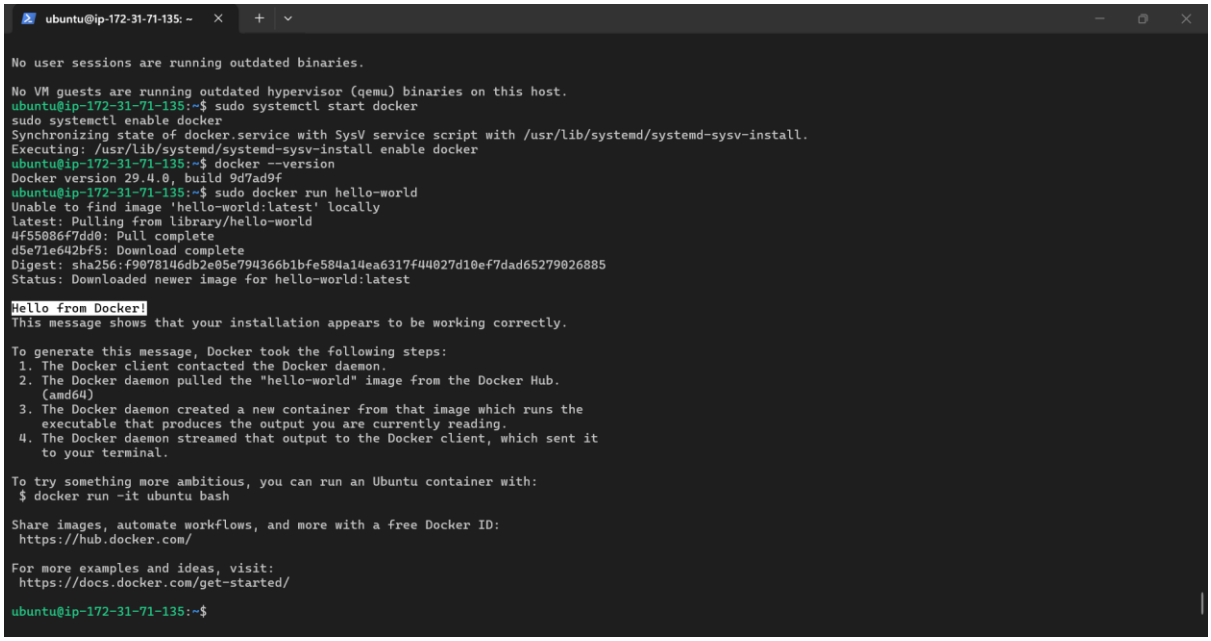
5.4 Verify Docker

docker --version

sudo docker run hello-world

Must print:

Hello from Docker!



5.5 Give Jenkins access to Docker (VERY IMPORTANT)

sudo usermod -aG docker jenkins

sudo usermod -aG docker ubuntu

```
For more examples and ideas, visit:
 https://docs.docker.com/get-started/

ubuntu@ip-172-31-71-135:~$ sudo usermod -aG docker jenkins
ubuntu@ip-172-31-71-135:~$ sudo usermod -aG docker ubuntu
ubuntu@ip-172-31-71-135:~$ sudo systemctl restart jenkins
ubuntu@ip-172-31-71-135:~$ newgrp docker
ubuntu@ip-172-31-71-135:~$
```

5.6 Restart Jenkins

sudo systemctl restart jenkins

5.7 (IMPORTANT) Reload groups for current user

newgrp docker

STEP 6 — Install Node.js + Trivy on Jenkins Server

PART A — Install Node.js

The repo (Book-My-Show) is Node-based, so:

Run this (Node 18 LTS — stable & lightweight)

curl -fsSL https://deb.nodesource.com/setup_18.x | sudo -E bash -
sudo apt install -y nodejs

Verify

node -v
npm -v

```
ubuntu@ip-172-31-71-135:~$ node -v
npm -v
v18.20.8
10.8.2
ubuntu@ip-172-31-71-135:~$ echo 'export NODE_OPTIONS="--max-old-space-siz
e=512"' >> ~/.bashrc
source ~/.bashrc
ubuntu@ip-172-31-71-135:~$
```

Set memory limit (VERY IMPORTANT)

echo 'export NODE_OPTIONS="--max-old-space-size=512"' >> ~/.bashrc
source ~/.bashrc

IMPORTANT (Memory optimisation)

Node builds can crash the t3.small if not controlled.

Set this:

export NODE_OPTIONS="--max-old-space-size=512"

// This limits RAM usage (VERY IMPORTANT)

PART B — Install Trivy (for security scan)

Run:

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

wget -qO - https://aquasecurity.github.io/trivy-repo/deb/public.key | \
gpg --dearmor | sudo tee /usr/share/keyrings/trivy.gpg > /dev/null

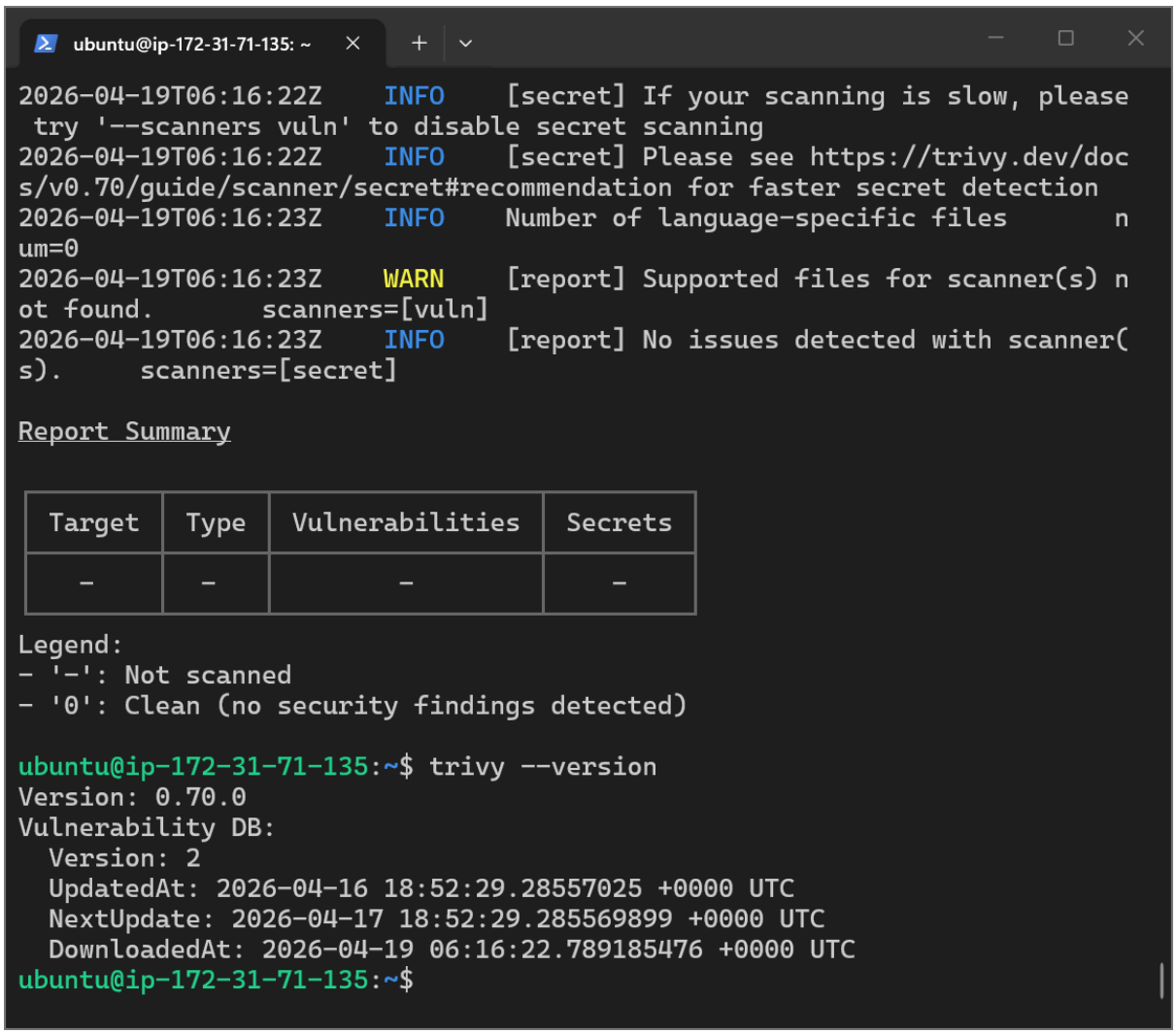
echo "deb [signed-by=/usr/share/keyrings/trivy.gpg]

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

sudo apt update
sudo apt install trivy -y

Verify

trivy –version



VERY IMPORTANT (for low RAM)

First time Trivy runs, it downloads DB → heavy (~100MB)

So run once manually:

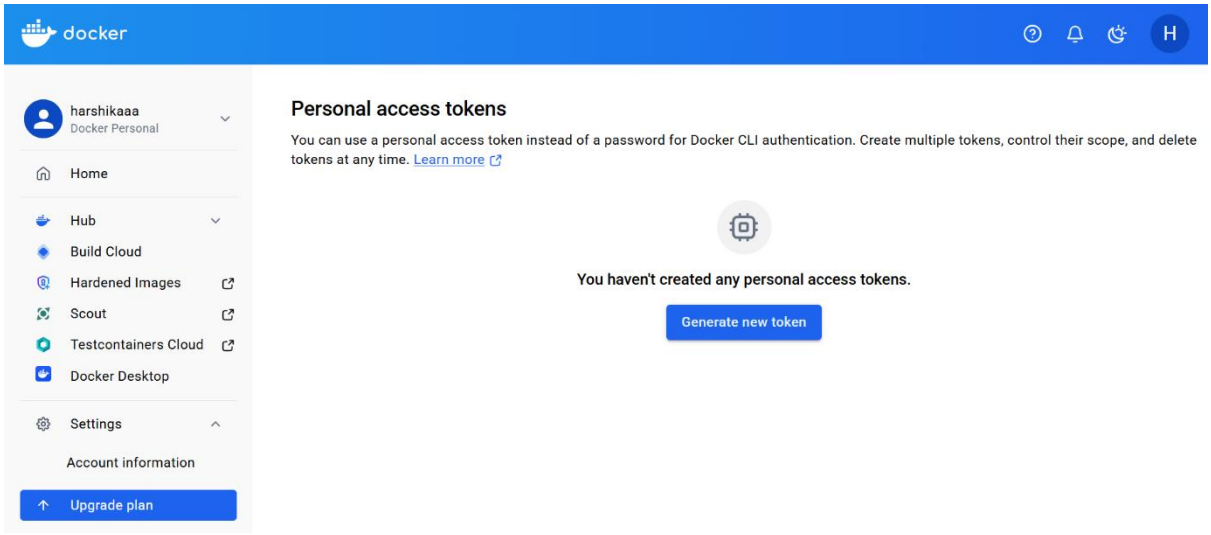
trivy image hello-world

```
ubuntu@ip-172-31-71-135: ~
No user sessions are running outdated binaries.

No VM guests are running outdated hypervisor (qemu) binaries on this
 host.
ubuntu@ip-172-31-71-135:~$ trivy image hello-world
2026-04-19T06:16:12Z    INFO    [vulndb] Need to update DB
2026-04-19T06:16:12Z    INFO    [vulndb] Downloading vulnerability DB...
2026-04-19T06:16:12Z    INFO    [vulndb] Downloading artifact...        repo="mirror.gcr.io/aquasec/trivy-db:2"
90.40 MiB / 90.40 MiB [----------------------] 100.00% 9.94 MiB p/s 9.3s
2026-04-19T06:16:22Z    INFO    [vulndb] Artifact successfully downloadedrepo="mirror.gcr.io/aquasec/trivy-db:2"
2026-04-19T06:16:22Z    INFO    [vuln] Vulnerability scanning is enabled
2026-04-19T06:16:22Z    INFO    [secret] Secret scanning is enabled
2026-04-19T06:16:22Z    INFO    [secret] If your scanning is slow, please try '--scanners vuln' to disable secret scanning
2026-04-19T06:16:22Z    INFO    [secret] Please see https://trivy.dev/docs/v0.70/guide/scanner/secret#recommendation for fas
ter secret detection
2026-04-19T06:16:23Z    INFO    Number of language-specific files       num=0
2026-04-19T06:16:23Z    WARN    [report] Supported files for scanner(s) not found.      scanners=[vuln]
2026-04-19T06:16:23Z    INFO    [report] No issues detected with scanner(s).    scanners=[secret]

Report Summary

┌────────┬──────┬─────────────────┬─────────┐
│ Target │ Type │ Vulnerabilities │ Secrets │
├────────┼──────┼─────────────────┼─────────┤
│   -    │  -   │        -        │    -    │
└────────┴──────┴─────────────────┴─────────┘
Legend:
- '-': Not scanned
- '0': Clean (no security findings detected)

ubuntu@ip-172-31-71-135:~$
```

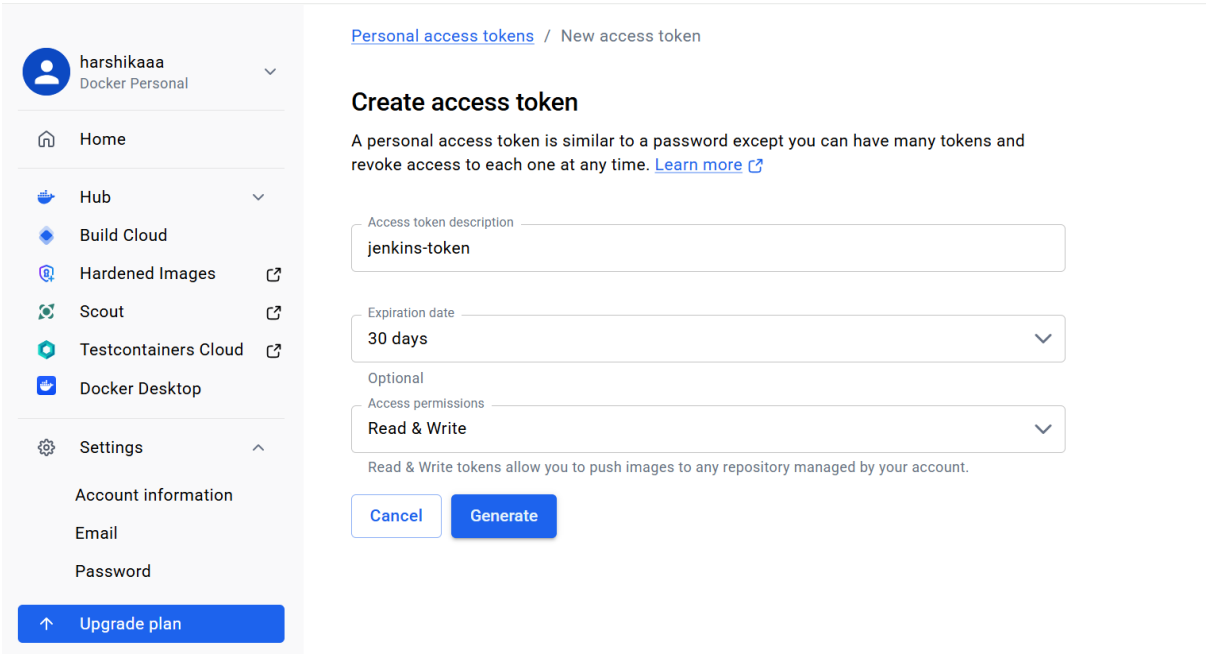
Phase 2:

Step 7- Create PAT in DockerHub:

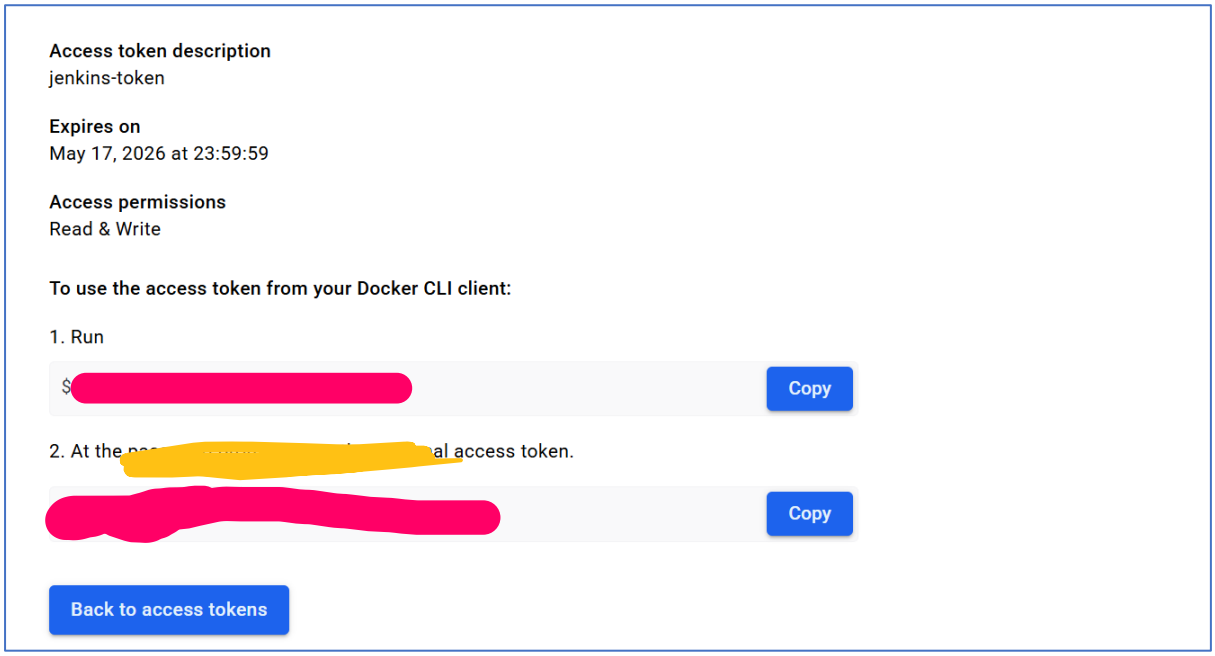
Log in to the DockerHub Account



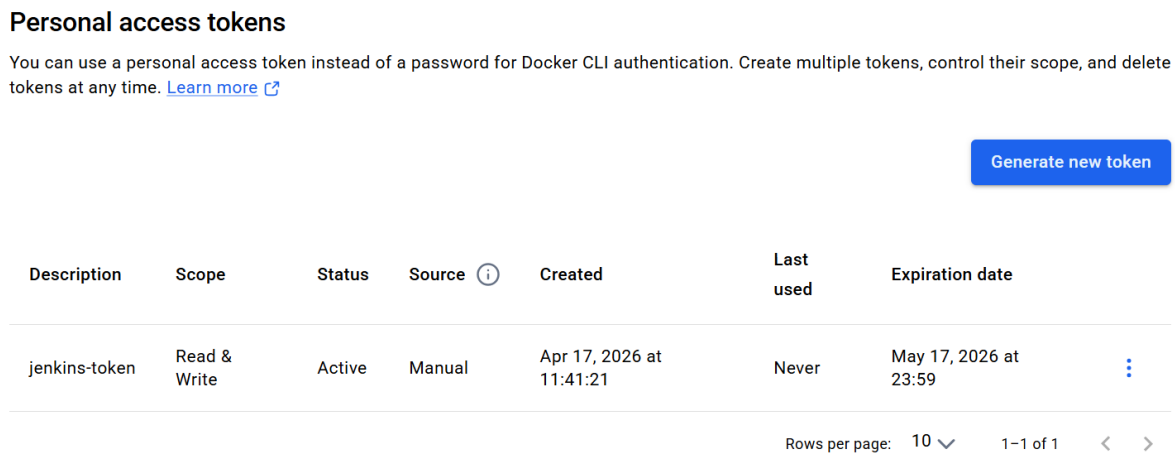
Create Personal Access Token



Generate



Generated!



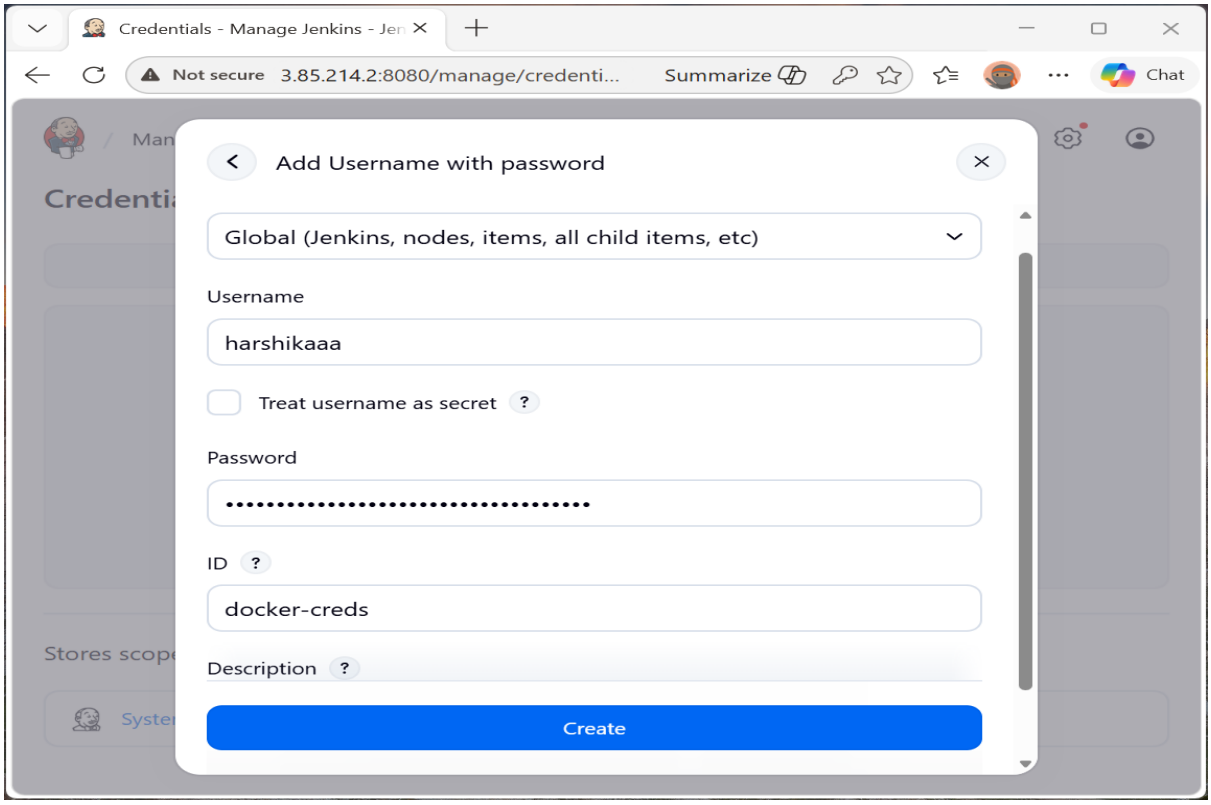
Step 8- Add DockerHub Credentials in Jenkins

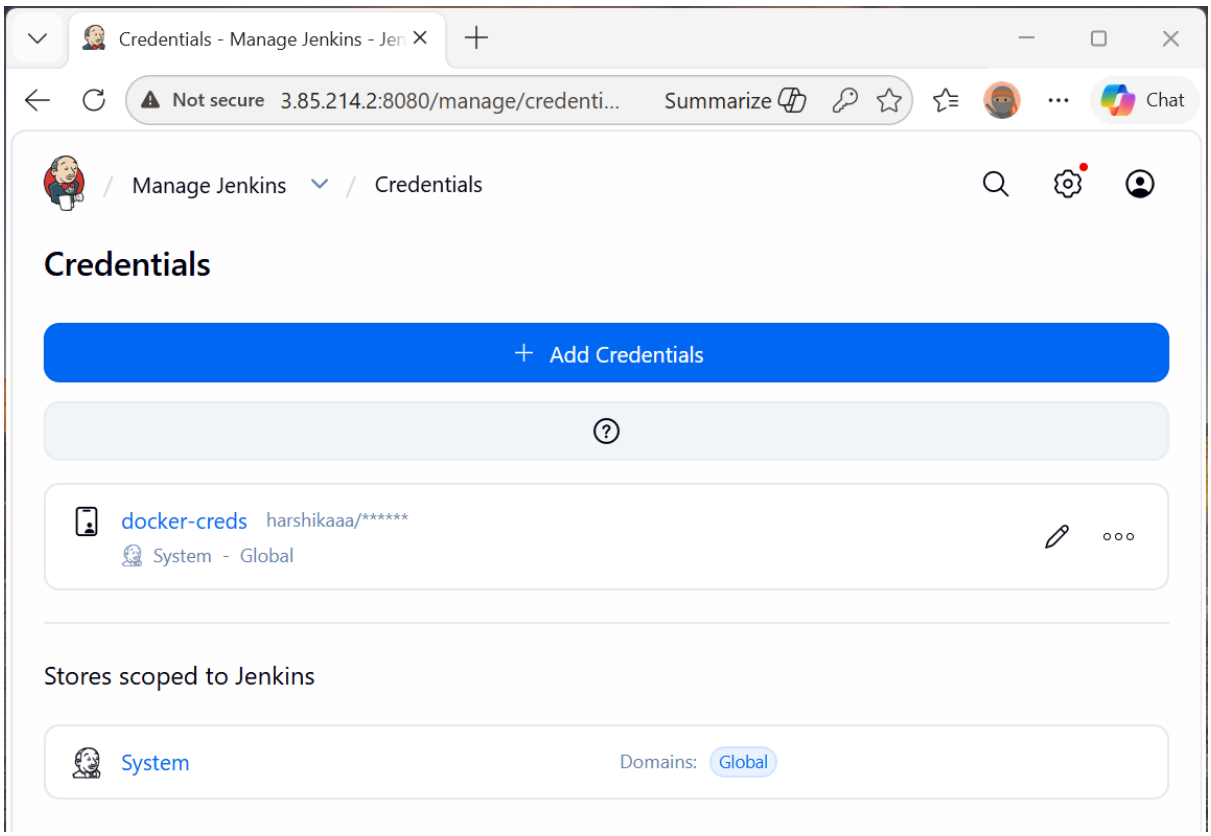
Go to:

Manage Jenkins → Credentials → Global → Add Credentials

Add:

- Username: your Docker username **(harshikaaa)**
- Password: NEW token
- ID: **docker-creds**


Credentials - Manage Jenkins - Jen
Not secure 3.85.214.2:8080/manage/credenti...
Summarize
Chat
Add Username with password
Global (Jenkins, nodes, items, all child items, etc)
Username
harshikaaa
Treat username as secret
Password
ID
docker-creds
Description
Create


Credentials - Manage Jenkins - Jen
Not secure 3.85.214.2:8080/manage/credenti...
Summarize
Chat
Manage Jenkins
Credentials
Credentials
Add Credentials
docker-creds harshikaaa/******
System - Global
Stores scoped to Jenkins
System
Domains: Global

Step 9- Add Tool Config

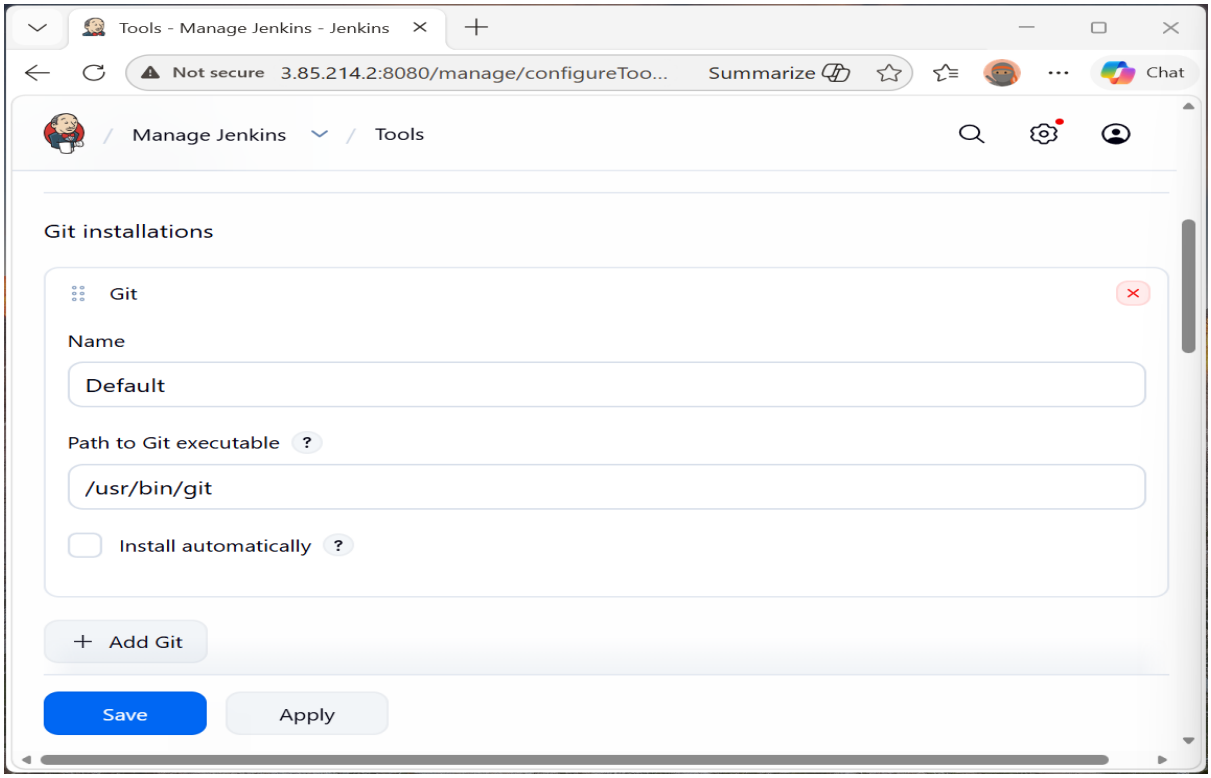
Go to manage Jenkins and tools:

Under Git

Name: Default

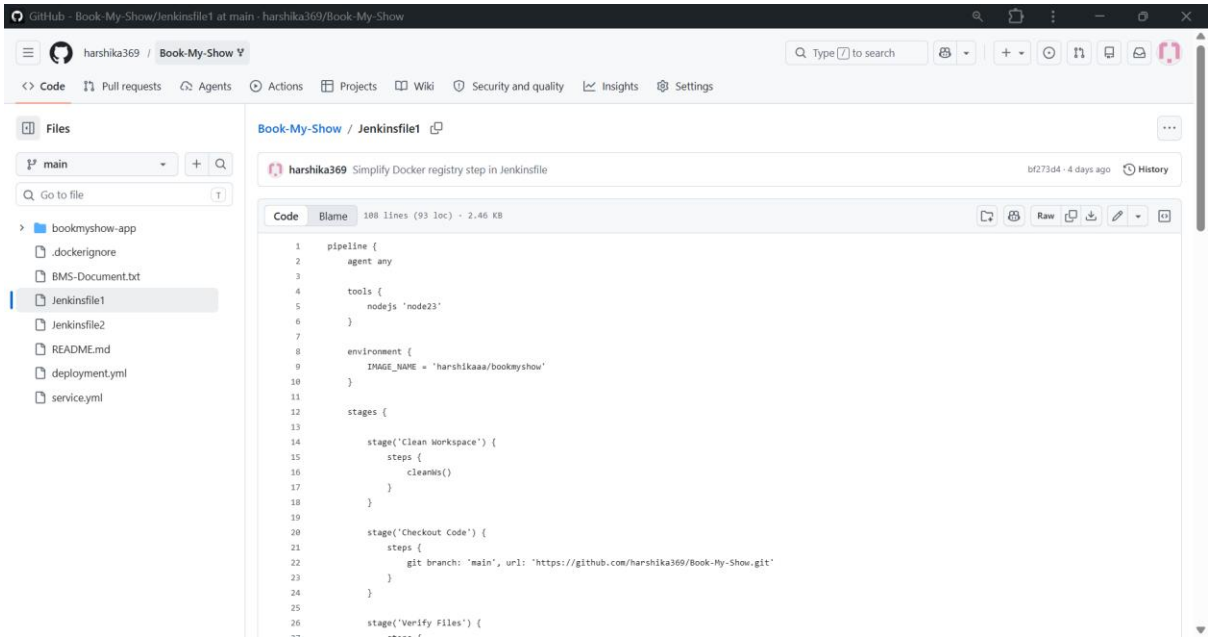
Path: /usr/bin/git

Click on apply and save



Now go to your GitHub repo that I have forked, and update the script of the file named **Jenkinsfile1** with this script and save commits.

https://github.com/harshika369/Book-My-Show.git



file- JenkinsFile1

```
pipeline {
    agent any

    tools {
        nodejs 'node18'
    }

    environment {
        IMAGE_NAME = 'harshikaaa/bookmyshow'
    }

    stages {
```

```
        stage('Clean Workspace') {
            steps {
                cleanWs()
            }
        }

        stage('Checkout Code') {
            steps {
                git branch: 'main', url: 'https://github.com/harshika369/Book-My-Show.git'
            }
        }

        stage('Verify Files') {
            steps {
                sh '''
                echo "Root folder:"
                ls -la
                echo "App folder:"
                ls -la bookmyshow-app
                '''
            }
        }
```

```
stage('Install Dependencies') {
    steps {
        dir('bookmyshow-app') {
            sh '''
            if [ -f package.json ]; then
                rm -rf node_modules package-lock.json
                npm install
            else
                echo "package.json not found!"
                exit 1
            fi
            '''
        }
    }
}

stage('Trivy FS Scan') {
    steps {
        sh 'trivy fs . > trivyfs.txt'
    }
}

stage('Build Docker Image') {
    steps {
        sh '''
```

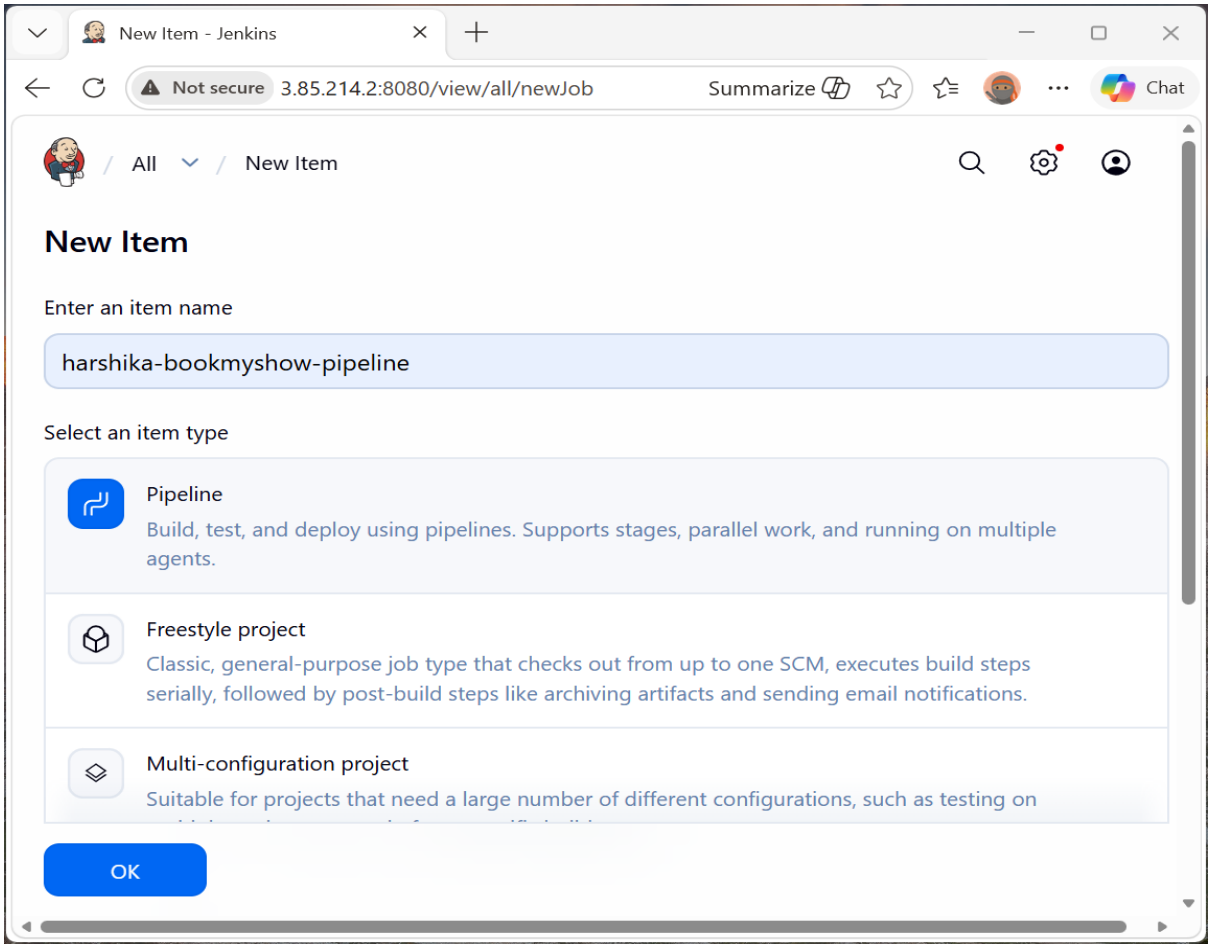
```
            docker build -t $IMAGE_NAME:latest -f bookmyshow-app/Dockerfile
bookmyshow-app
            '''
        }
    }

    stage('Trivy Image Scan') {
        steps {
            sh 'trivy image $IMAGE_NAME:latest > trivyimage.txt'
        }
    }

    stage('Docker Push') {
        steps {
            script {
                withDockerRegistry(credentialsId: 'docker-creds') {
                    sh '''
                    docker push $IMAGE_NAME:latest
                    '''
                }
            }
        }
    }

    stage('Deploy Container') {
```

```
        steps {
            sh '''
            docker stop bms || true
            docker rm bms || true

            docker run -d \
              --name bms \
              -p 3000:3000 \
              --restart=always \
              $IMAGE_NAME:latest

            docker ps -a
            '''
        }
      }
    }

    post {
      always {
        archiveArtifacts artifacts: 'trivyfs.txt,trivyimage.txt', fingerprint: true
      }
    }
}
```

STEP 10: Create Jenkins Pipeline Job

Go to Jenkins Dashboard:

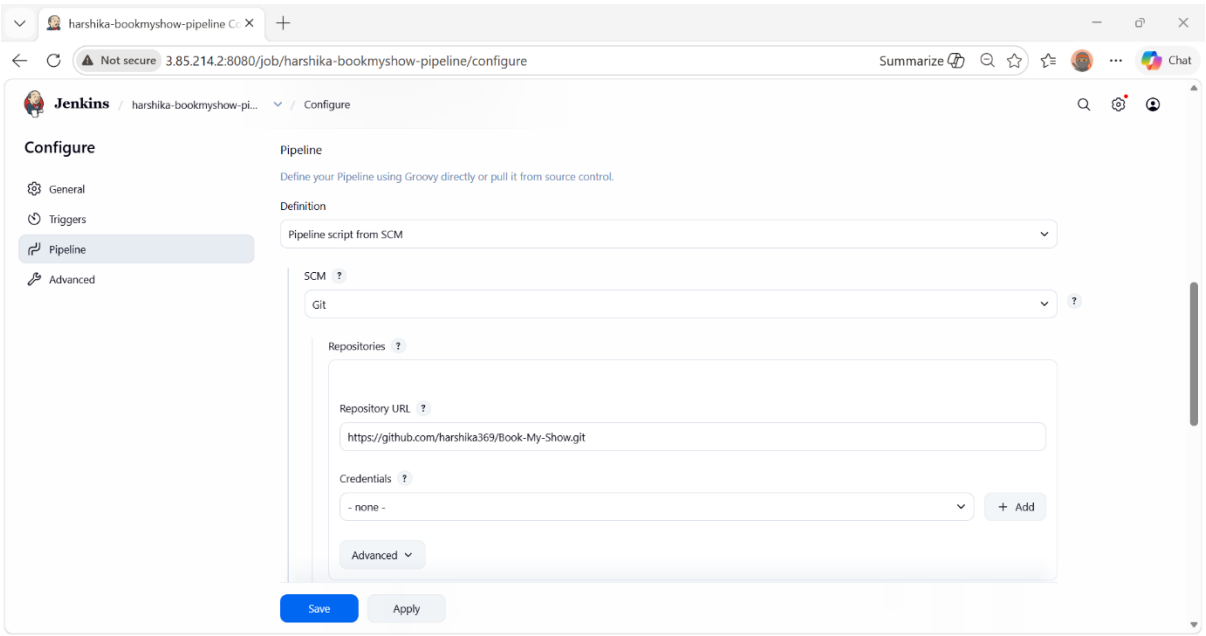
1. **Click New Item**
2. **Name: Harshika-bookmyshow-pipeline**
3. Select Pipeline



Configure:

Under the Pipeline section:

- Definition → Pipeline script from SCM
- SCM → Git
- Repo URL:
- https://github.com/harshika369/Book-My-Show.git



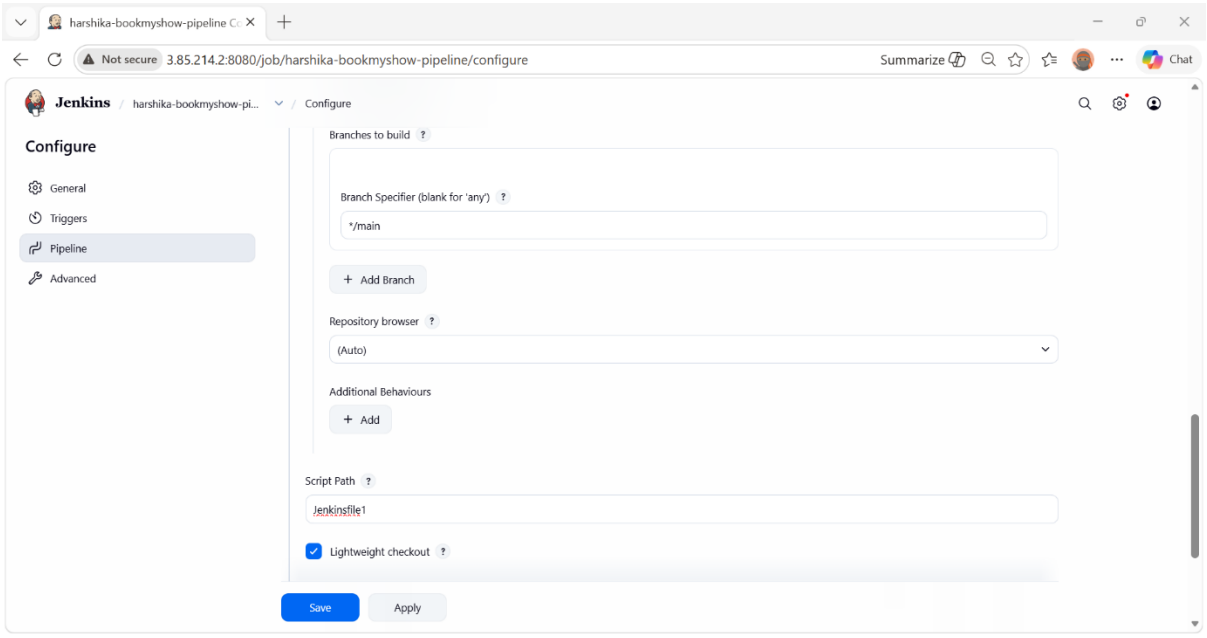
- Branch:

  ***/main**

- Script Path:

  Jenkinsfile1

Save

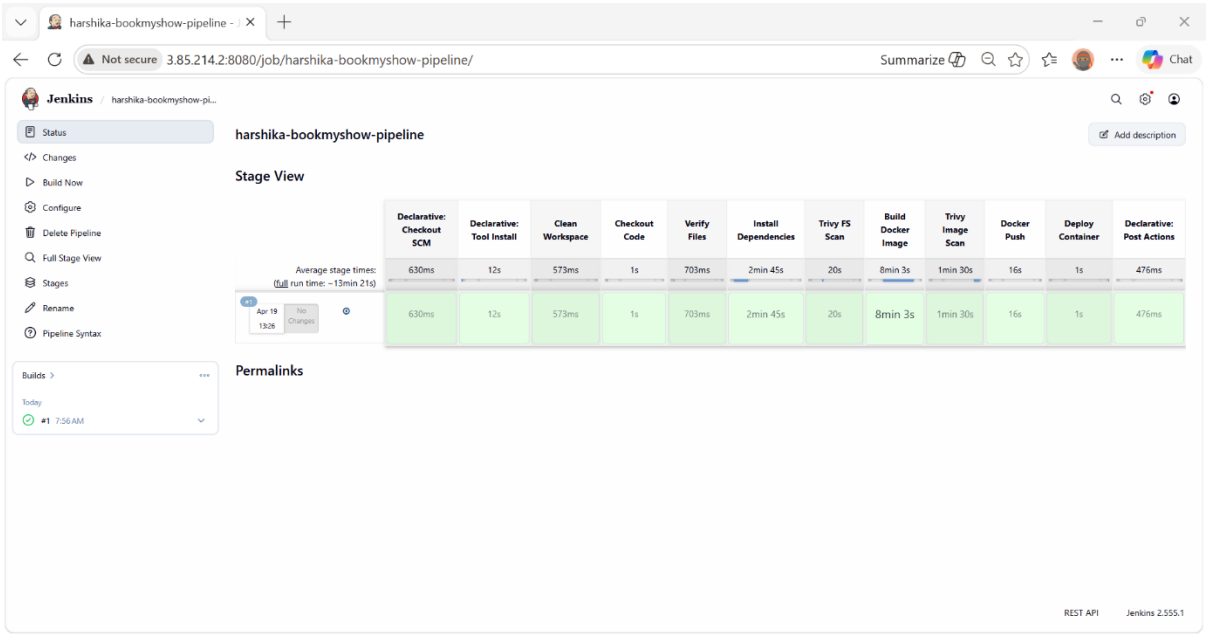


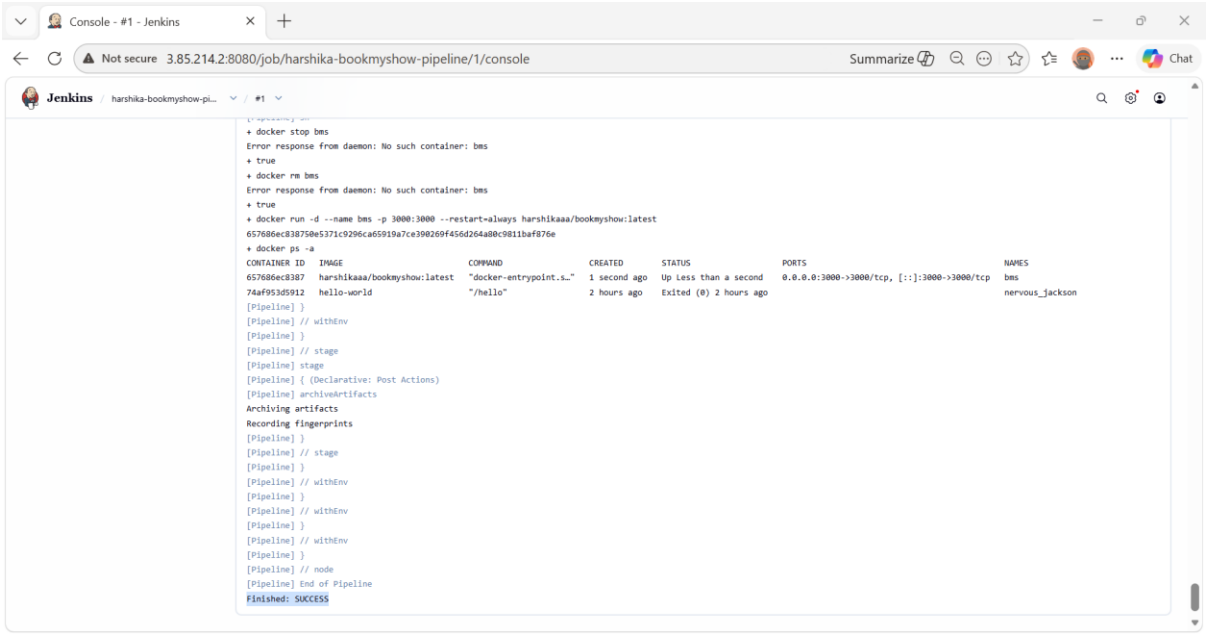
STEP 11: Run Pipeline

Click:
Build Now

Stages will run:

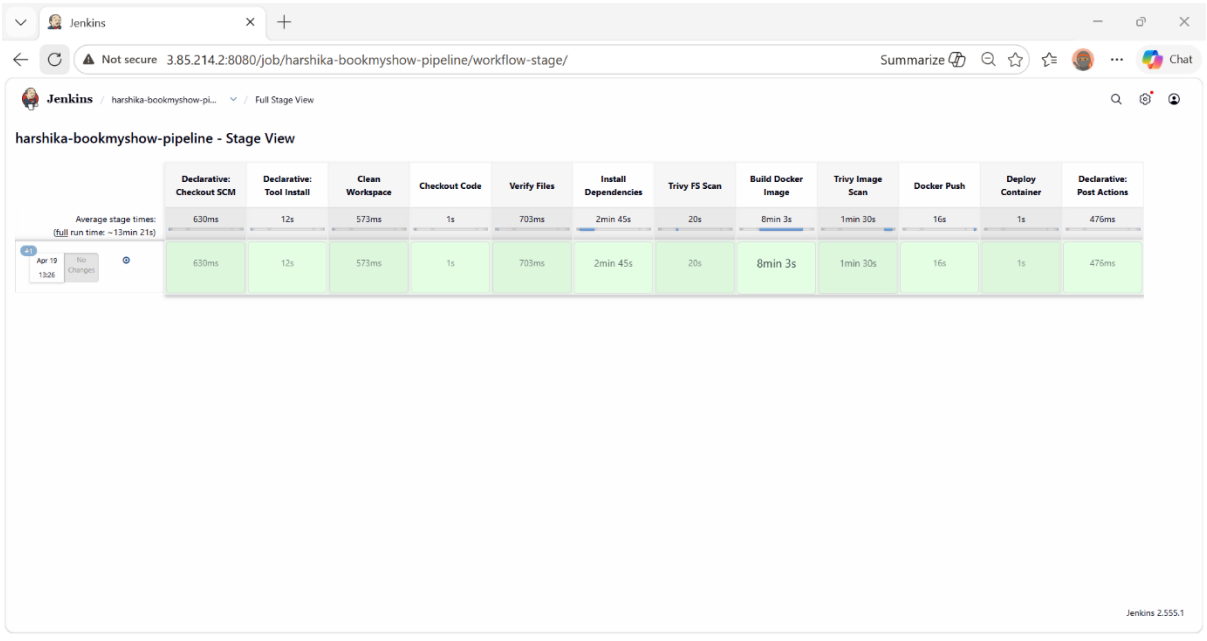
1. Clean Workspace ✅
2. Checkout Code ✅
3. Install Dependencies (Node works) ✅
4. Trivy Scan ✅
5. Docker Build ✅
6. Docker Push ✅
7. Deploy Container ✅





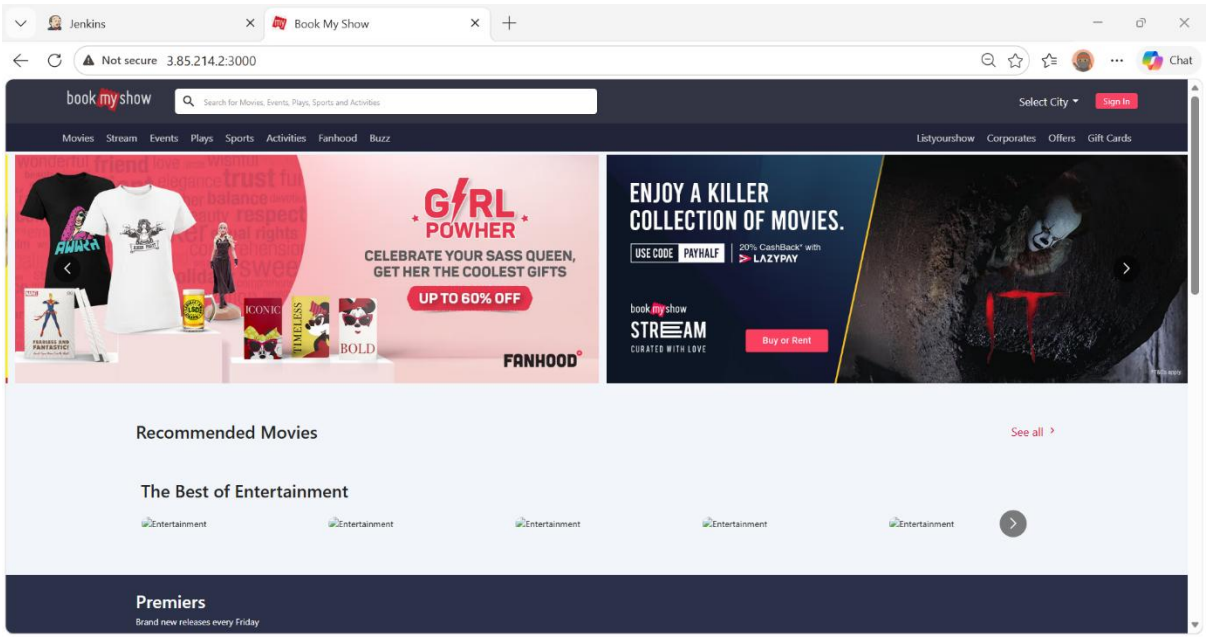
STEP 12: Access the App

After success:



Open in browser:

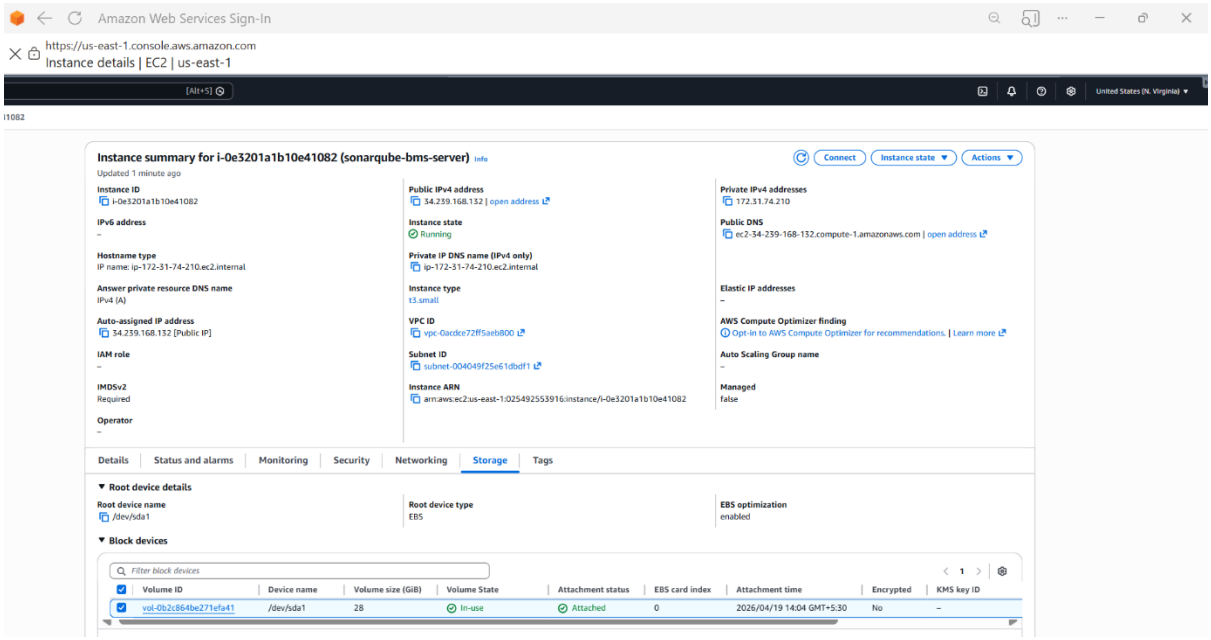
http://<EC2-PUBLIC-IP>:3000



STEP 13: Set up SonarQube Server (Separate VM)

1. Create VM:

- Ubuntu 22.04 / 24.04
- Instance: t3.small
- Open port: **9000**



2. Connect to the Sonarqube server:

cd Downloads

icacls "your-key.pem" /inheritance:r

icacls "your-key.pem" /grant:r "$($env:USERNAME):R"

ssh -i "your-key.pem" ubuntu@<SONAR_PUBLIC_IP>

3. Once Connected (Inside Server)

Now run these **exact commands for Sonar setup**

Add SWAP (VERY IMPORTANT)

sudo fallocate -l 2G /swapfile
sudo chmod 600 /swapfile
sudo mkswap /swapfile

sudo swapon /swapfile
free -h

Make permanent:

echo '/swapfile none swap sw 0 0' | sudo tee -a /etc/fstab

```
ubuntu@ip-172-31-74-210:~$ sudo sysctl -w vm.max_map_count=262144
sudo sysctl -w fs.file-max=65536
vm.max_map_count = 262144
fs.file-max = 65536
ubuntu@ip-172-31-74-210:~$ echo "vm.max_map_count=262144" | sudo tee -a /etc/s
ysctl.conf
echo "fs.file-max=65536" | sudo tee -a /etc/sysctl.conf
vm.max_map_count=262144
fs.file-max=65536
ubuntu@ip-172-31-74-210:~$ docker --version
Docker version 29.1.3, build 29.1.3-0ubuntu3~24.04.1
ubuntu@ip-172-31-74-210:~$ sudo chmod 666 /var/run/docker.sock
ubuntu@ip-172-31-74-210:~$ free -h
               total        used        free      shared  buff/cache   availab
le
Mem:           1.9Gi       427Mi       667Mi       2.8Mi       987Mi       1.4
Gi
Swap:          2.0Gi          0B       2.0Gi
ubuntu@ip-172-31-74-210:~$
```

4. Install Docker

sudo apt update
sudo apt install docker.io -y
sudo systemctl start docker
sudo systemctl enable docker
sudo chmod 666 /var/run/docker.sock

```
ubuntu@ip-172-31-74-210:~$ sudo apt update
sudo apt install docker.io -y
sudo systemctl start docker
sudo systemctl enable docker
sudo chmod 666 /var/run/docker.sock
```

Then check logs:

docker logs sonar

Make sure you see:

SonarQube is up

```
ubuntu@ip-172-31-74-210:~$ docker ps
CONTAINER ID   IMAGE                    COMMAND                  CREATED         STATUS         PORTS
          NAMES
c16ef4f3df8a   sonarqube:lts-community   "/opt/sonarqube/dock…"   9 seconds ago   Up 9 seconds   0.0.0.0:9000->9000/tcp, [::]:9000->9
000/tcp   sonar
ubuntu@ip-172-31-74-210:~$
```

5. Kernel settings (MANDATORY for Sonar)

sudo sysctl -w vm.max_map_count=262144
sudo sysctl -w fs.file-max=65536

Permanent:

echo "vm.max_map_count=262144" | sudo tee -a /etc/sysctl.conf
echo "fs.file-max=65536" | sudo tee -a /etc/sysctl.conf

```
ubuntu@ip-172-31-74-210:~$ sudo sysctl -w vm.max_map_count=262144
sudo sysctl -w fs.file-max=65536
vm.max_map_count = 262144
fs.file-max = 65536
ubuntu@ip-172-31-74-210:~$ echo "vm.max_map_count=262144" | sudo tee -a /etc/s
ysctl.conf
echo "fs.file-max=65536" | sudo tee -a /etc/sysctl.conf
vm.max_map_count=262144
fs.file-max=65536
ubuntu@ip-172-31-74-210:~$
```

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

```
ubuntu@ip-172-31-74-210:~$ docker run -d --name sonar \
-p 9000:9000 \
sonarqube:community
Unable to find image 'sonarqube:community' locally
community: Pulling from library/sonarqube
c235bc109909: Pull complete
5f6278b6d85f: Pull complete
b40150c1c271: Pull complete
4f4fb700ef54: Pull complete
69b037921f81: Pull complete
16992be7d806: Pull complete
456eee9f1cc2: Pull complete
c25dbfa2b07e: Download complete
19f6168e631a: Download complete
Digest: sha256:177e3708ef0c205ad7f9564377fe974757ea588a1e2a2e25c4050bcf1958718
8
Status: Downloaded newer image for sonarqube:community
74f5f0c8e9673af7b3ee34dd767a41e33abcad55f2bb9260e3b3635213bb3e4b
ubuntu@ip-172-31-74-210:~$
```

6. Permissions for Docker

Run this (important):

sudo chmod 666 /var/run/docker.sock

```
ubuntu@ip-172-31-74-210:~$ sudo chmod 666 /var/run/docker.sock
ubuntu@ip-172-31-74-210:~$ free -h
               total        used        free      shared  buff/cache   availab
le
Mem:           1.9Gi       427Mi       667Mi       2.8Mi       987Mi       1.4
Gi
Swap:          2.0Gi          0B       2.0Gi
ubuntu@ip-172-31-74-210:~$
```

6. SWAP memory

Check:free -h

swapon --show

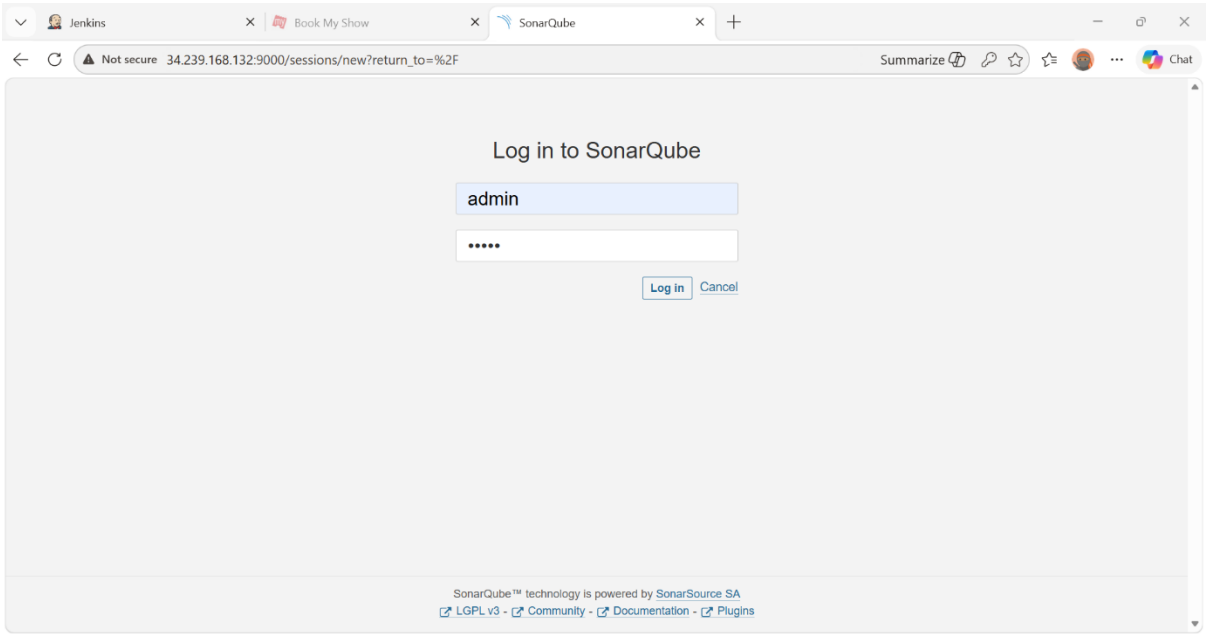
7. Access Sonar UI

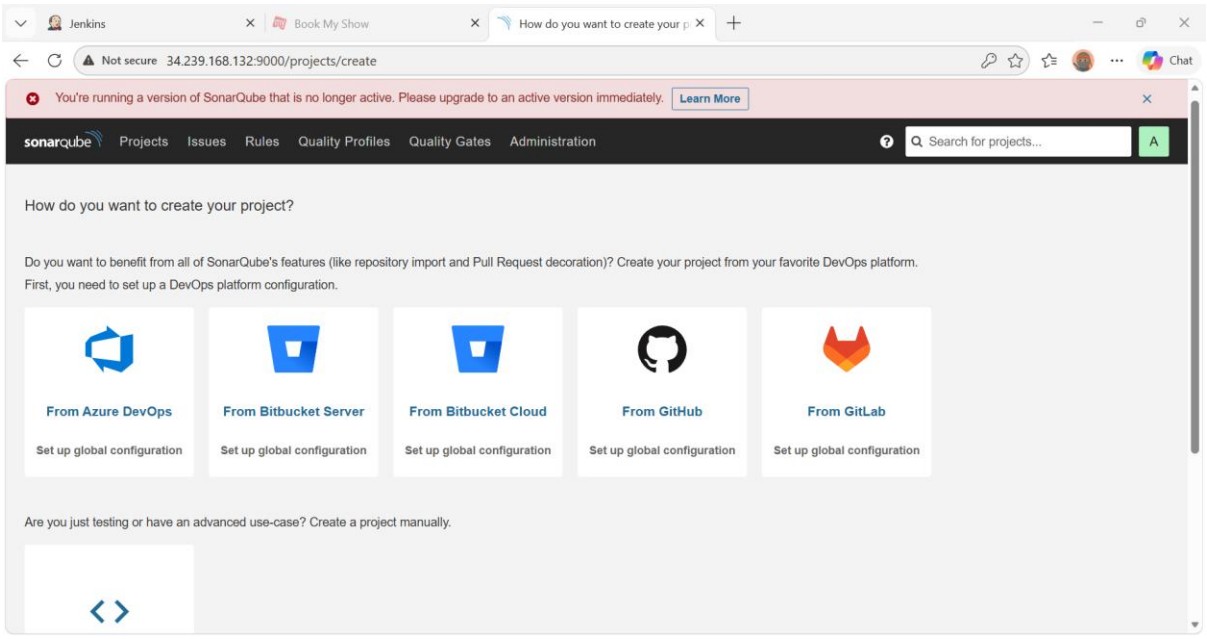
Open browser:

http://<SONAR_PUBLIC_IP>:9000

Login

username: admin
password: admin





8. Connect SonarQube with Jenkins

Generate Token in SonarQube

Open:

http://<your-sonar-public-ip>:9000

Login:

- username: admin
- password: (your new password)

9. Generate token

Step 1:

Top right corner → click **your profile icon (A)**

Step 2:

Click **“My Account”**

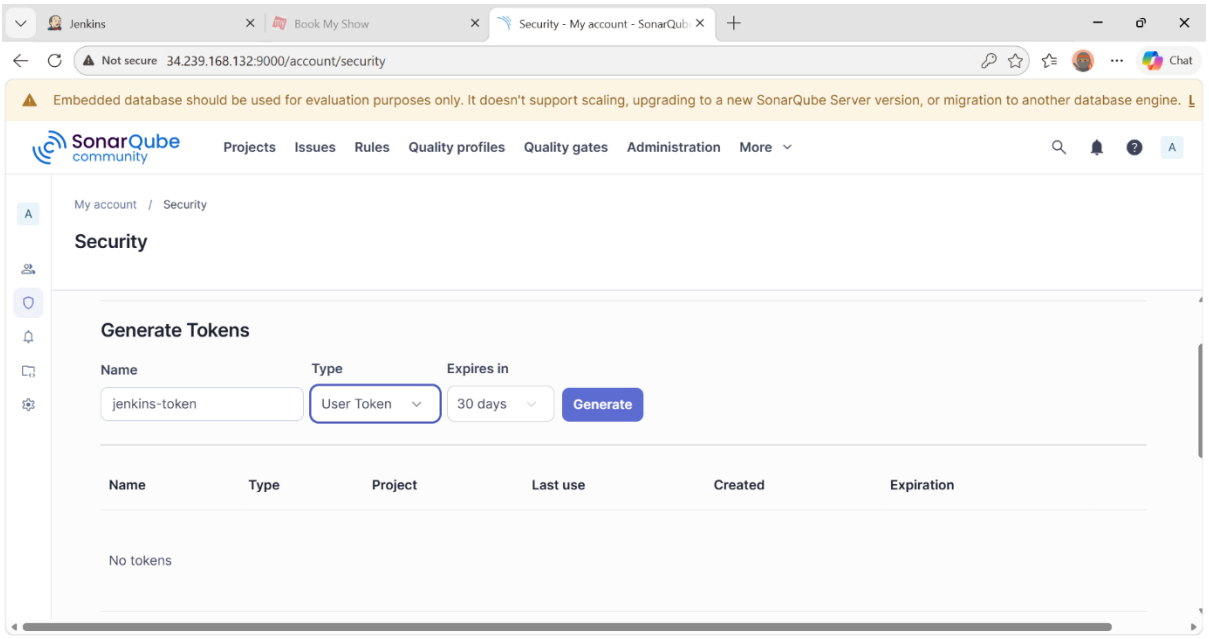
Step 3:

Go to the **“Security” tab**

Step 4:

Under **Tokens section**:

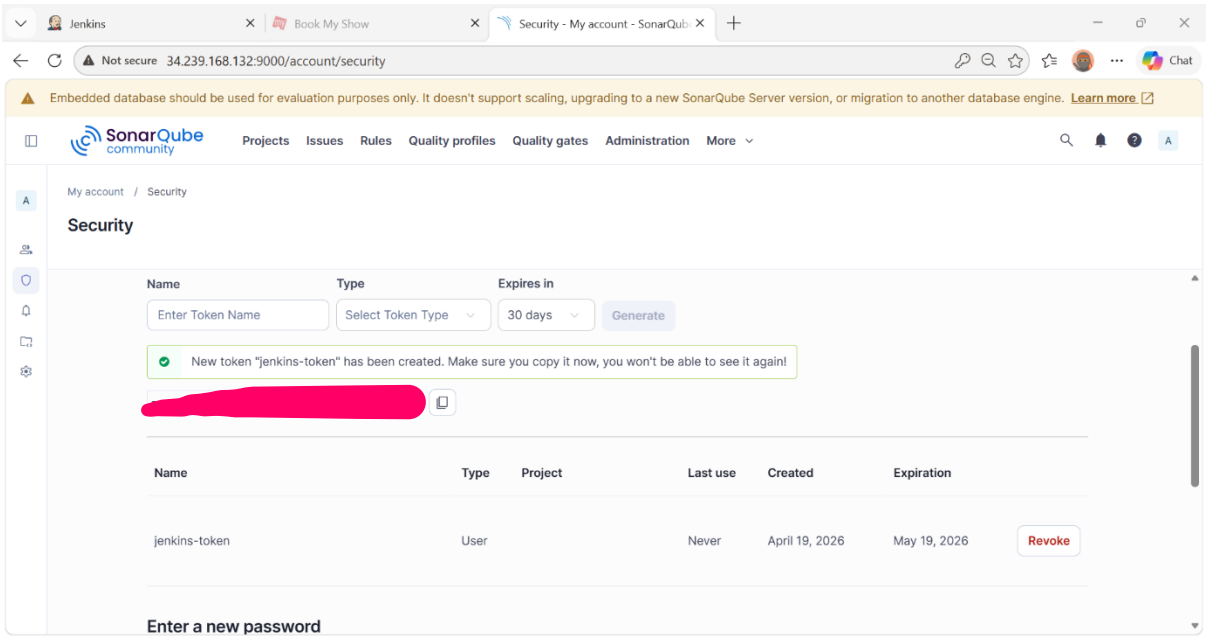
- Name: jenkins-token
- Click **Generate**



Step 5:

IMPORTANT

- Copy the token immediately (you won't see it again)



10. Connect SonarQube with Jenkins

Configure SonarQube in Jenkins

Manage Jenkins → System

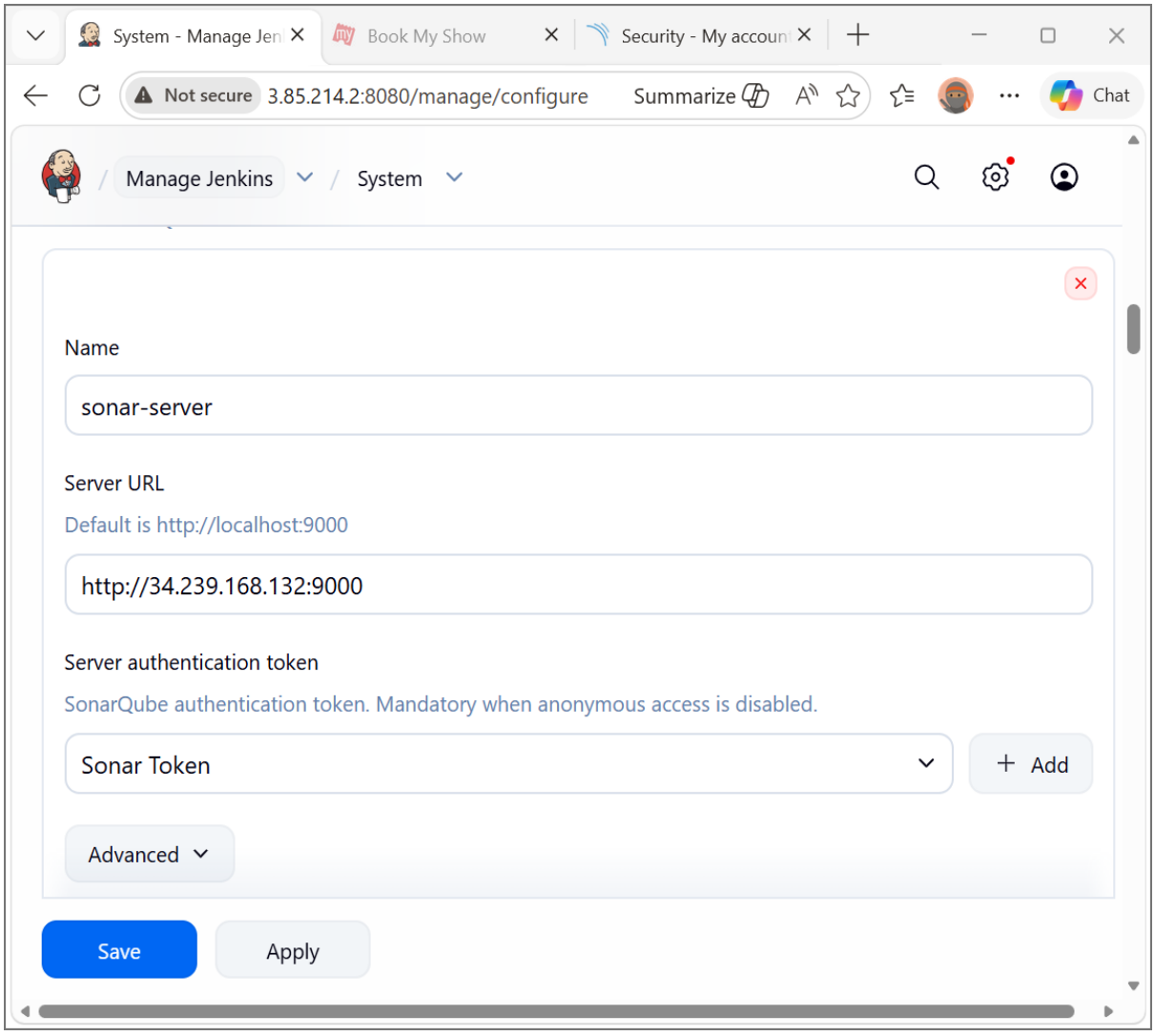
Scroll to: **SonarQube servers**

Click → **Add SonarQube**

Fill:

- Name: sonar-server
- Server URL:

http://<sonar-private-ip>:9000



Add Token:

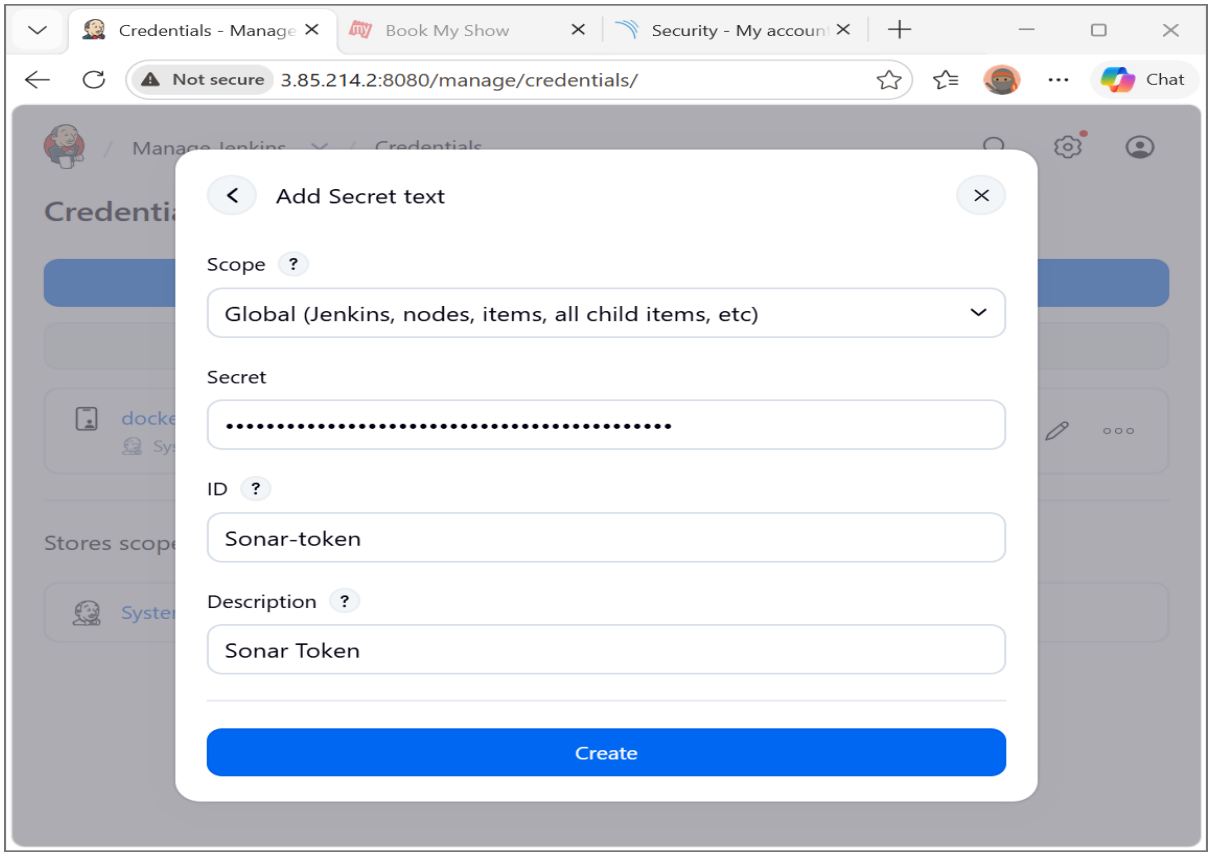
Click:

- **Add → Jenkins**

Fill:

- Kind → Secret Text
- Secret → *(paste your token)*
- ID → Sonar-token

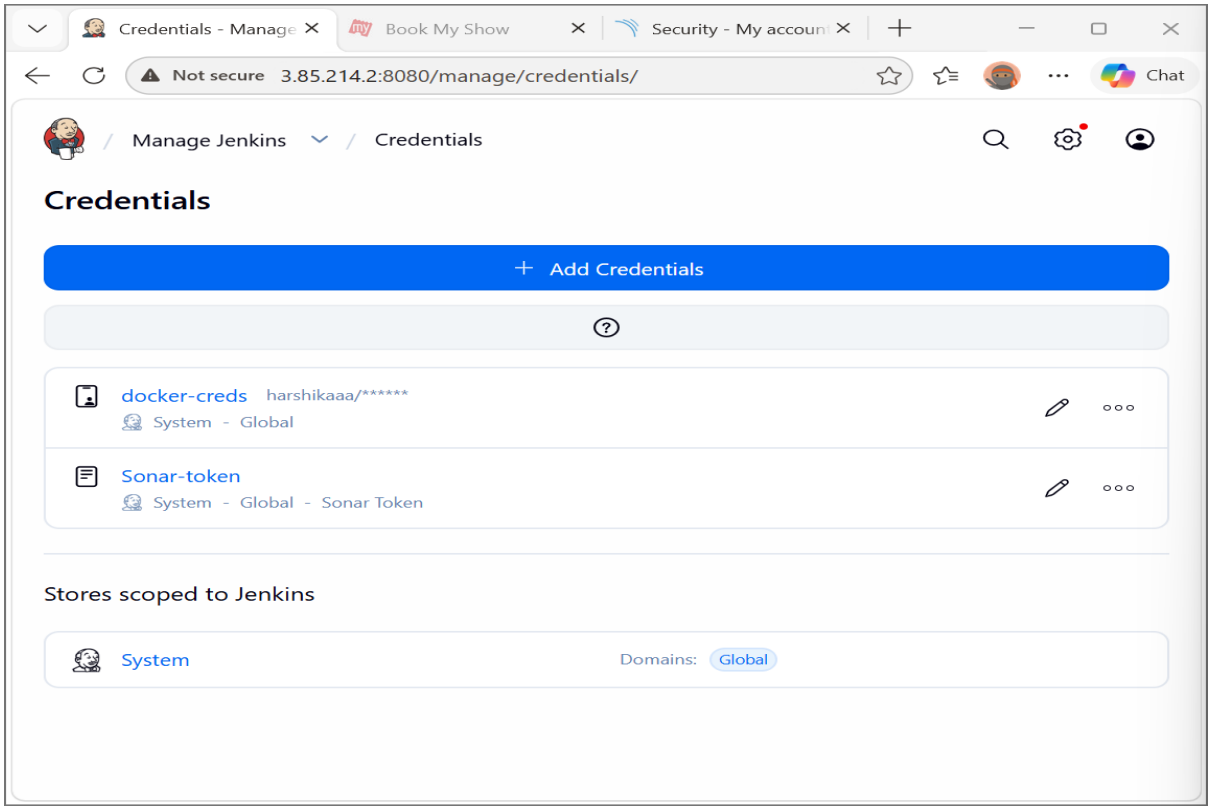
Click **Add**



Now select:

- Credentials → Sonar-token

Click **Save**



11. Configure Sonar Scanner Tool

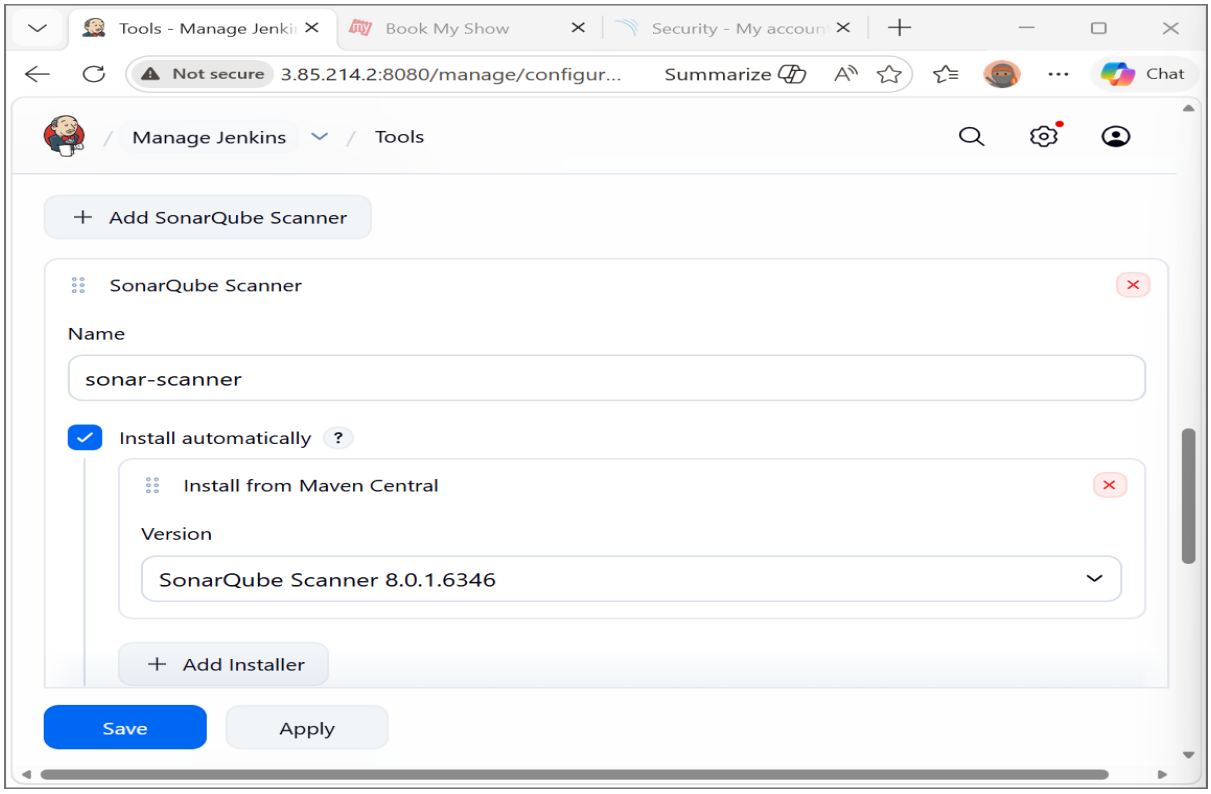
Manage Jenkins → Tools

SonarQube Scanner

Click **Add**

- Name → sonar-scanner
- Tick → Install automatically

Click **Save**



Restart Jenkins (IMPORTANT)

Run on Jenkins server:

sudo systemctl restart jenkins

```
ubuntu@ip-172-31-71-135:~$ sudo systemctl restart jenkins
ubuntu@ip-172-31-71-135:~$
```

12. Other Tools Configuration in Jenkins:

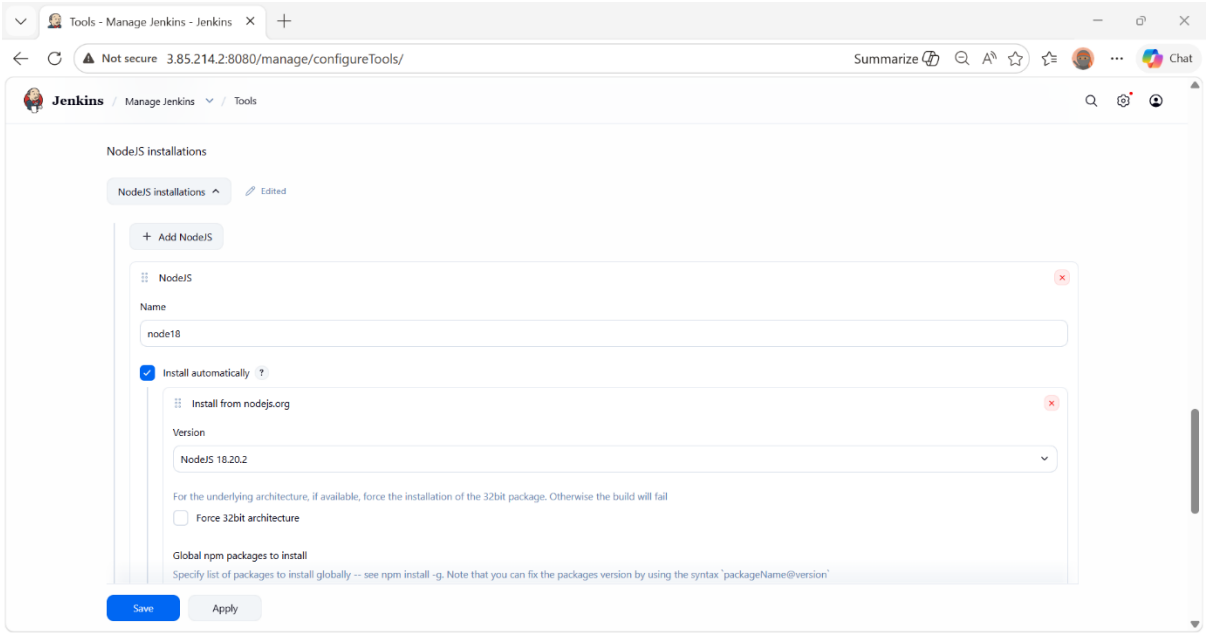
Go to manage Jenkins and tools

Node.js installation

Name: node18

Select install automatically

Version?: NodeJS 23.7.0



Dependency-check installations

Add Dependency-check

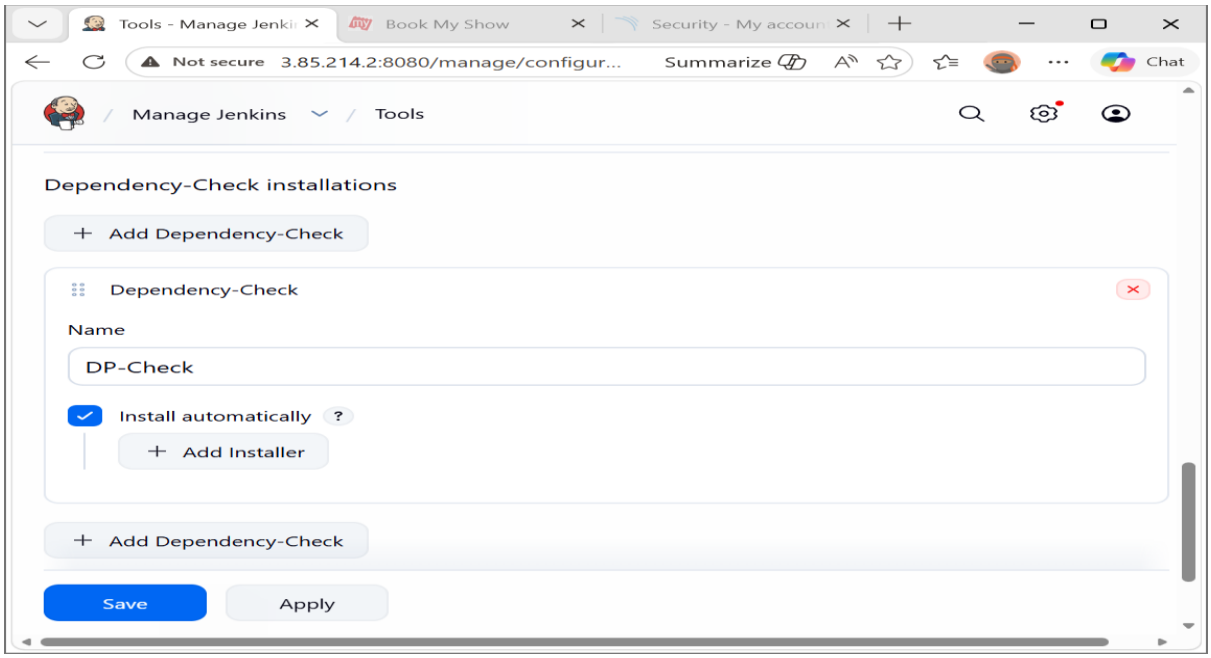
Name: DP-Check

Select install automatically

add installer

Install from github.com

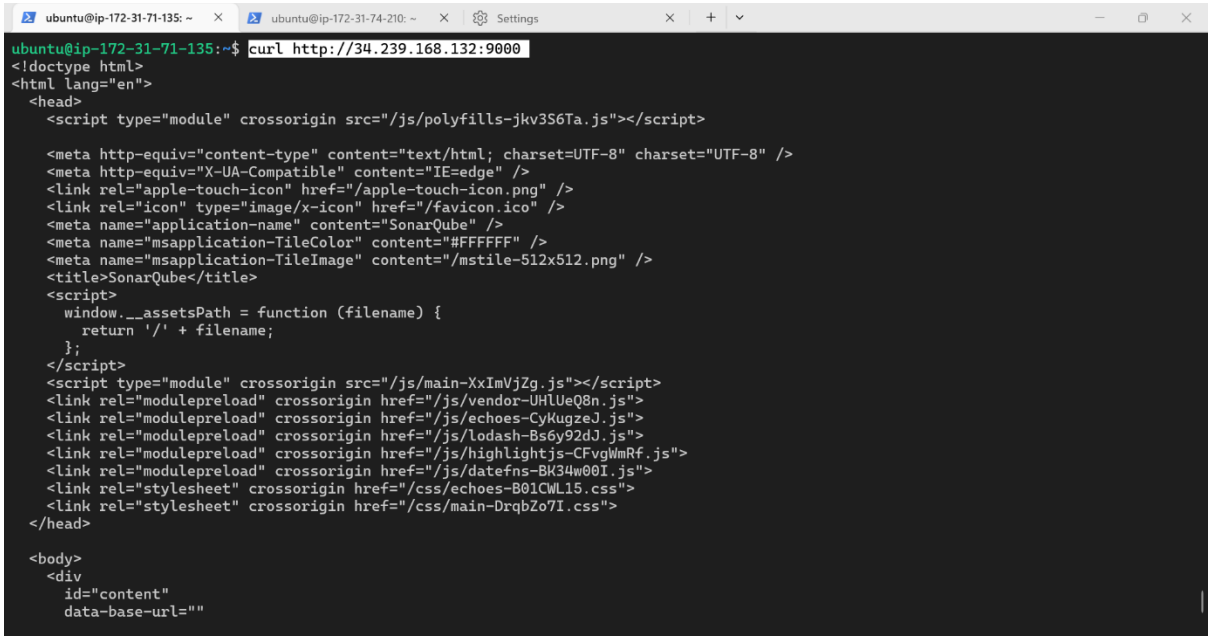
version?: dependency-check 12.0.2



13. To confirm the SonarQube setup in Jenkins

Replace with your actual SonarQube public IP:

curl http://YOUR_SONAR_PUBLIC_IP:9000



Expected Output

You should see HTML like:

<!doctype html>
<html ...

That means the Jenkins server **can reach SonarQube**

14. Generate Gmail App Password

Step 1: Open Google Account Security

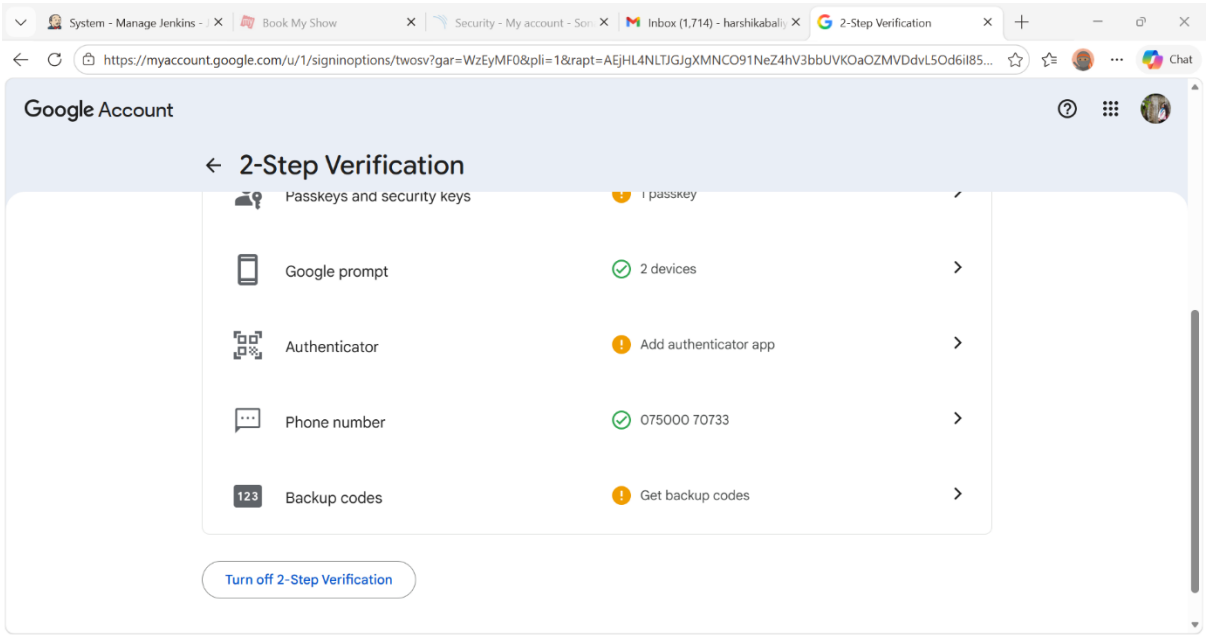
In your browser, go to:
https://myaccount.google.com/security

Step 2: Enable 2-Step Verification (MANDATORY)

Scroll → find:

“2-Step Verification”

- Click it
- Click **Get Started**
- Enter your Gmail password
- Choose verification method (OTP / phone)
- Complete setup



Step 3: Open App Passwords

Now on the same page:

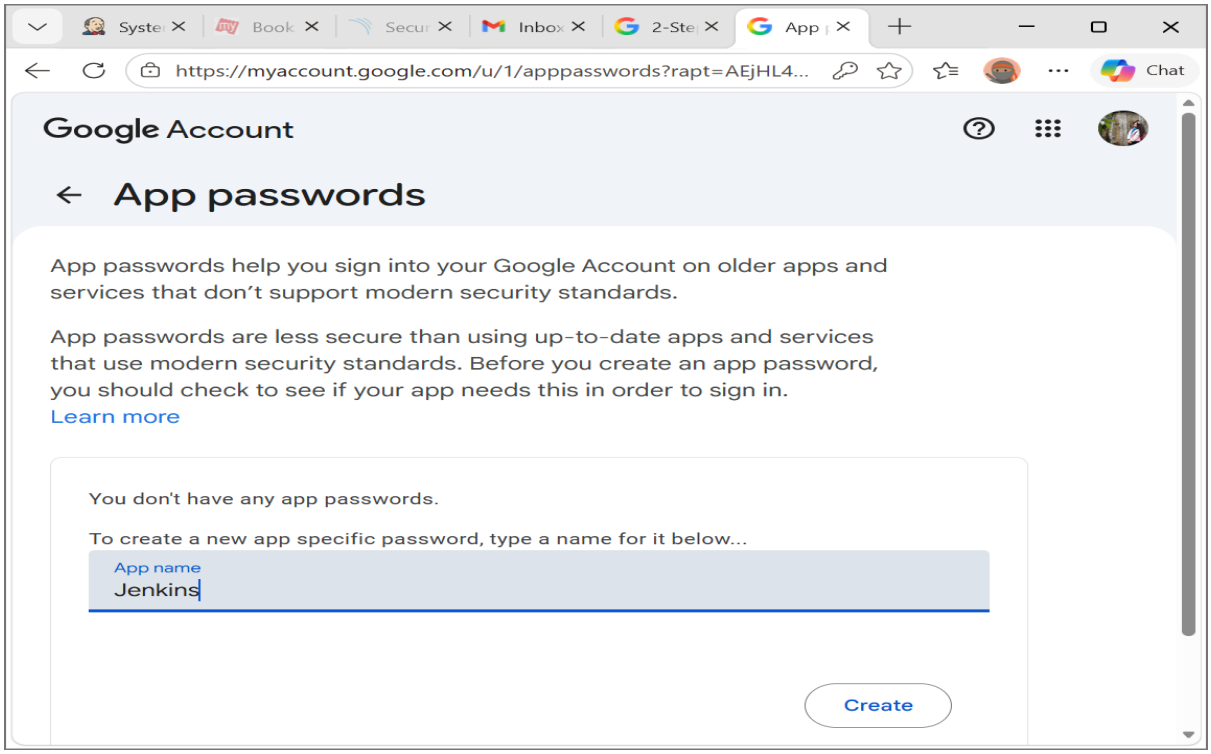
Search at top: **App Passwords**

Step 4: Generate Password

Inside App Passwords:

- Select App → **Mail**
- Select Device → **Other**
- Enter name → Jenkins

Click **Generate**

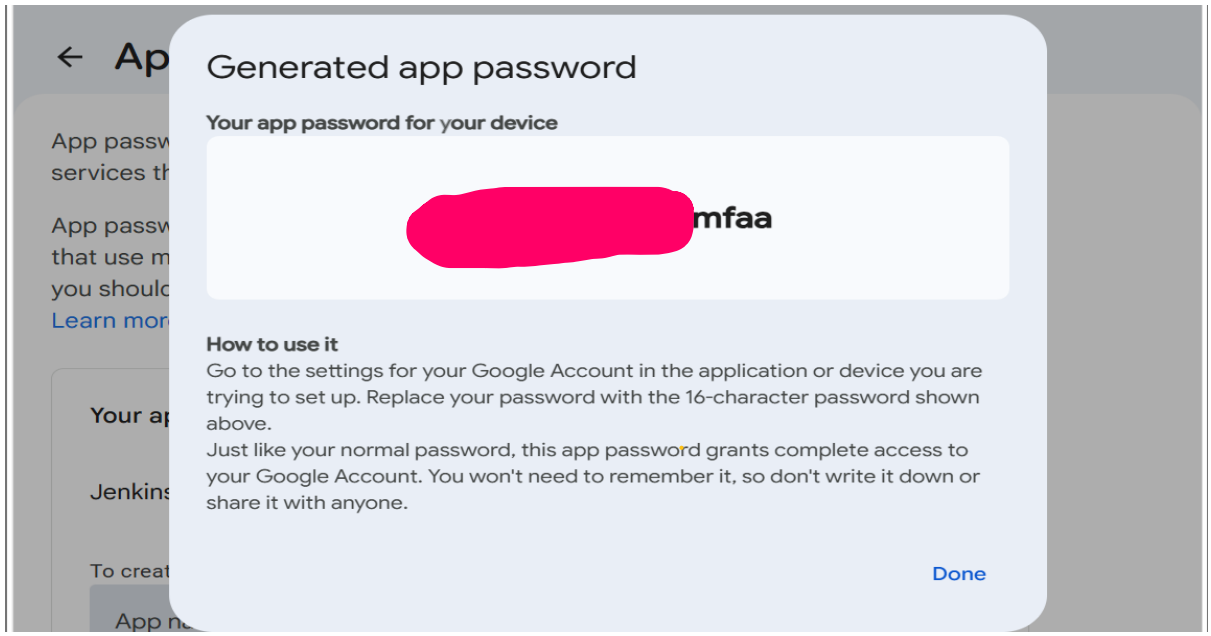


Step 5: Copy Password

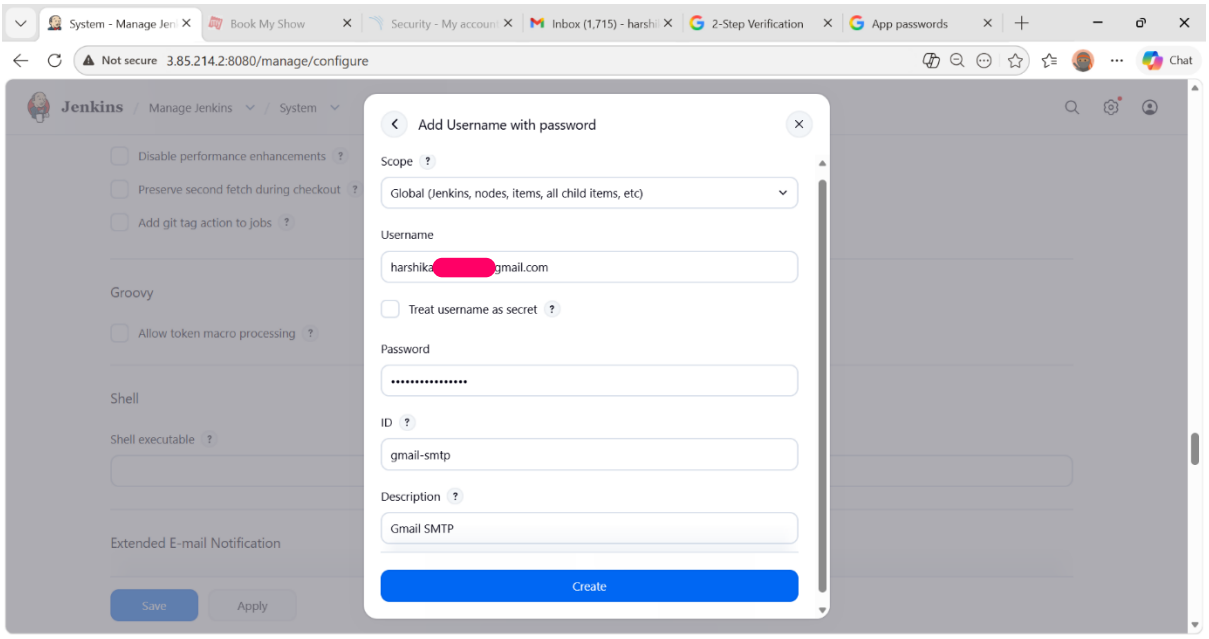
Copy it
REMOVE spaces → becomes:

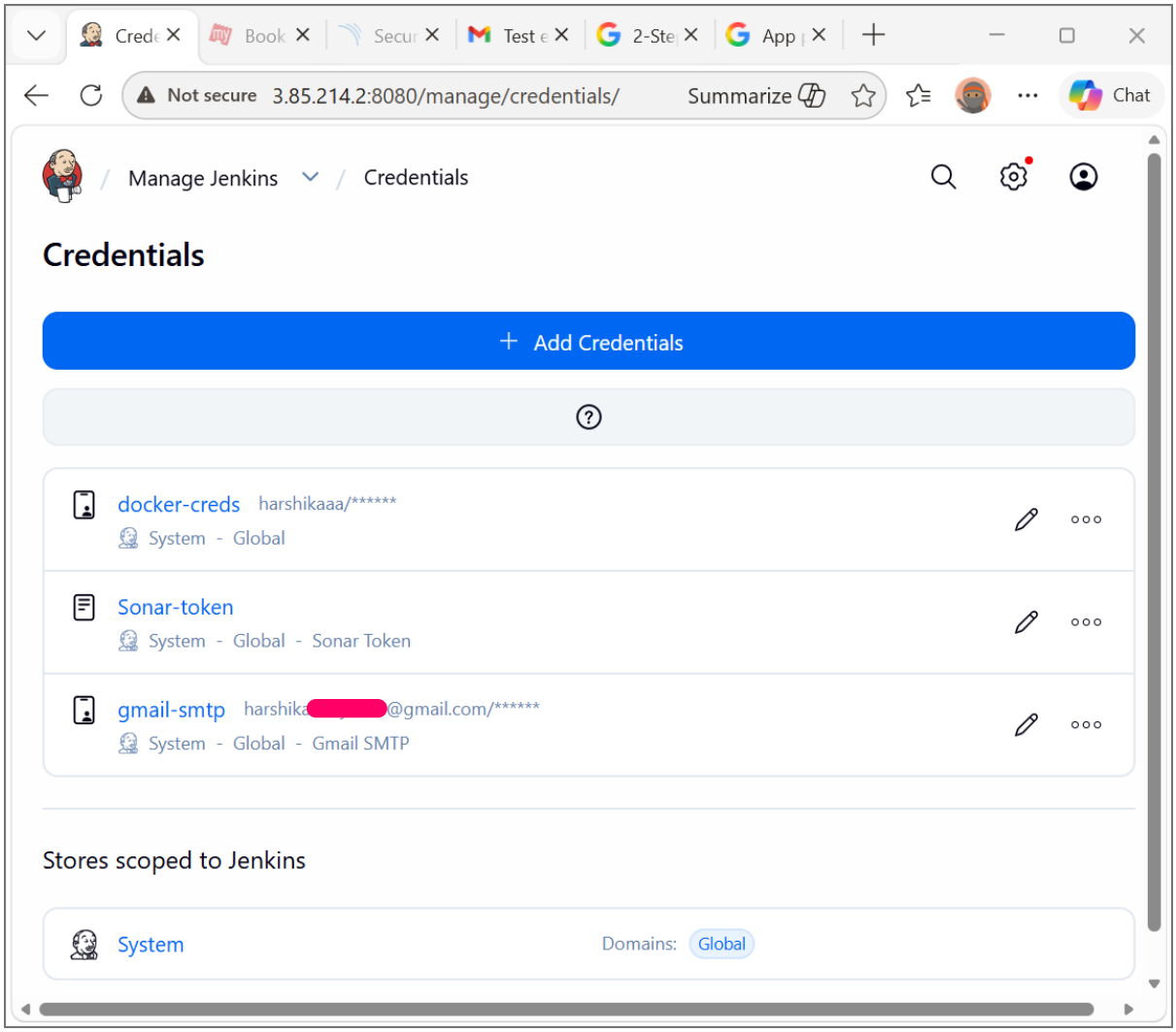
abcdefghijklmnop

This is your **SMTP password**



Step 6: Add Gmail SMTP Credentials In Jenkins





15. Configure SMTP in Jenkins

Step 6: Go to Jenkins Config

Open Jenkins:

Manage Jenkins → Configure System

Step 7: Find "Extended E-mail Notification"

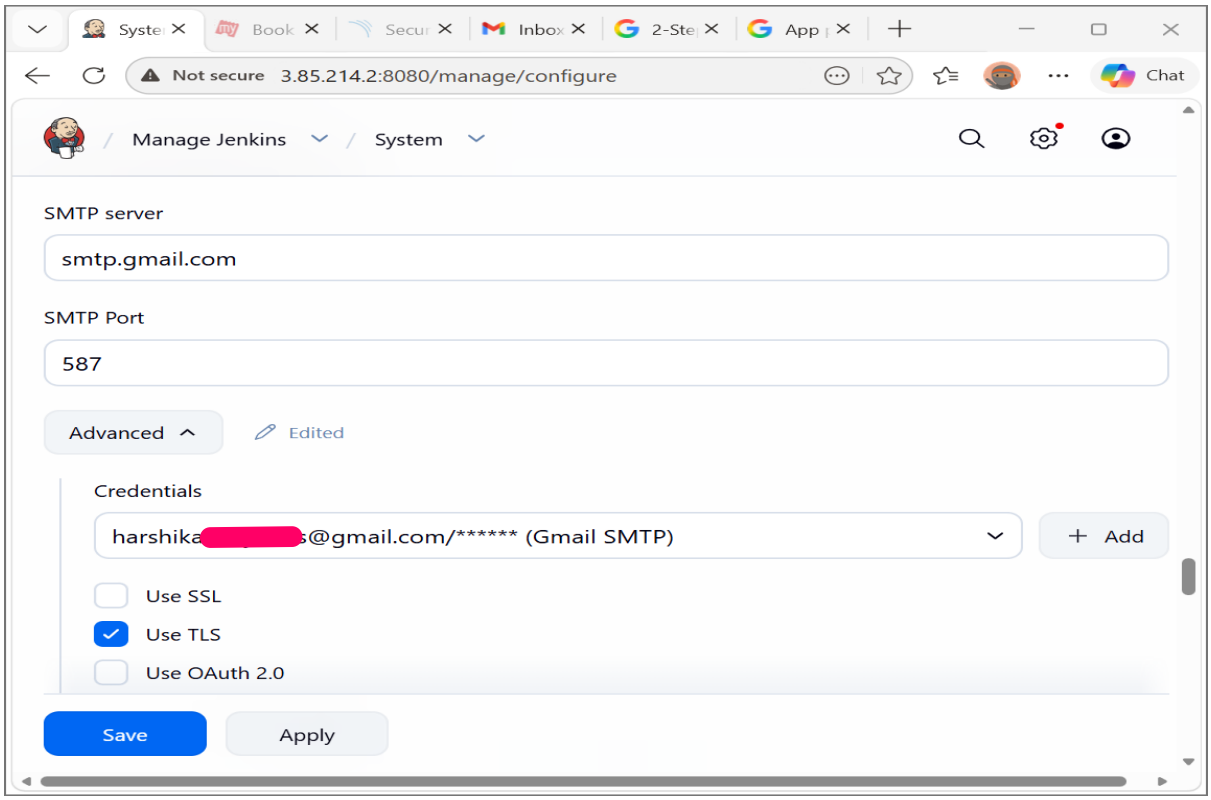
Scroll down until you see:

Extended E-mail Notification

Step 8: Fill SMTP Details

Enter EXACTLY this:

SMTP Server: smtp.gmail.com
SMTP Port: 587
Use SMTP Authentication
Username: your-email@gmail.com
Password: (paste app password WITHOUT spaces)
Use TLS



Step 9: Advanced Settings

Click **Advanced**

Make sure:

Use TLS → checked
Use SSL → NOT checked

Step 10: Test Email

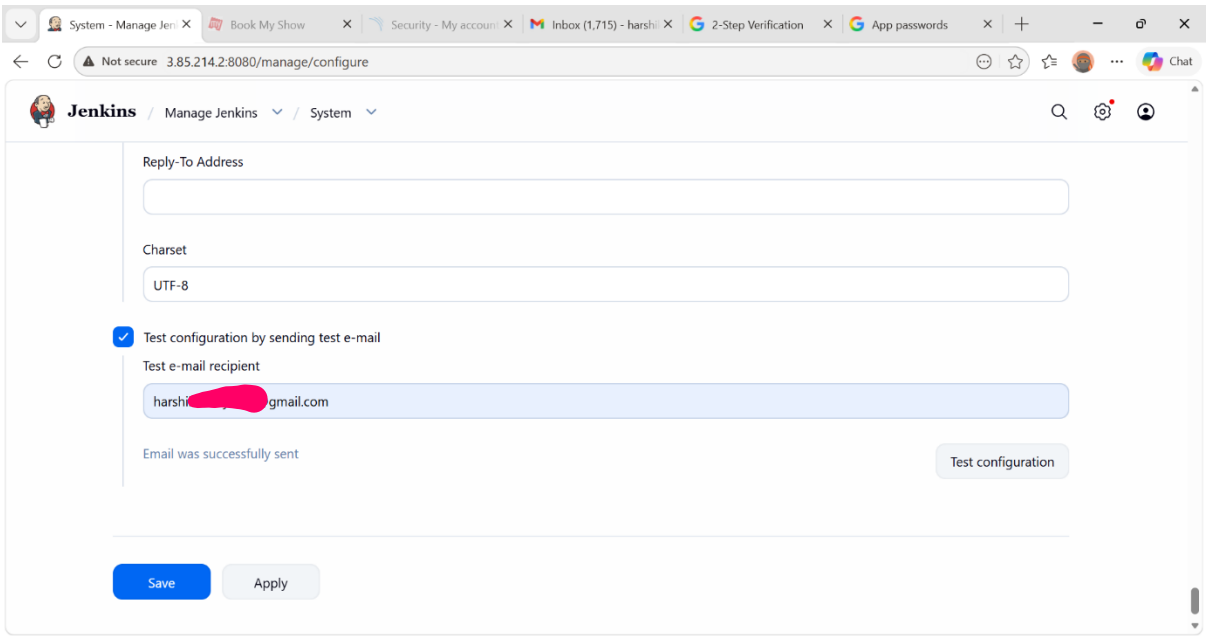
Scroll slightly down:

Find **"Test configuration by sending test e-mail"**

Enter your Gmail:

harshik[redacted]@gmail.com

Click **Test Configuration**

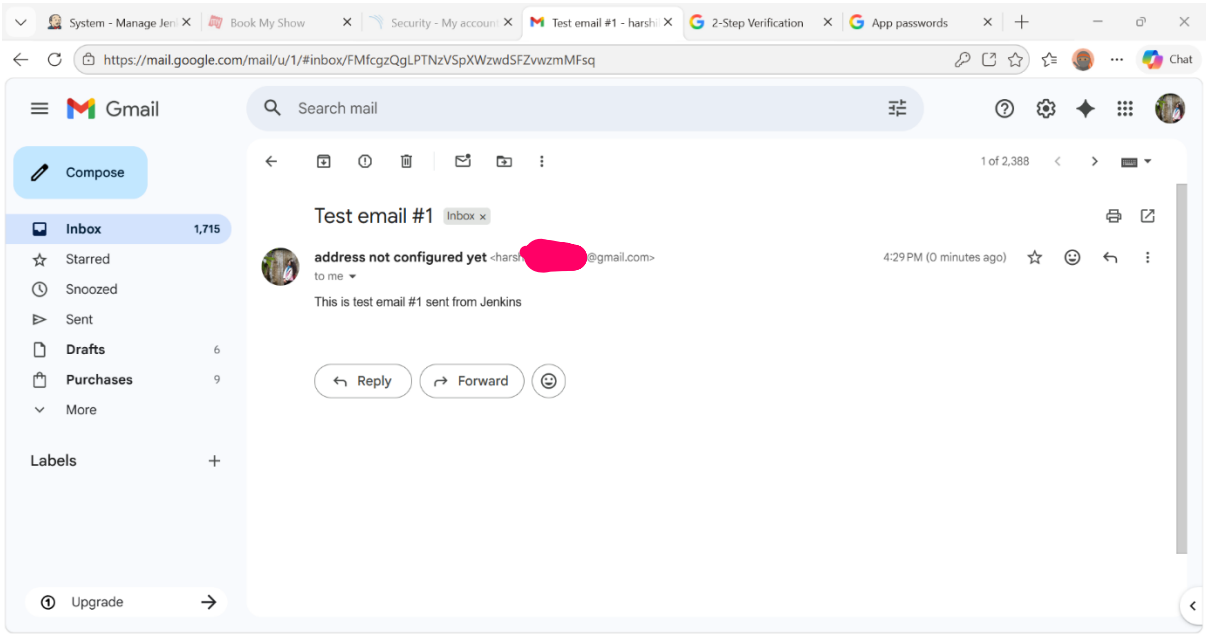


Expected Result

You should see:

Email sent successfully

Also check inbox



Now, create a webhook in Sonarqube:

Go to Webhooks Section

- Click on Administration (top menu)
- Then go to:
  Configuration → Webhooks

Click “Create”

- On the right side, click the Create button

Fill Webhook Details

Enter the following:

- Name:Jenkins

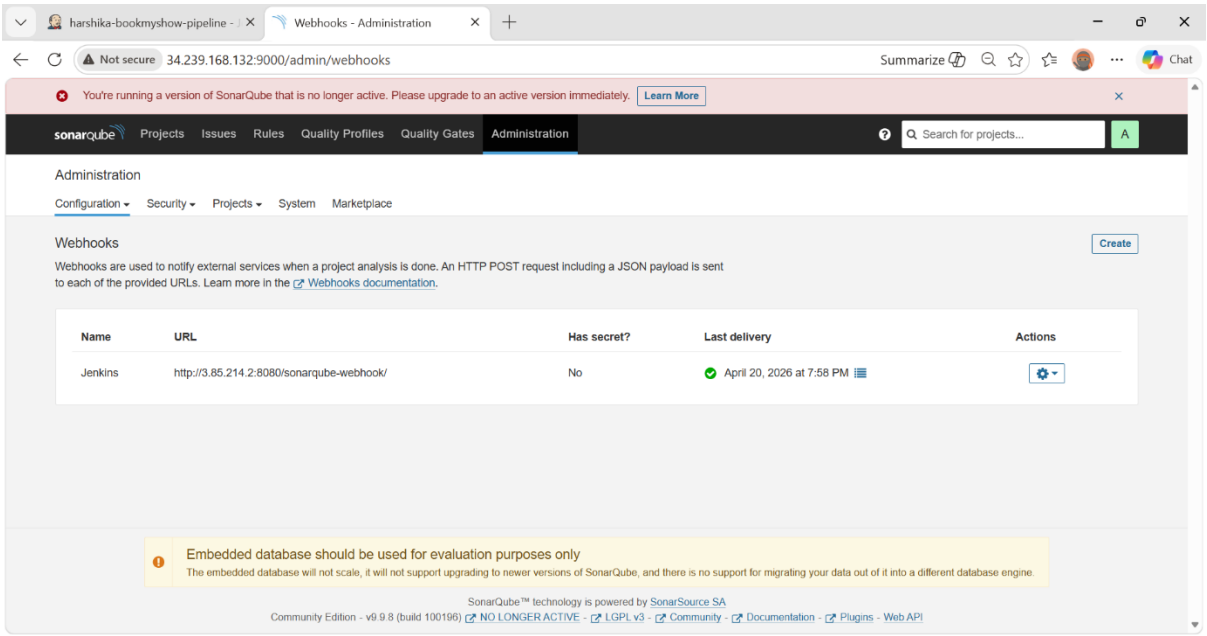
- URL:

http://<jenkins-ip>:8080/sonarqube-webhook/

http://3.85.214.2:8080/sonarqube-webhook/

Click “Create”

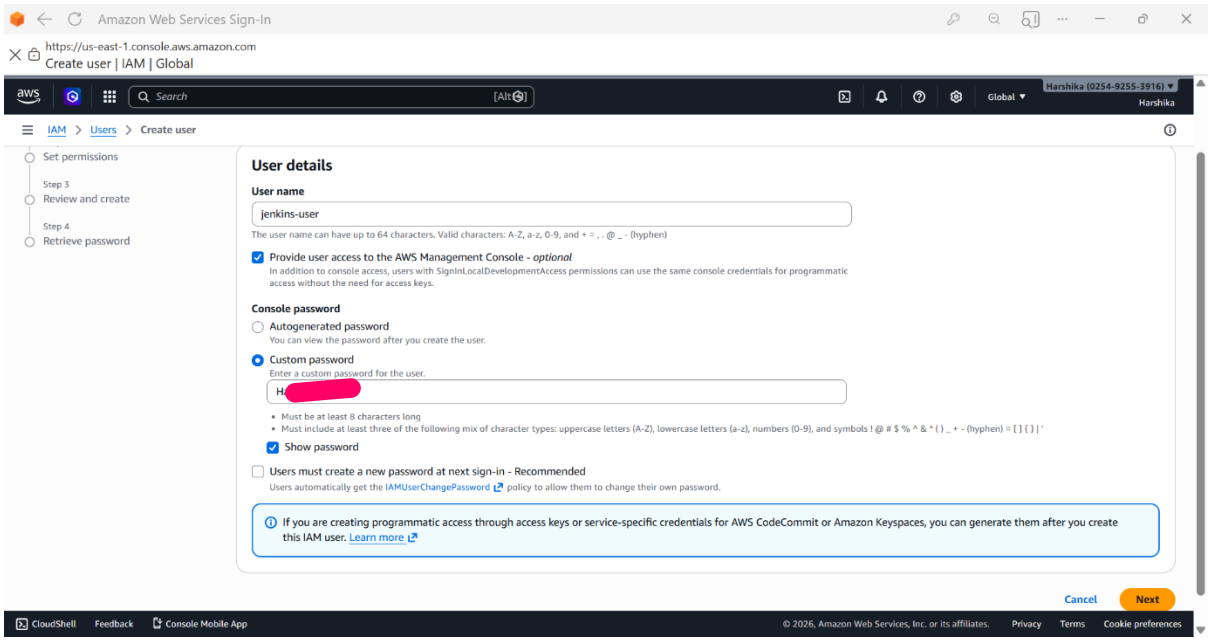
- After filling in the details, click Create



PHASE 3: EKS SETUP (STEP-BY-STEP)

STEP 1: Create IAM User

Amazon Web Services Console
→ Search **IAM**
→ Click **Users** → **Create user**



Fill details:

Username:

jenkins-user

Access:

✓ Check **Programmatic access**

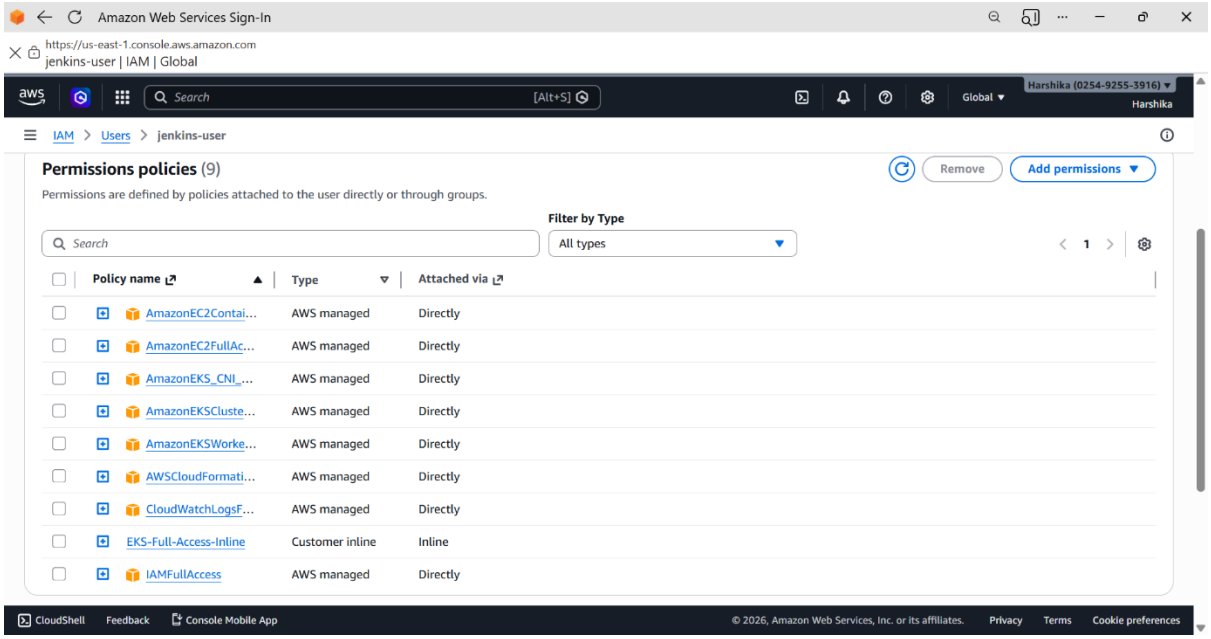
Permissions:

Select:
Attach policies directly

Add ONLY these:

- AmazonEKSClusterPolicy

- AmazonEKSWorkerNodePolicy
- AmazonEC2ContainerRegistryFullAccess
- AmazonEC2FullAccess ⚠️ (needed for EKS infra)
- CloudWatchLogsFullAccess
- AmazonEKS_CNI_Policy
- All other policy shown in below snapshot



Finish:

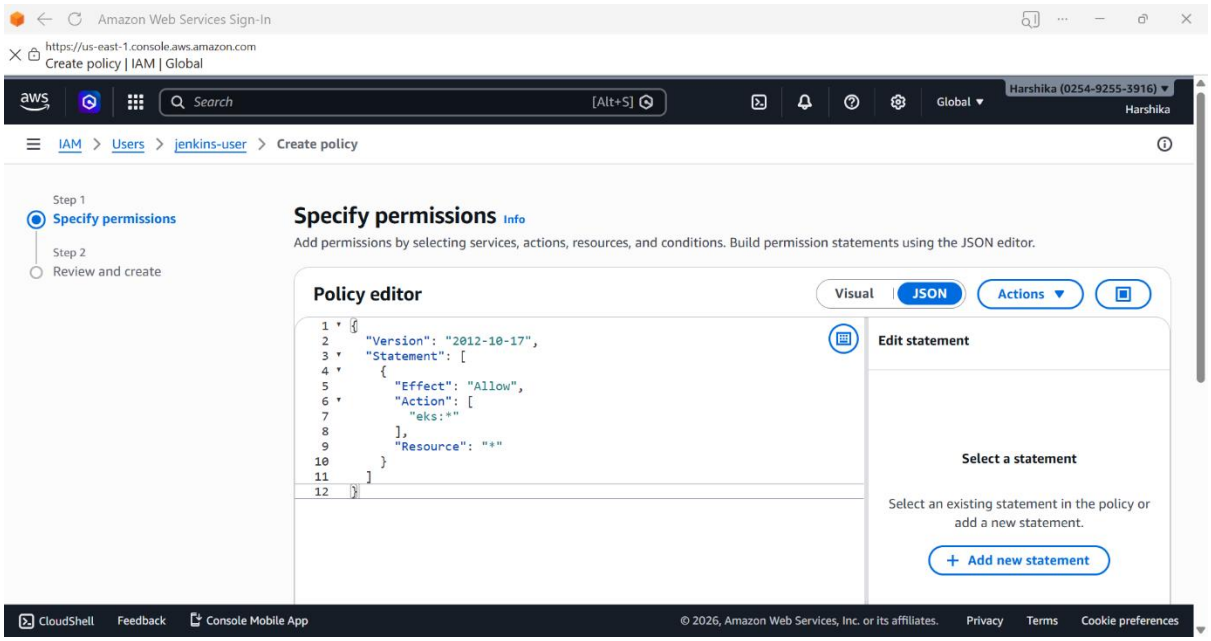
Add Inline Policy

Click:
Add permissions → Create inline policy

Choose the JSON tab

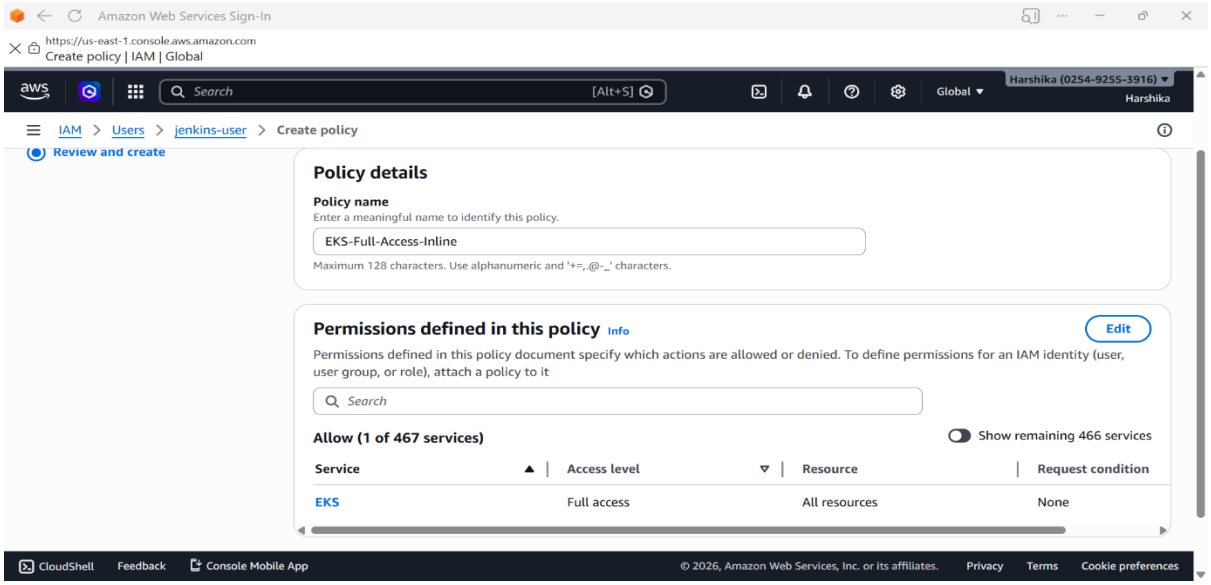
```
{
 "Version": "2012-10-17",
 "Statement": [
  {
   "Effect": "Allow",
   "Action": [
    "eks:*"
```

```
    ],
    "Resource": "*"
   }
  ]
}
```

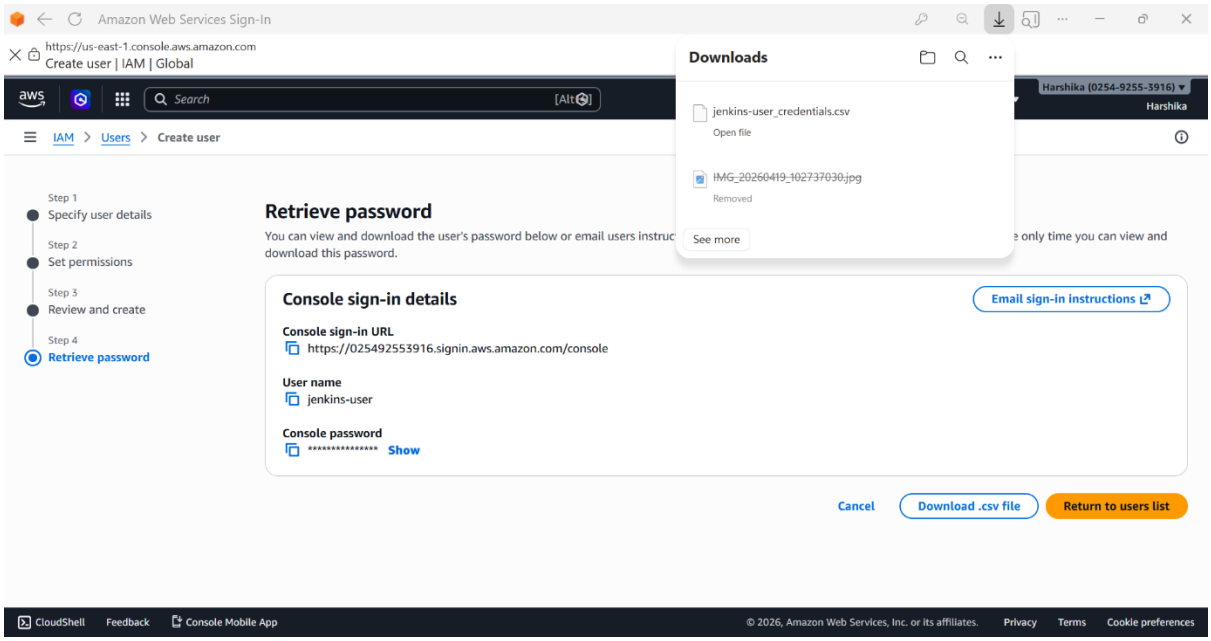


Then:

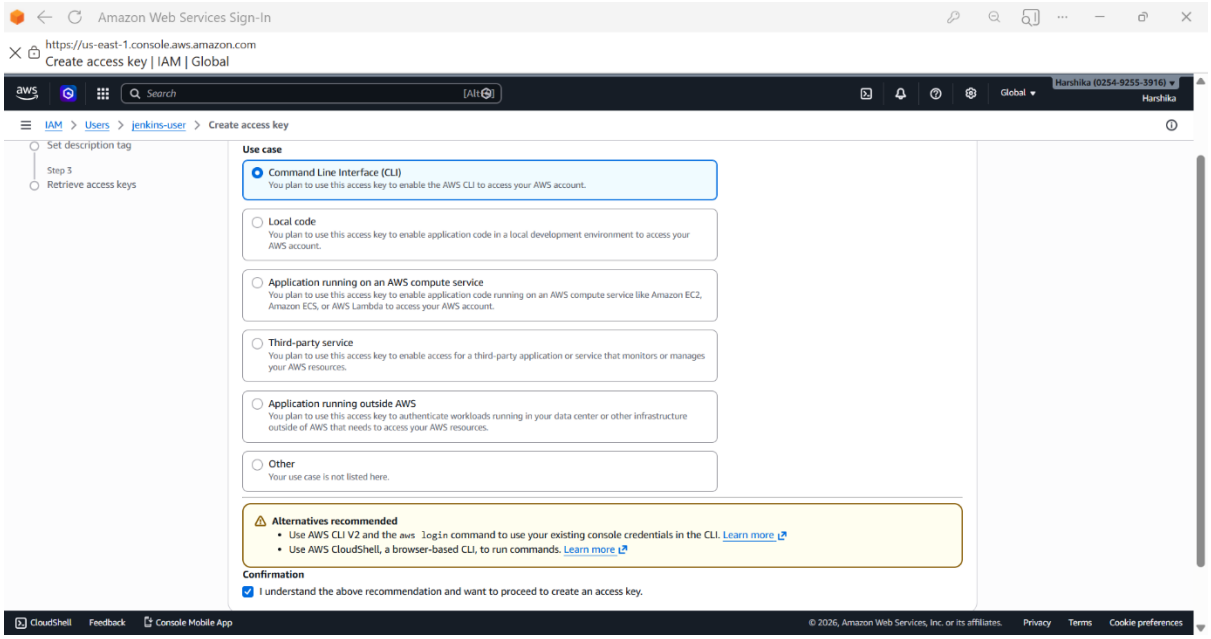
- **Click Next**
- **Name: EKS-Full-Access-Inline**
- **Click Create policy**



Click **Create User**



Click on jenkins-user and create access key:



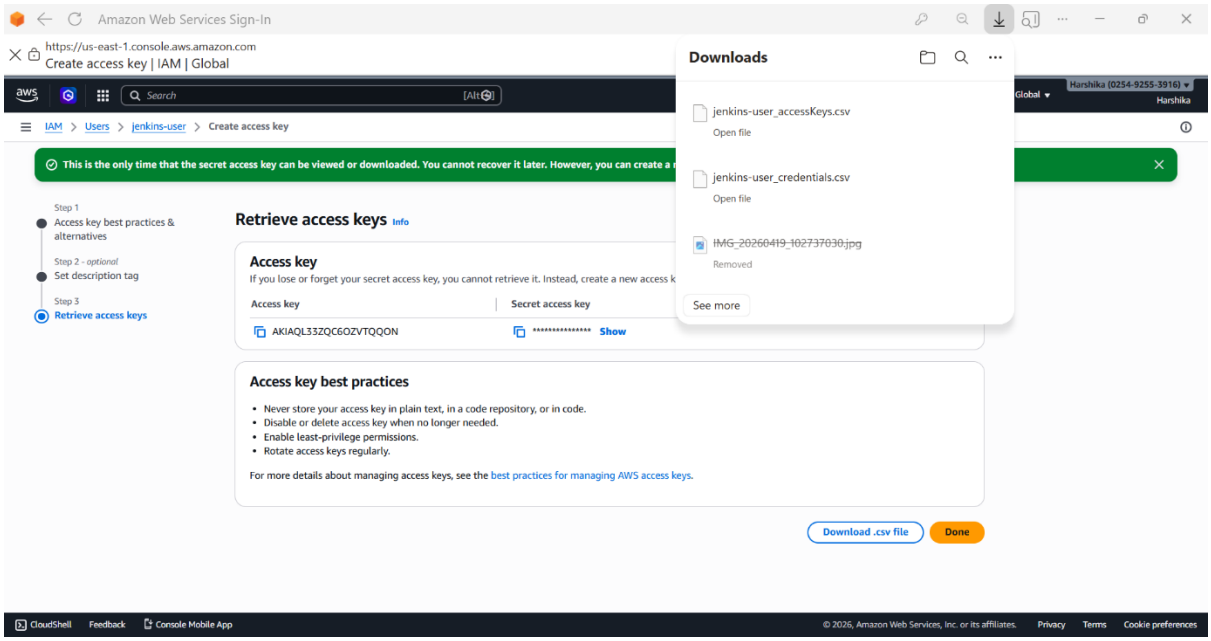
IMPORTANT

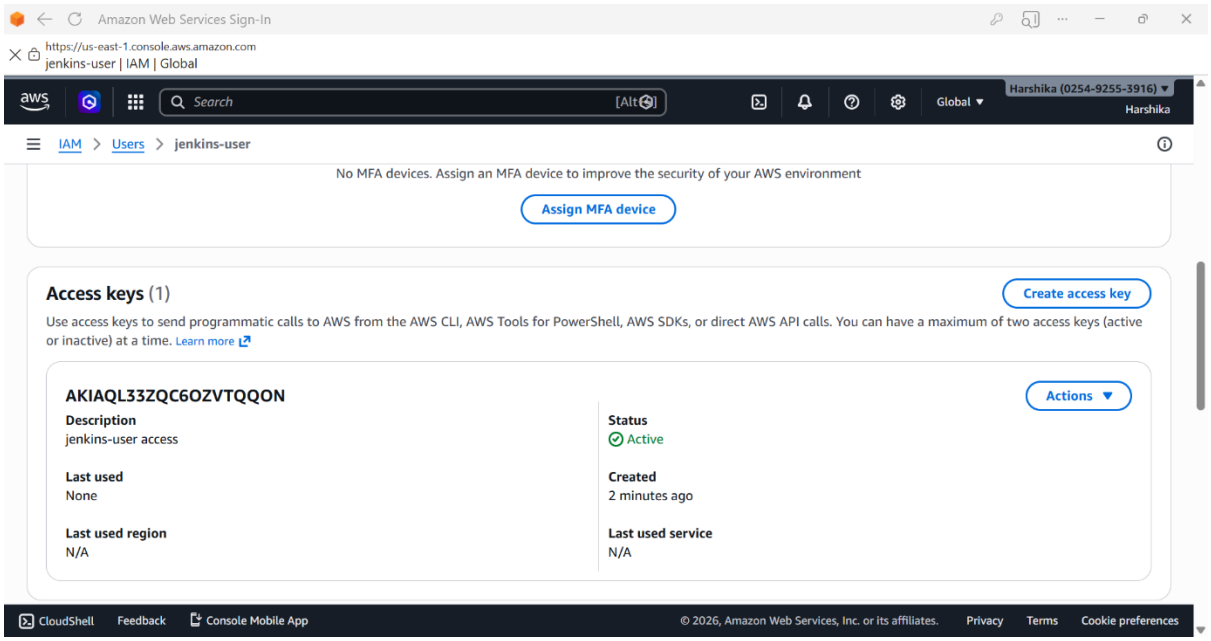
After creation, you will see:

✓ Access Key

✓ Secret Key

Download or copy them NOW (you won't see them again)





Step 2: Add AWS Credentials in Jenkins

Go to:

Jenkins → Manage Jenkins → Credentials → Global → Add Credentials

Add:

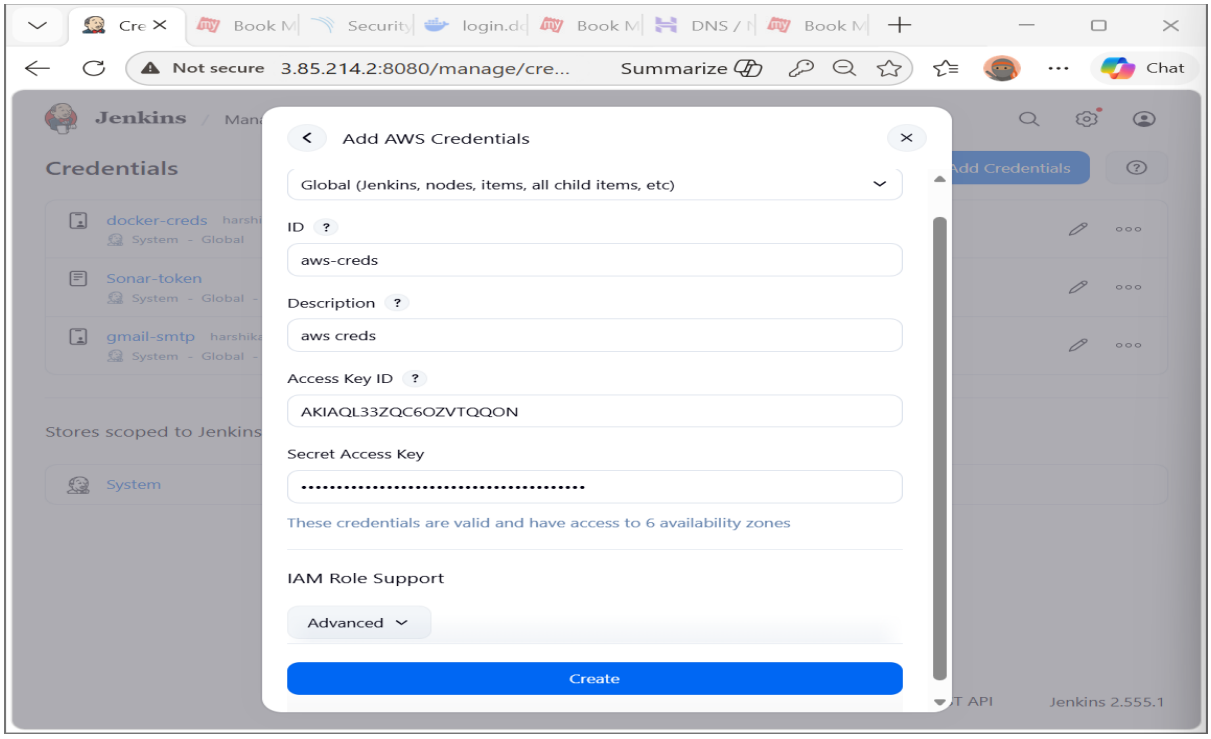
Type

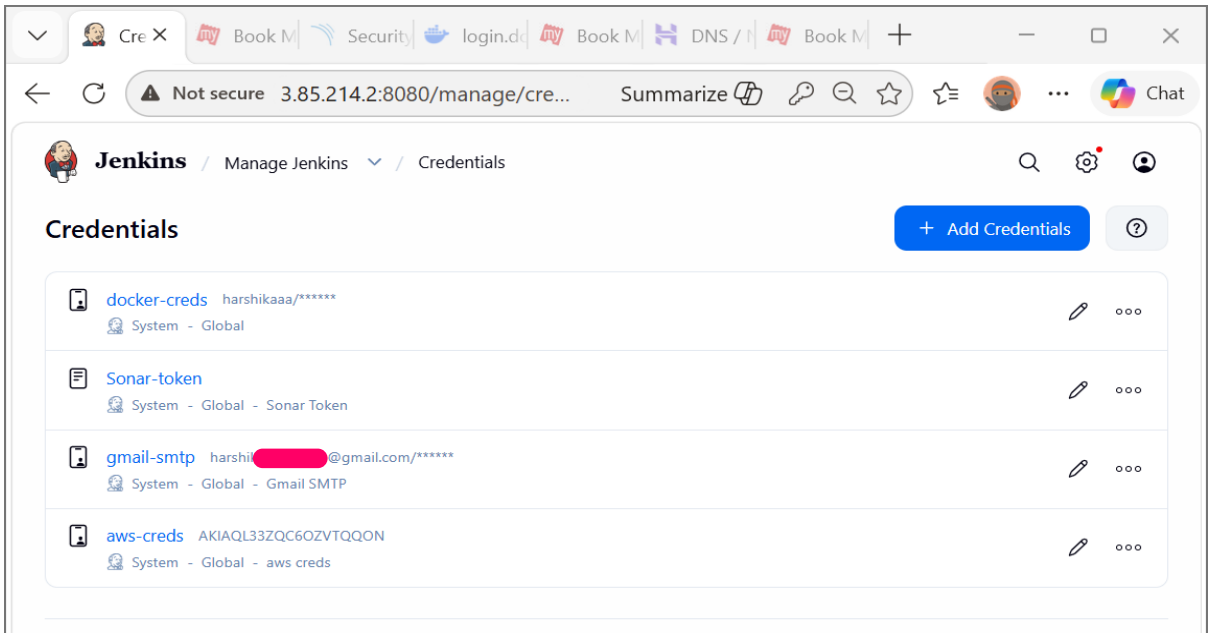
AWS Credentials

Fill:

- **Access Key ID** → (copy)
- **Secret Access Key** → (copy)
- **ID** → aws-creds

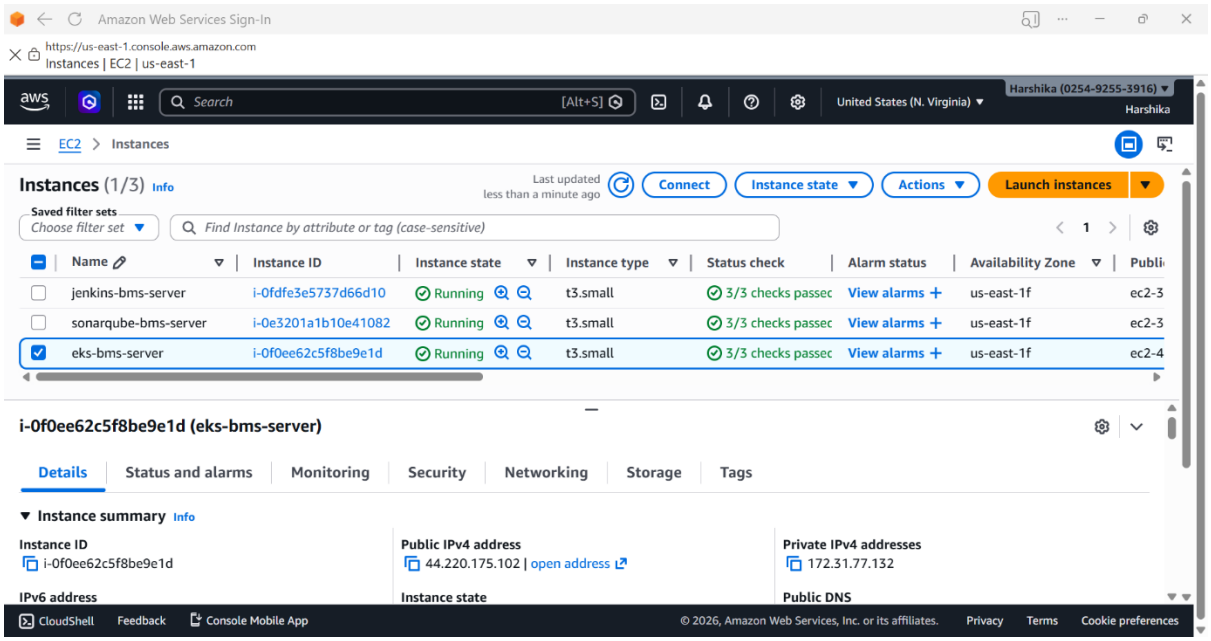
Click **Save**





STEP 3: Create a new server for the EKS Cluster (Launch EC2 (t3.small))

Launch the instance with the same configuration as the Jenkins-server



Connect via SSH from PowerShell

1. Install AWS CLI v2 (Ubuntu 24.04)

curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o "awscliv2.zip"

sudo apt update
sudo apt install unzip -y

unzip awscliv2.zip

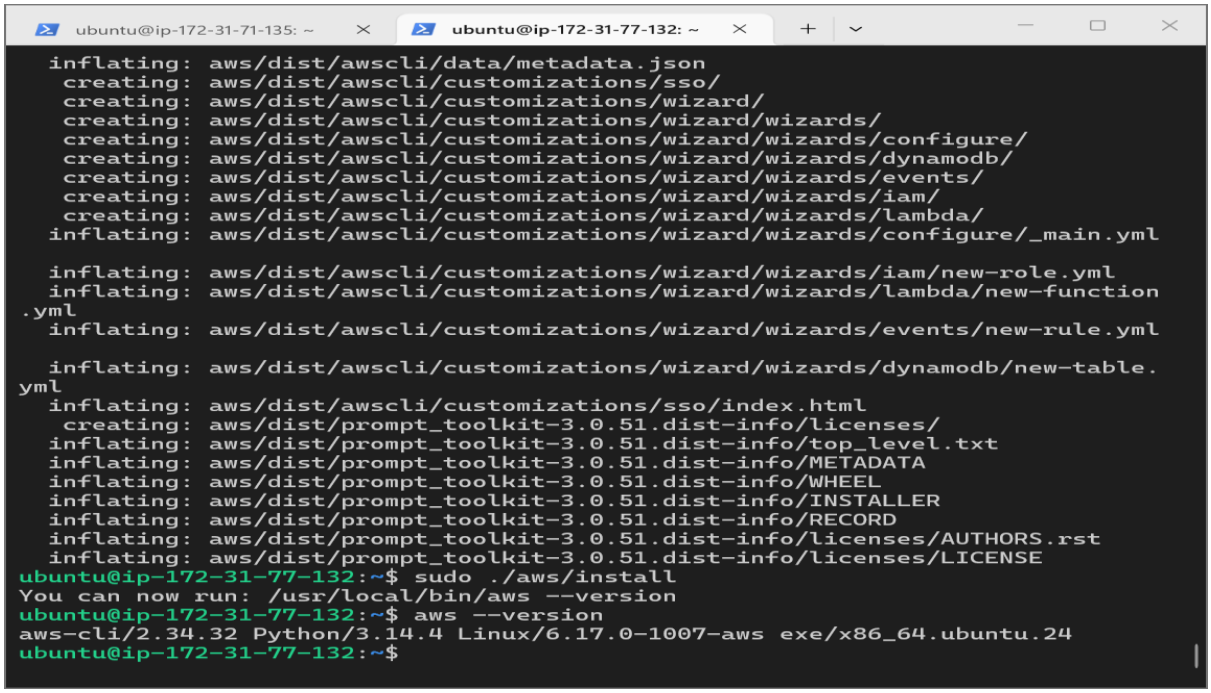
sudo ./aws/install

VERIFY

aws --version

Expected:

aws-cli/2.x.x Python/3.x ...



2. Configure AWS

aws configure

1. Access Key

Access Key ID

2. Secret Key

Secret Access Key

3. Region

us-east-1

(important — same region we'll use for EKS)

4. Output format

json

AFTER THIS (VERIFY)

Run:

aws sts get-caller-identity

```
ubuntu@ip-172-31-77-132:~$ sudo ./aws/install
You can now run: /usr/local/bin/aws --version
ubuntu@ip-172-31-77-132:~$ aws --version
aws-cli/2.34.32 Python/3.14.4 Linux/6.17.0-1007-aws exe/x86_64.ubuntu.24
ubuntu@ip-172-31-77-132:~$ aws configure

Tip: You can deliver temporary credentials to the AWS CLI using your AWS Conso
le session by running the command 'aws login'.

AWS Access Key ID [None]: AKIAQL33ZQC6OZVTQQON
AWS Secret Access Key [None]: XplmMRmZPPujc1dl3Dinx2I6OMvgztk+Un85sHCo
Default region name [None]: us-east-1
Default output format [None]: json
ubuntu@ip-172-31-77-132:~$ aws sts get-caller-identity
{
    "UserId": "AIDAQL33ZQC6KT5MMGK4A",
    "Account": "025492553916",
    "Arn": "arn:aws:iam::025492553916:user/jenkins-user"
}
ubuntu@ip-172-31-77-132:~$
```

STEP 4: Install eksctl

curl --silent --location "https://github.com/weaveworks/eksctl/releases/latest/download/eksctl_Linux_amd64.tar.gz" | tar xz -C /tmp

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

eksctl version

```
ubuntu@ip-172-31-77-132:~$ curl --silent --location "https://github.com/weavew
orks/eksctl/releases/latest/download/eksctl_Linux_amd64.tar.gz" | tar xz -C /t
mp
ubuntu@ip-172-31-77-132:~$ sudo mv /tmp/eksctl /usr/local/bin
ubuntu@ip-172-31-77-132:~$ eksctl version
0.225.0
ubuntu@ip-172-31-77-132:~$
```

STEP 5: Install kubectl

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

chmod +x kubectl
sudo mv kubectl /usr/local/bin/

kubectl version --client

```
ubuntu@ip-172-31-77-132:~$ curl -LO "https://dl.k8s.io/release/$(curl -L -s ht
tps://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl"
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Curren
t
                                 Dload  Upload   Total   Spent    Left  Speed
100 55.8M  100 55.8M    0     0   140M      0 --:--:-- --:--:-- --:--:--  140M
ubuntu@ip-172-31-77-132:~$ chmod +x kubectl
sudo mv kubectl /usr/local/bin/
ubuntu@ip-172-31-77-132:~$ kubectl version --client
Client Version: v1.35.4
Kustomize Version: v5.7.1
ubuntu@ip-172-31-77-132:~$
```

STEP 6: Create EKS Cluster (optimised)

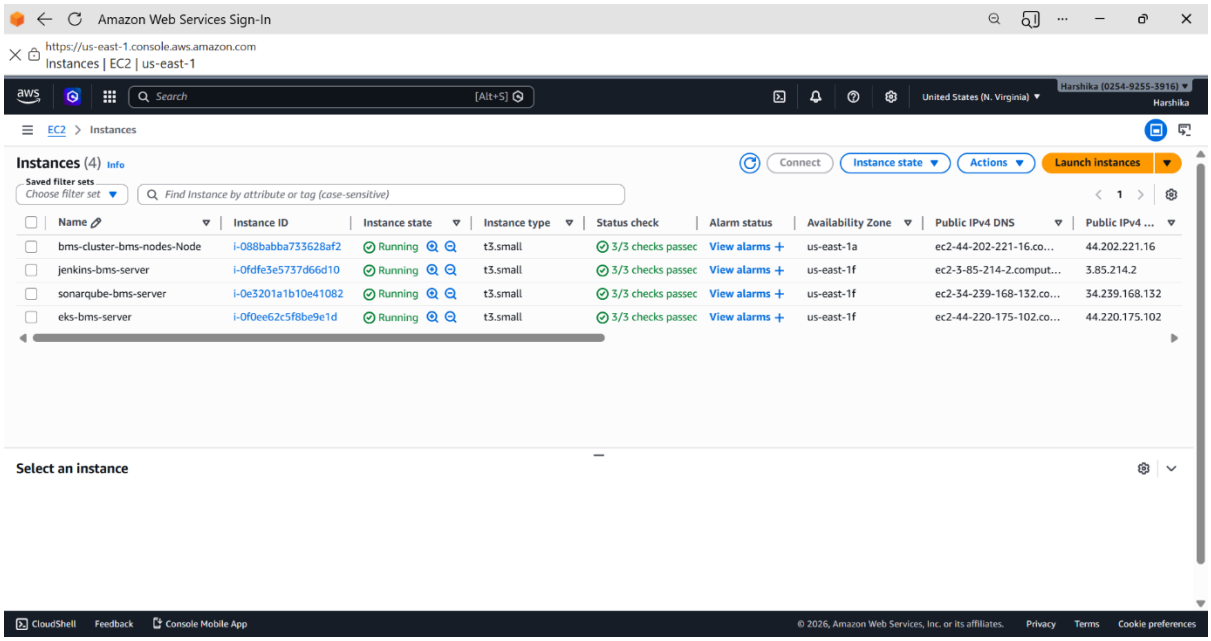
Run this:

eksctl create cluster \
--name bms-cluster \
--region us-east-1 \
--nodegroup-name bms-nodes \
--node-type t3.small \
--nodes 1 \
--nodes-min 1 \
--nodes-max 1

after completion

```
Windows PowerShell   ubuntu@ip-172-31-77-132: ~
2026-04-19 15:53:27 [ℹ]  deploying stack "eksctl-bms-cluster-nodegroup-bms-nodes"
2026-04-19 15:53:27 [ℹ]  waiting for CloudFormation stack "eksctl-bms-cluster-nodegroup-bms-nodes"
2026-04-19 15:53:57 [ℹ]  waiting for CloudFormation stack "eksctl-bms-cluster-nodegroup-bms-nodes"
2026-04-19 15:54:34 [ℹ]  waiting for CloudFormation stack "eksctl-bms-cluster-nodegroup-bms-nodes"
2026-04-19 15:56:12 [ℹ]  waiting for CloudFormation stack "eksctl-bms-cluster-nodegroup-bms-nodes"
2026-04-19 15:56:12 [ℹ]  no tasks
2026-04-19 15:56:12 [✔]  created 0 nodegroup(s) in cluster "bms-cluster"
2026-04-19 15:56:12 [ℹ]  nodegroup "bms-nodes" has 1 node(s)
2026-04-19 15:56:12 [ℹ]  node "ip-192-168-5-81.ec2.internal" is ready
2026-04-19 15:56:12 [ℹ]  waiting for at least 1 node(s) to become ready in "bms-nodes"
2026-04-19 15:56:12 [ℹ]  nodegroup "bms-nodes" has 1 node(s)
2026-04-19 15:56:12 [ℹ]  node "ip-192-168-5-81.ec2.internal" is ready
2026-04-19 15:56:12 [✔]  created 1 managed nodegroup(s) in cluster "bms-cluster"
2026-04-19 15:56:12 [ℹ]  checking security group configuration for all nodegroups
2026-04-19 15:56:12 [ℹ]  all nodegroups have up-to-date cloudformation templates
ubuntu@ip-172-31-77-132:~$ kubectl get nodes
NAME                           STATUS   ROLES    AGE     VERSION
ip-192-168-5-81.ec2.internal   Ready    <none>   2m20s   v1.34.6-eks-bbe087e
ubuntu@ip-172-31-77-132:~$ kubectl get pods -A
NAMESPACE     NAME                                  READY   STATUS    RESTARTS   AGE
kube-system   aws-node-kg5s9                        2/2     Running   0          3m10s
kube-system   coredns-7d58d485c9-64hj5              1/1     Running   0          15m
kube-system   coredns-7d58d485c9-ts2r7              1/1     Running   0          15m
kube-system   kube-proxy-8xjt6                      1/1     Running   0          3m10s
kube-system   metrics-server-6f49c4bc6c-7djx7       1/1     Running   0          14m
kube-system   metrics-server-6f49c4bc6c-n8dd8       1/1     Running   0          14m
ubuntu@ip-172-31-77-132:~$
```

Check your EC2 Console for the node



Step 7: Install VPC CNI (MOST IMPORTANT)

Run:

```
eksctl create addon \
--name vpc-cni \
--cluster bms-cluster \
--region us-east-1 \
--force
```

Step 8: Install CoreDNS

```
eksctl create addon \
--name coredns \
--cluster bms-cluster \
--region us-east-1
```

Step 9: Install kube-proxy

```
eksctl create addon \
--name kube-proxy \
--cluster bms-cluster \
--region us-east-1
```

Step 10: (optional but good)

eksctl create addon \
--name metrics-server \
--cluster bms-cluster \
--region us-east-1

Wait 2–3 minutes

Now check:

Eksctl get cluster

```
ubuntu@ip-172-31-77-132:~$ eksctl get cluster
NAME            REGION          EKSCTL CREATED
bms-cluster     us-east-1       True
ubuntu@ip-172-31-77-132:~$ aws eks describe-cluster --name bms-cluster --regio
n us-east-1 --query "cluster.status"
"ACTIVE"
ubuntu@ip-172-31-77-132:~$
```

kubectl get nodes

```
Windows PowerShell   ubuntu@ip-172-31-77-132: ~
2026-04-19 15:53:27 [i]  deploying stack "eksctl-bms-cluster-nodegroup-bms-nodes"
2026-04-19 15:53:27 [i]  waiting for CloudFormation stack "eksctl-bms-cluster-nodegroup-bms-nodes"
2026-04-19 15:53:57 [i]  waiting for CloudFormation stack "eksctl-bms-cluster-nodegroup-bms-nodes"
2026-04-19 15:54:34 [i]  waiting for CloudFormation stack "eksctl-bms-cluster-nodegroup-bms-nodes"
2026-04-19 15:56:12 [i]  waiting for CloudFormation stack "eksctl-bms-cluster-nodegroup-bms-nodes"
2026-04-19 15:56:12 [i]  no tasks
2026-04-19 15:56:12 [✔]  created 0 nodegroup(s) in cluster "bms-cluster"
2026-04-19 15:56:12 [i]  nodegroup "bms-nodes" has 1 node(s)
2026-04-19 15:56:12 [i]  node "ip-192-168-5-81.ec2.internal" is ready
2026-04-19 15:56:12 [i]  waiting for at least 1 node(s) to become ready in "bms-nodes"
2026-04-19 15:56:12 [i]  nodegroup "bms-nodes" has 1 node(s)
2026-04-19 15:56:12 [i]  node "ip-192-168-5-81.ec2.internal" is ready
2026-04-19 15:56:12 [✔]  created 1 managed nodegroup(s) in cluster "bms-cluster"
2026-04-19 15:56:12 [i]  checking security group configuration for all nodegroups
2026-04-19 15:56:12 [i]  all nodegroups have up-to-date cloudformation templates
ubuntu@ip-172-31-77-132:~$ kubectl get nodes
NAME                          STATUS   ROLES    AGE     VERSION
ip-192-168-5-81.ec2.internal   Ready    <none>   2m20s   v1.34.6-eks-bbe087e
ubuntu@ip-172-31-77-132:~$ kubectl get pods -A
NAMESPACE     NAME                              READY   STATUS    RESTARTS   AGE
kube-system   aws-node-kg5s9                    2/2     Running   0          3m10s
kube-system   coredns-7d58d485c9-64hj5          1/1     Running   0          15m
kube-system   coredns-7d58d485c9-ts2r7          1/1     Running   0          15m
kube-system   kube-proxy-8xjt6                  1/1     Running   0          3m10s
kube-system   metrics-server-6f49c4bc6c-7djx7   1/1     Running   0          14m
kube-system   metrics-server-6f49c4bc6c-n8dd8   1/1     Running   0          14m
ubuntu@ip-172-31-77-132:~$
```

4. Log in to your Jenkins EC2 server

Then install Docker:

sudo apt update
sudo apt install docker.io -y
sudo systemctl start docker
sudo systemctl enable docker

Give Jenkins permission to use Docker

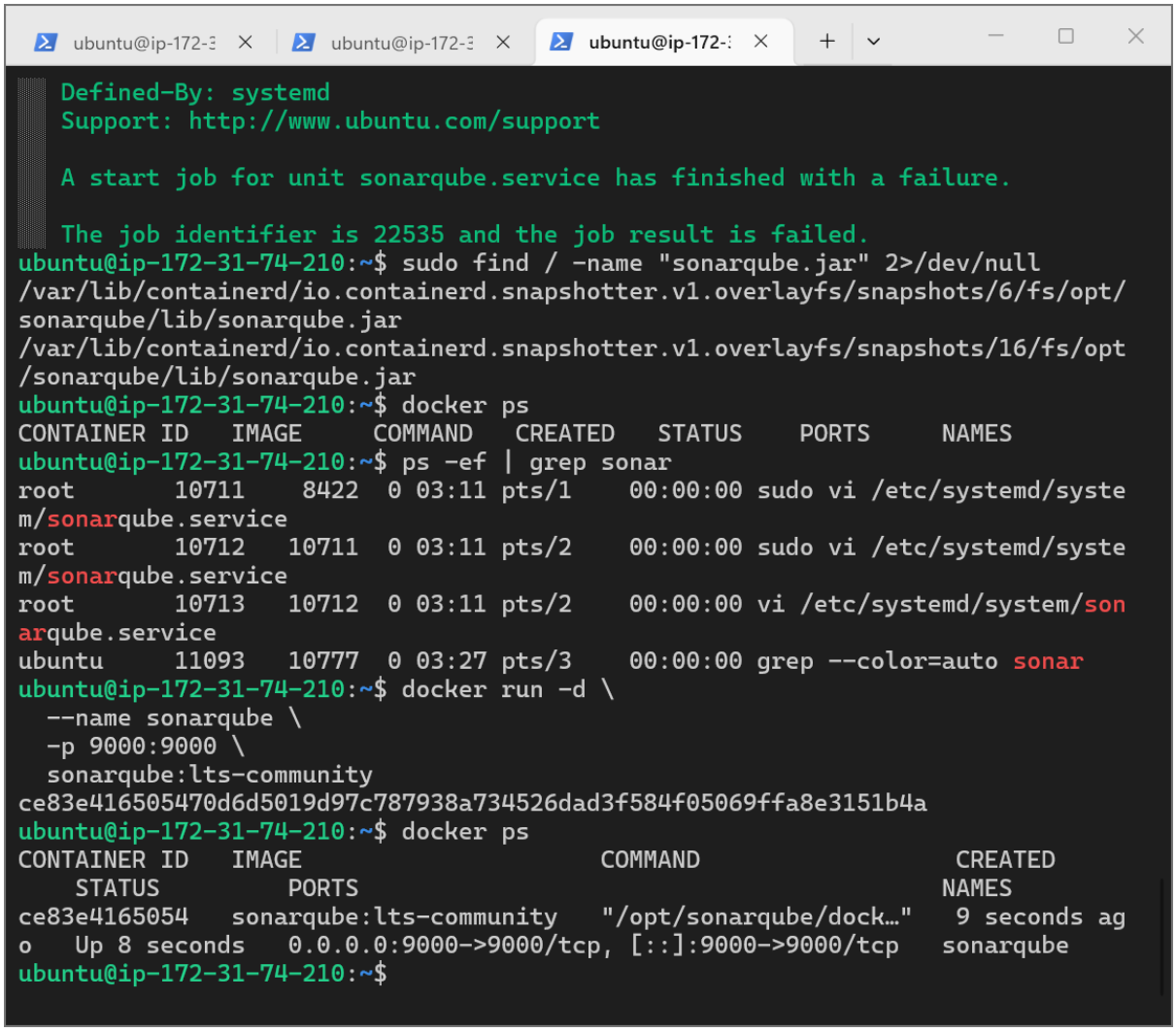
sudo usermod -aG docker jenkins
sudo usermod -aG docker ubuntu

Then restart:

sudo systemctl restart jenkins

Verify

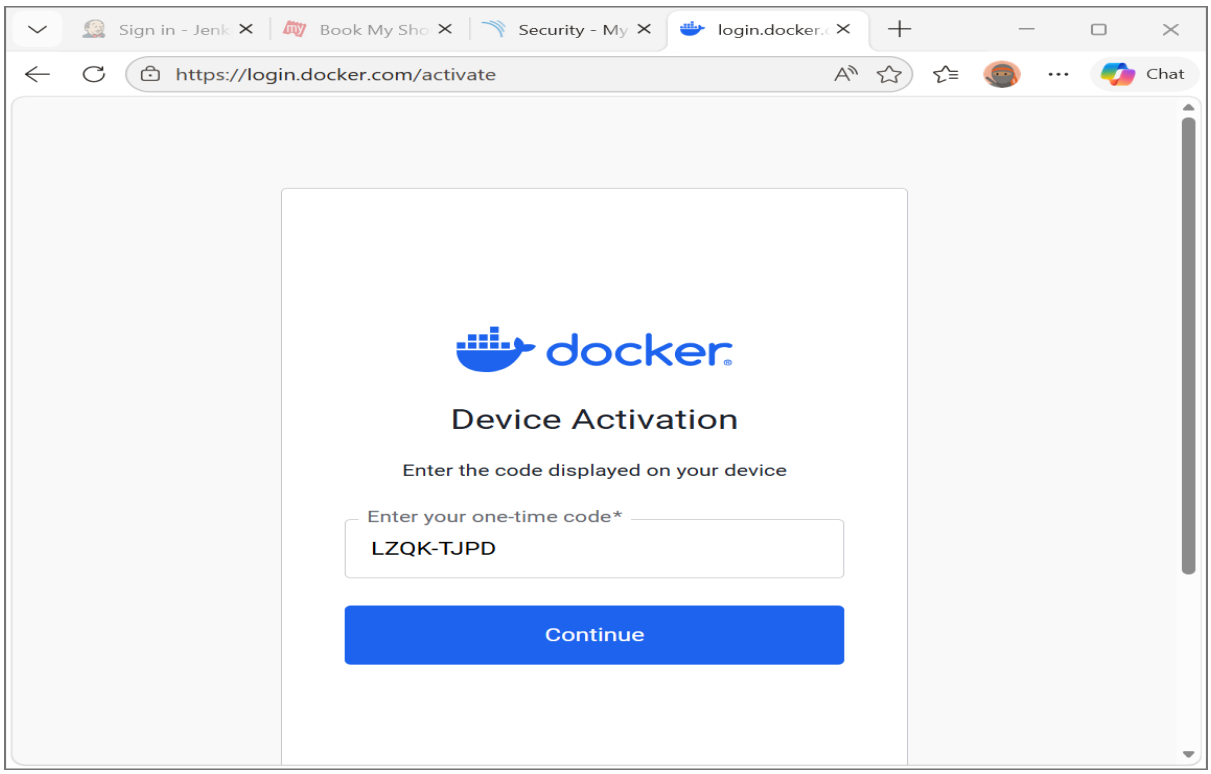
docker ps

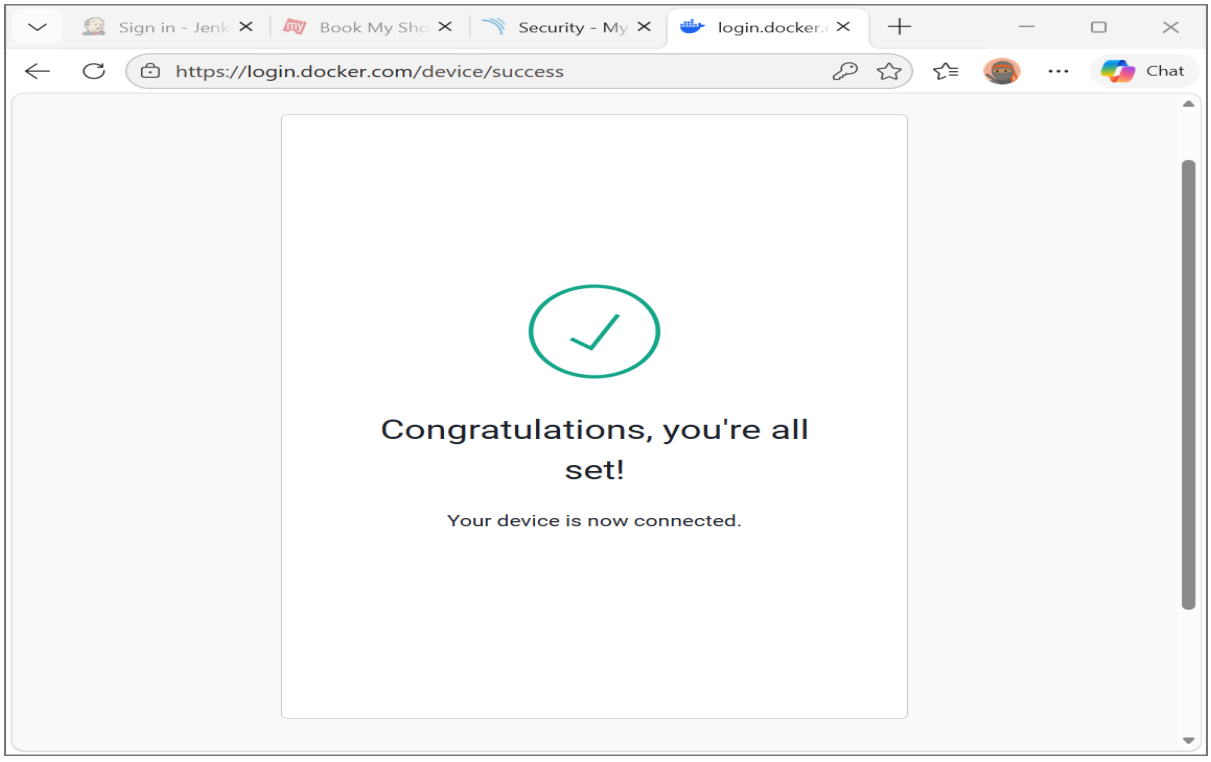


Step 1: Push your image to DockerHub

Login

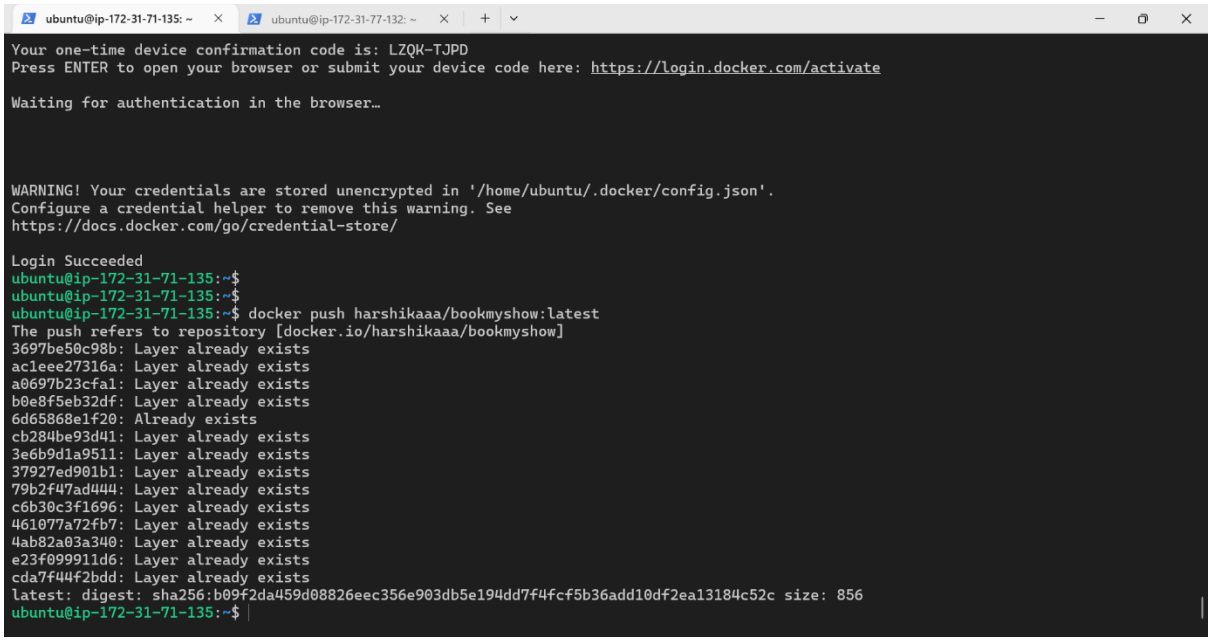
docker login





Push image

docker push harshikaaa/bookmyshow:latest



Now deploy to EKS (final step)

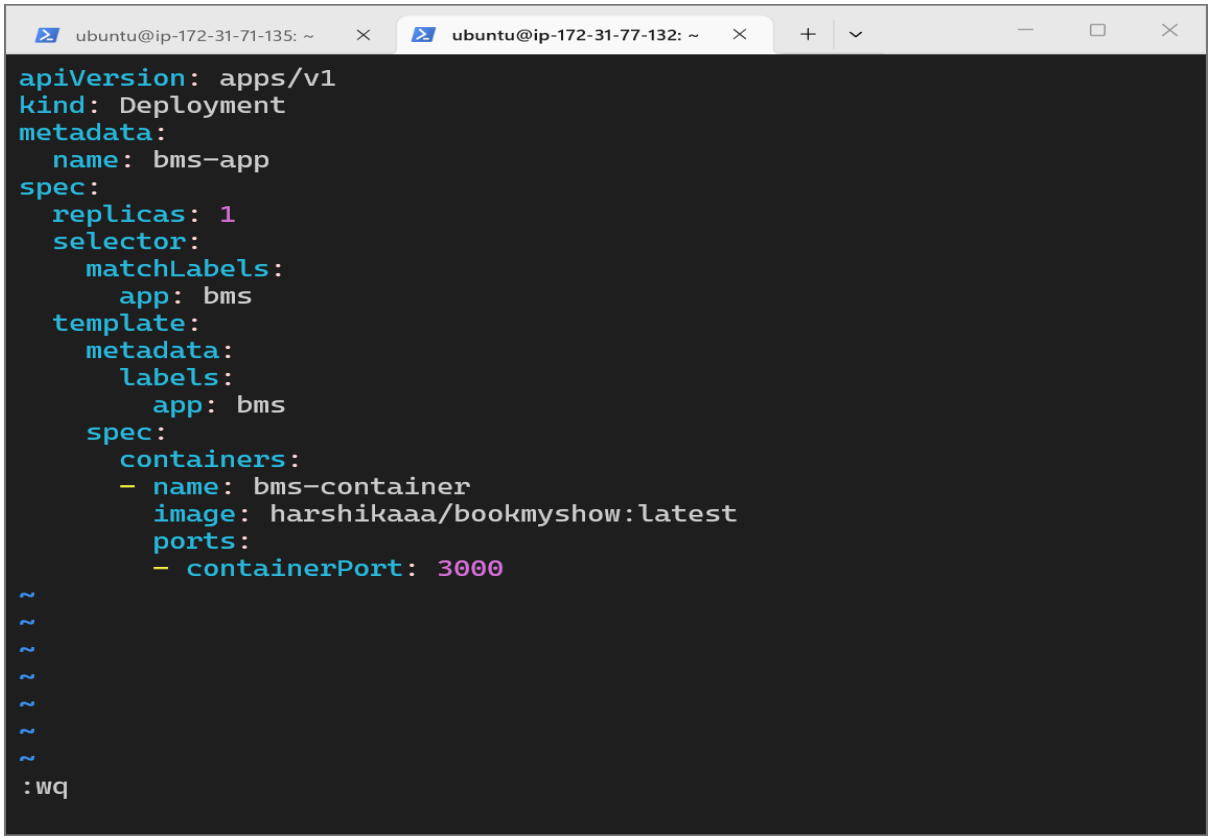
Go to your **EKS server, mine-** (ip-172-31-77-132)

Step 1: Create deployment.yaml (vi)

vi deployment.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: bms-app
spec:
  replicas: 1
  selector:
    matchLabels:
      app: bms
  template:
    metadata:
```

```
    labels:
      app: bms
  spec:
    containers:
    - name: bms-container
      image: harshikaaa/bookmyshow:latest
      ports:
      - containerPort: 3000
```



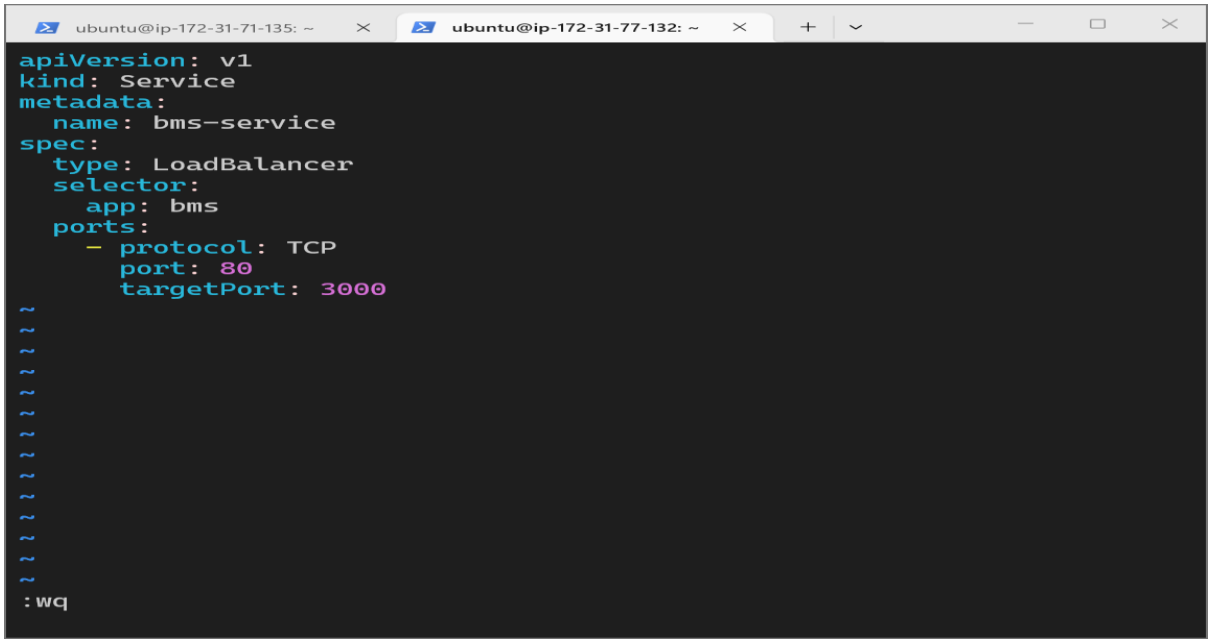
Step 2: Create service.yaml

vi service.yaml

Press i and paste:

```
apiVersion: v1
kind: Service
metadata:
  name: bms-service
spec:
```

```
  type: LoadBalancer
 selector:
   app: bms
 ports:
   - protocol: TCP
     port: 80
     targetPort: 3000
```

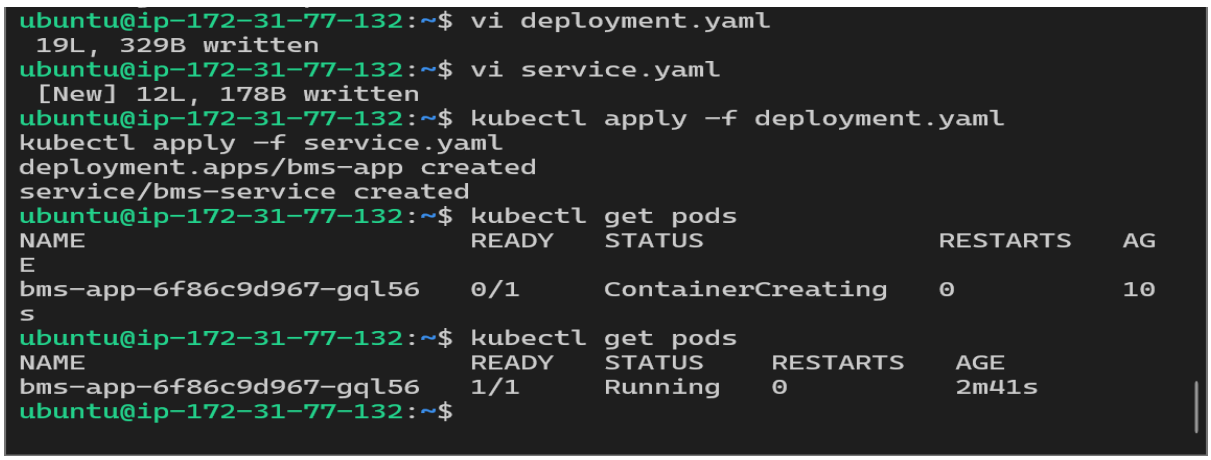


Step 3: Apply configs

kubectl apply -f deployment.yaml
kubectl apply -f service.yaml

Step 4: Check pods

kubectl get pods



Step 5: Get public URL

kubectl get svc

```
ubuntu@ip-172-31-77-132:~$ kubectl get svc
NAME          TYPE           CLUSTER-IP       EXTERNAL-IP
  PORT(S)        AGE
bms-service   LoadBalancer   10.100.247.173   ae99c65dd23334066a26ab997e574b43-971348613.us-east-1.elb.amazonaws.com
  80:32664/TCP   4m3s
kubernetes    ClusterIP      10.100.0.1       <none>
  443/TCP        159m
ubuntu@ip-172-31-77-132:~$
```

What you'll see

NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S)
bms-service LoadBalancer 10.x.x.x <pending> 80:xxxxx/TCP

Wait 1–3 minutes until:

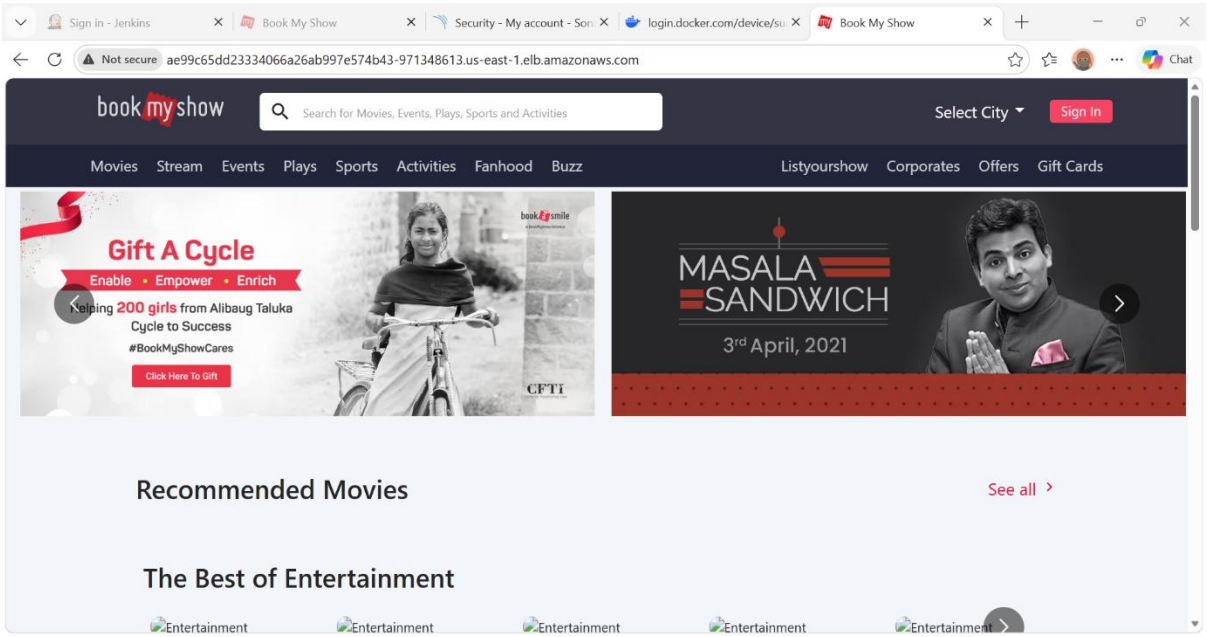
EXTERNAL-IP → something.elb.amazonaws.com

Final step

Open in browser:

http://<EXTERNAL-IP>

http://ae99c65dd23334066a26ab997e574b43-971348613.us-east-1.elb.amazonaws.com



5. Configure DNS in Hostinger

Mine LoadBalancer URL

Use this (your EKS endpoint):

ae99c65dd23334066a26ab997e574b43-971348613.us-east-1.elb.amazonaws.com

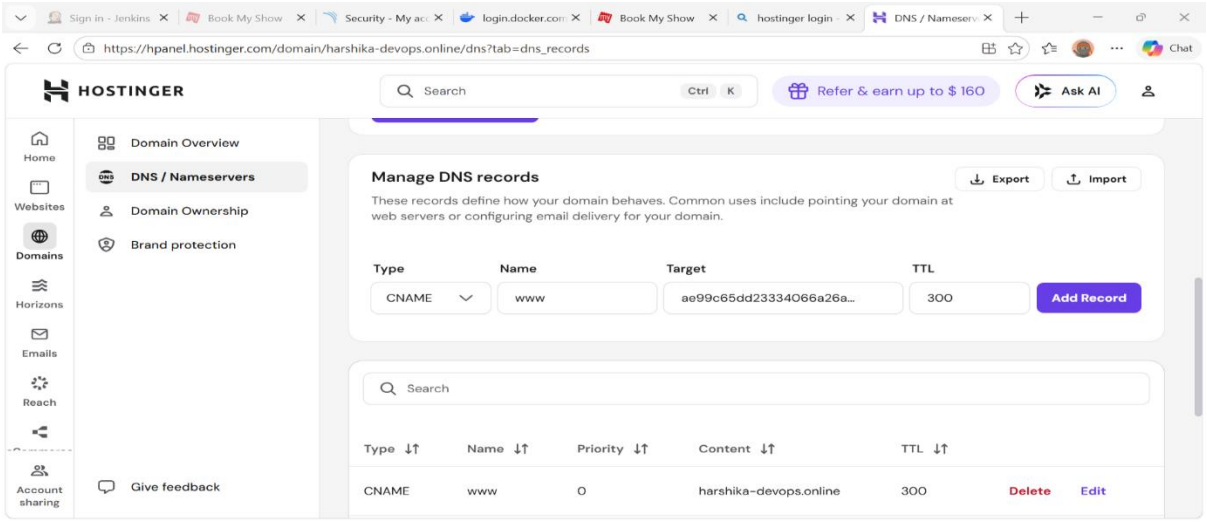
Step 1: Open Hostinger DNS

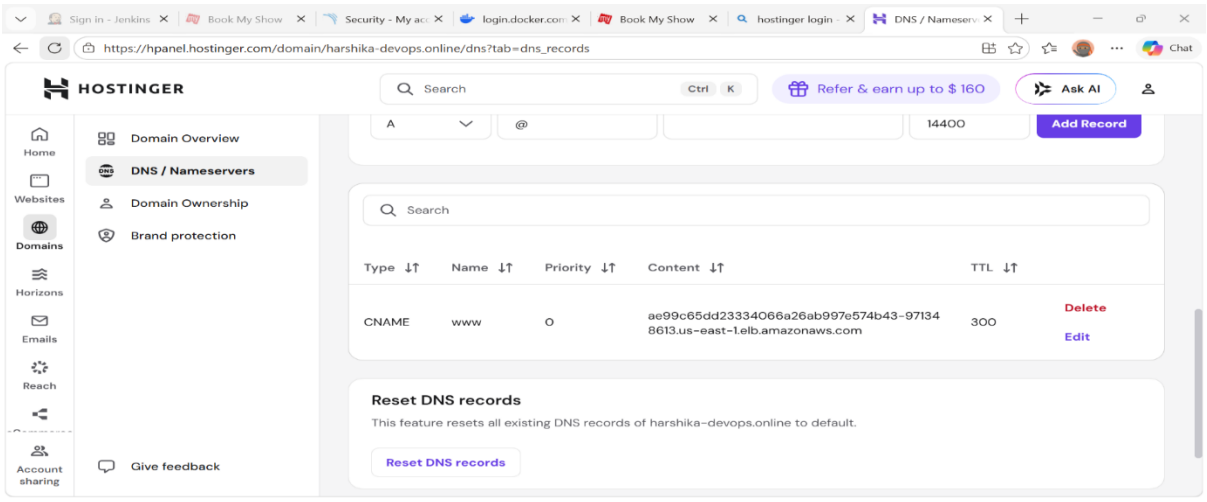
1. Log in to **Hostinger**
2. Go to:
    - **Domains**
    - Click: harshika-devops.online
3. Open:
   DNS / Nameservers / Manage DNS

Step 2: Add DNS Record

Add a CNAME record

Field	**Value**
Type	CNAME
Name	www
Target	ae99c65dd23334066a26ab997e574b43-971348613.us-east-1.elb.amazonaws.com

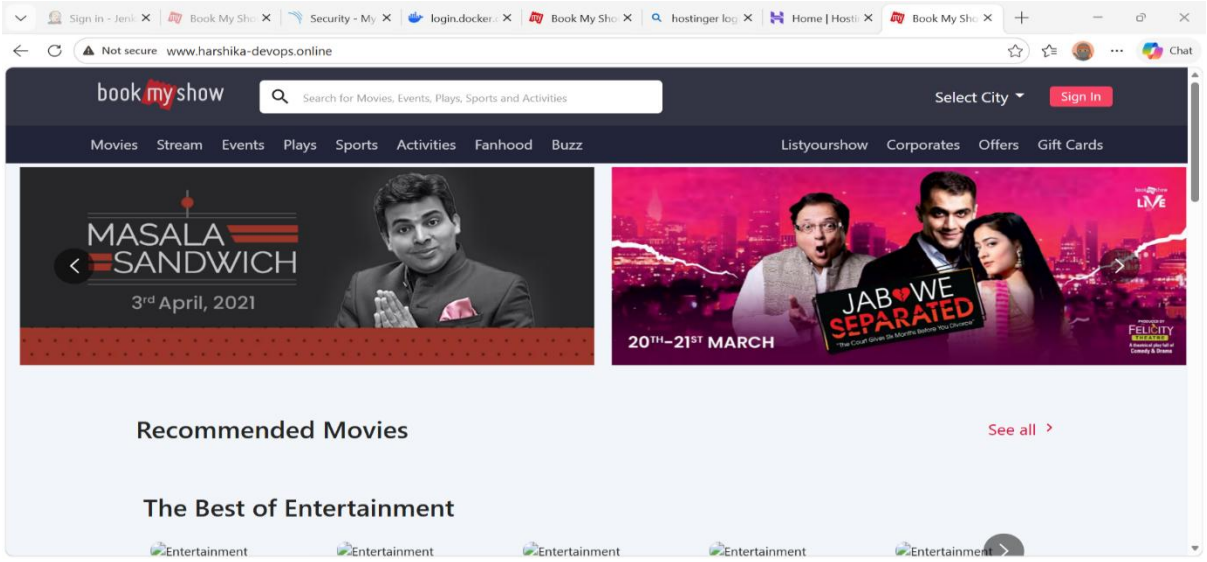




Step 6: Test

Open:

http://www.harshika-devops.online



6. Enable webhook trigger in Jenkins

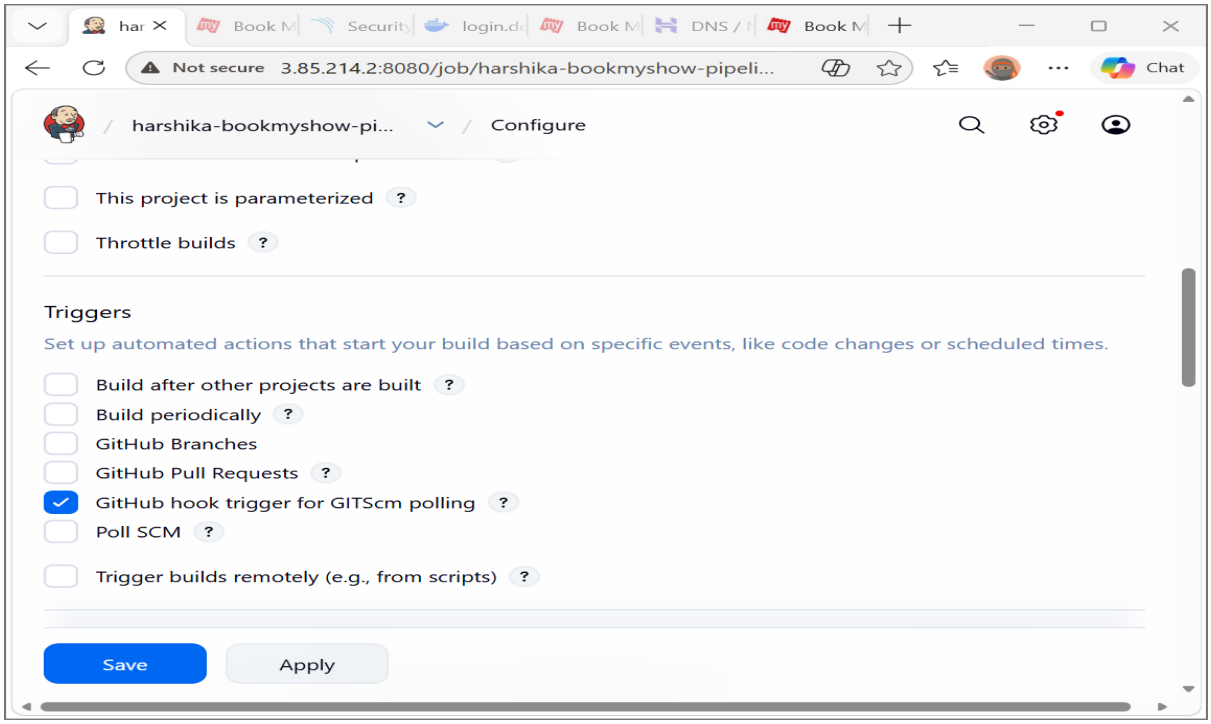
In Jenkins:

1. Open your job → Harshika-bookmyshow-pipeline
2. Click → **Configure**
3. Scroll to → **Build Triggers**

Enable:

GitHub hook trigger for GITScm polling

Click **Save**



7. Add a webhook in GitHub

Go to your repo in GitHub:

harshika369/Book-My-Show

Then:

1. Click → **Settings**
2. Click → **Webhooks**

3. Click → **Add webhook**

Fill these fields EXACTLY

Payload URL:

http://<your-jenkins-ip>:8080/github-webhook/

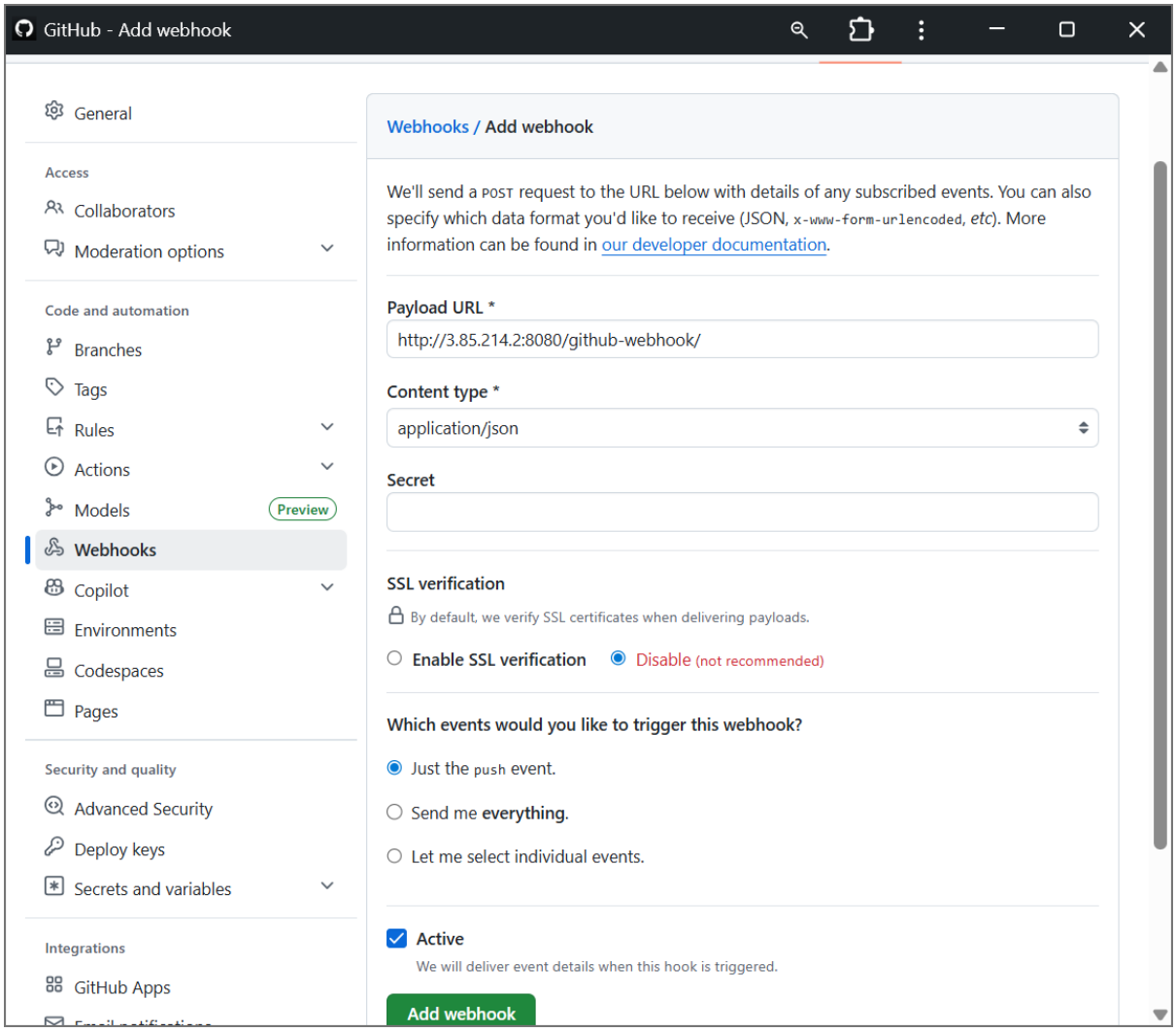
Content type:

application/json

Events:

Select:

Just the push event



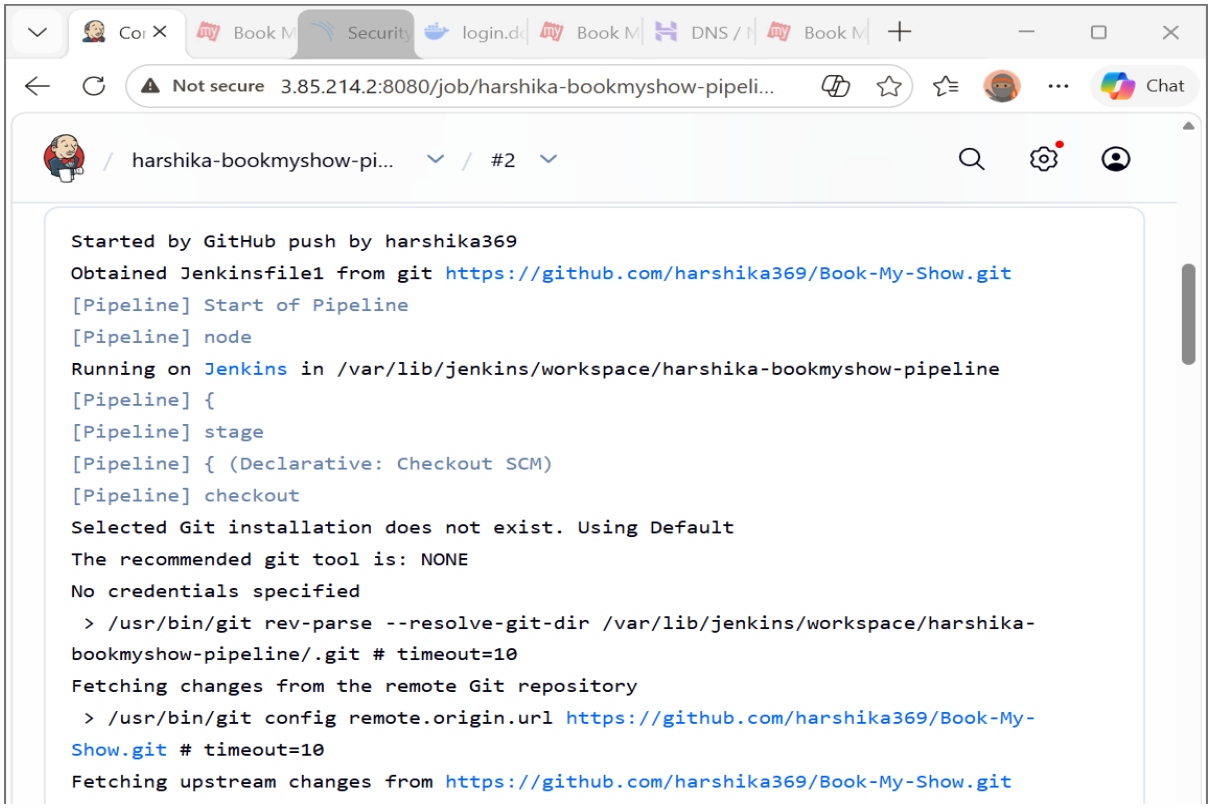
Click:

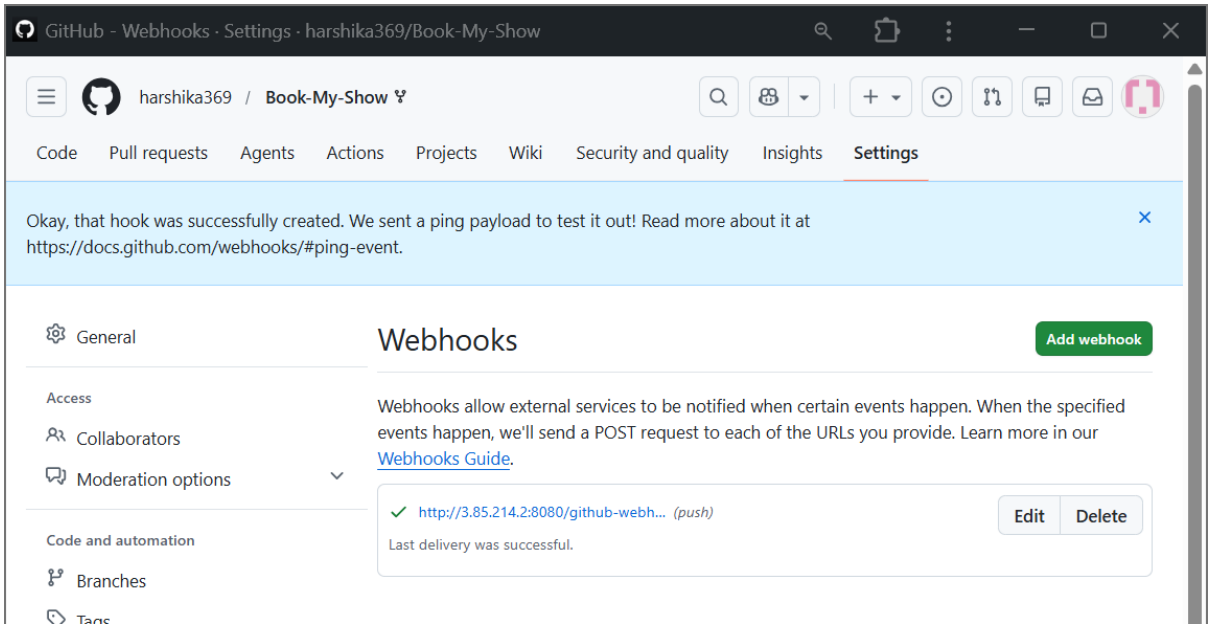
Add webhook

Check the Webhook:

Trigger push

Go to GitHub: your repo → edit any file → commit again





In Jenkins-Server:

9. Edit Dockerfile using vi

Open Dockerfile

cd Book-My-Show/bookmyshow-app
vi Dockerfile

Paste this **final working Dockerfile**:

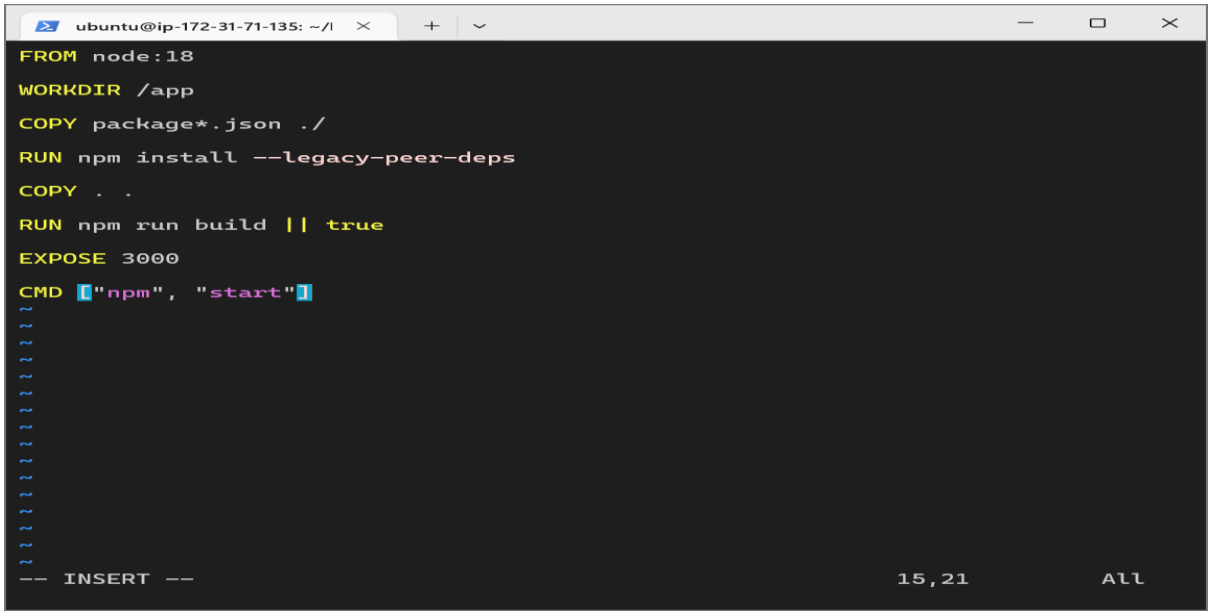
```
FROM node:18

WORKDIR /app

COPY package*.json ./

RUN npm install --legacy-peer-deps

COPY . .

RUN npm run build || true

EXPOSE 3000

CMD ["npm", "start"]
```

Save and exit

Press:Esc

Then type: :wq

and hit **Enter**



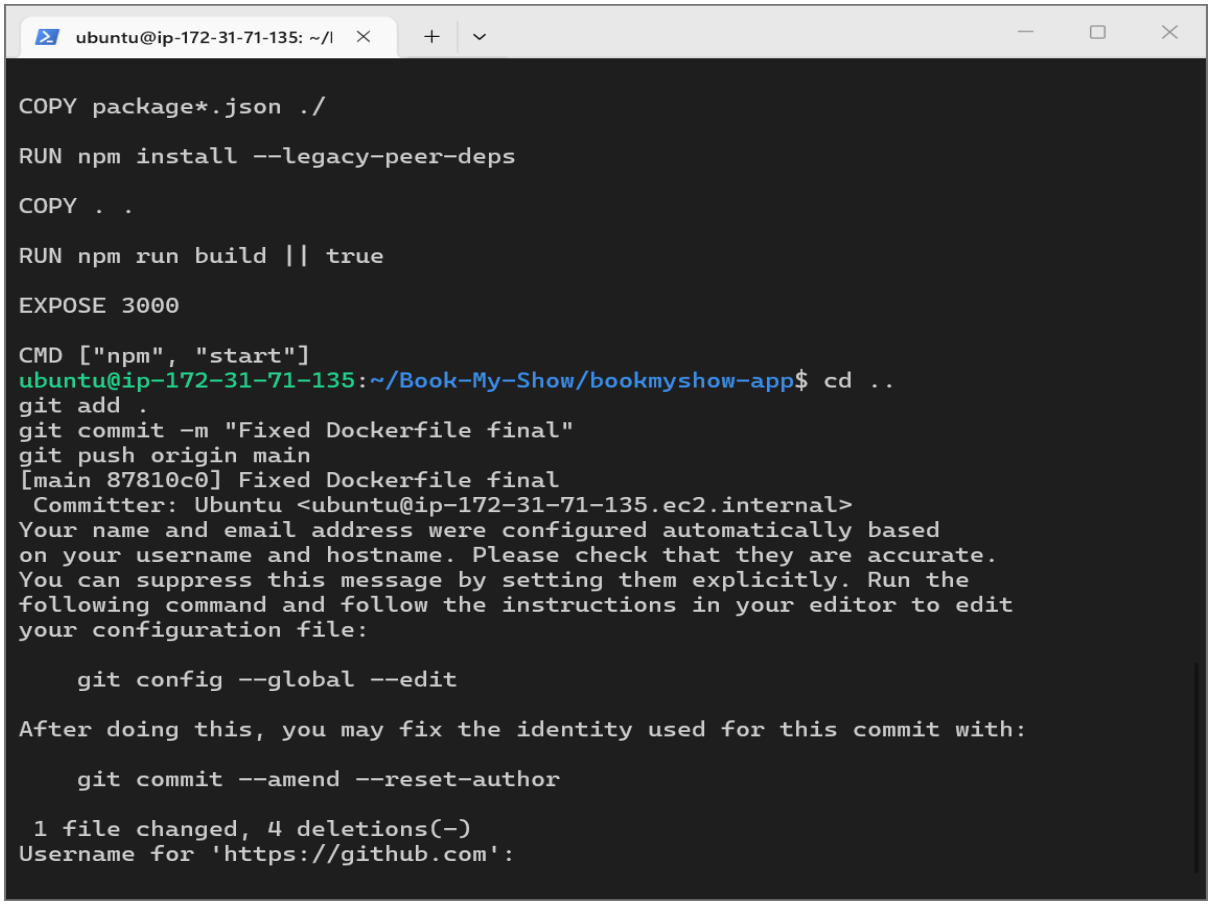
10. Push changes

cd ..

git add .

git commit -m "Fixed Dockerfile issue"

git push origin main



11. Configure Job in Jenkins

In General

Check: GitHub project

Add: https://github.com/harshika369/Book-My-Show

In Build Triggers

Check: GitHub hook trigger for GITScm polling

This is VERY important for auto trigger

In Pipeline Section

Definition: Pipeline script from SCM

SCM: Git

Repository URL: https://github.com/harshika369/Book-My-Show.git

Branch: */main

Script Path: Jenkinsfile2

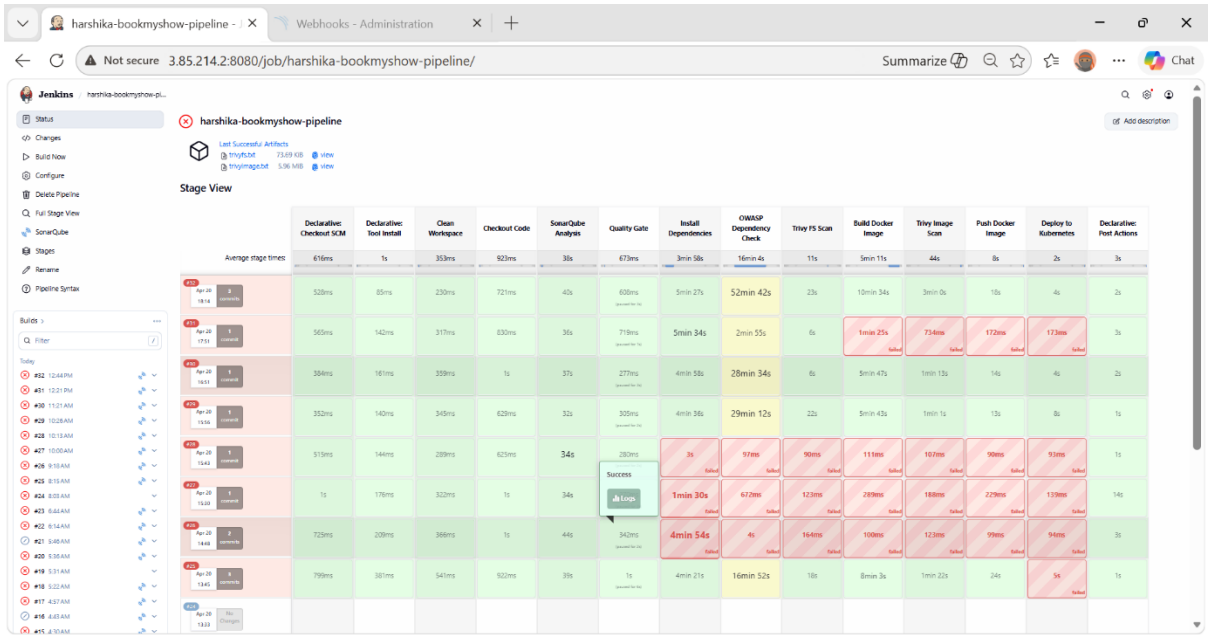
12. Replace the final script in JenkinsFile2 -without OWASP Stage

Justification to Disable OWASP

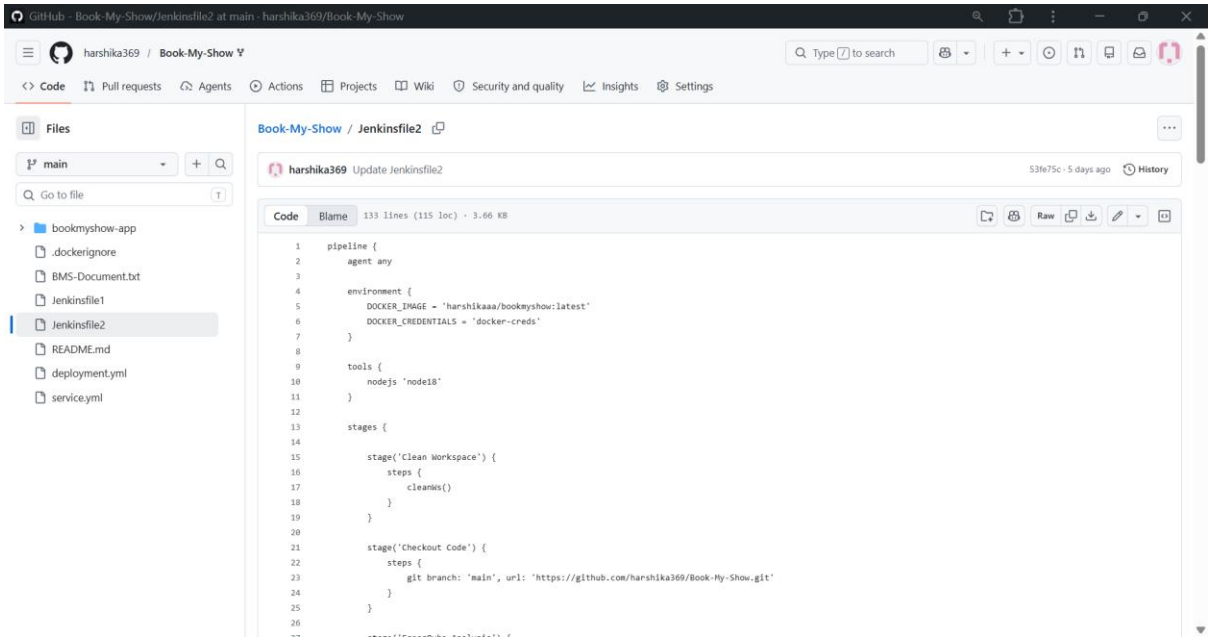
"OWASP Dependency Check was integrated into the Jenkins pipeline for security scanning. However, during execution, it significantly increased build time (30–45 minutes) due to large NVD database downloads (~300k+ CVEs) and network retries.

Since the primary objective of this project is to demonstrate a complete CI/CD pipeline (build, scan, Docker, deploy to Kubernetes), OWASP was configured as a non-blocking stage / optionally skipped, to ensure pipeline reliability and timely execution.

Lightweight security scanning using Trivy is still included, which validates container and filesystem vulnerabilities efficiently."



File- JenkinsFile2




```
pipeline {
    agent any

    environment {
        DOCKER_IMAGE = 'harshikaaa/bookmyshow:latest'
```

```
    DOCKER_CREDENTIALS = 'docker-creds'
  }

  tools {
    nodejs 'node18'
  }

  stages {

    stage('Clean Workspace') {
      steps {
        cleanWs()
      }
    }

    stage('Checkout Code') {
      steps {
        git branch: 'main', url: 'https://github.com/harshika369/Book-My-Show.git'
      }
    }

    stage('SonarQube Analysis') {
      steps {
        dir('bookmyshow-app') {
```

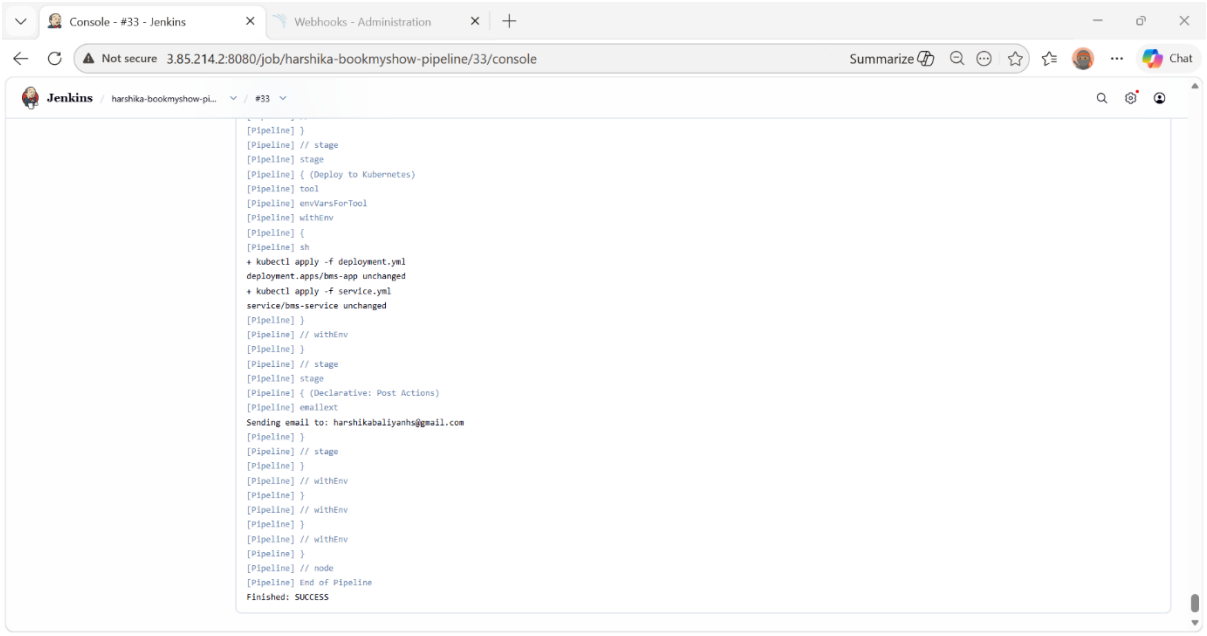
```
            script {
                def scannerHome = tool 'sonar-scanner'
                withSonarQubeEnv('sonar-server') {
                    withCredentials([string(credentialsId: 'sonar-token', variable:
'SONAR_TOKEN')]) {
                        sh """
                        ${scannerHome}/bin/sonar-scanner \
                        -Dsonar.projectName=BMS \
                        -Dsonar.projectKey=bms-app \
                        -Dsonar.sources=. \
                        -Dsonar.host.url=$SONAR_HOST_URL \
                        -Dsonar.token=$SONAR_TOKEN
                        """
                    }
                }
            }
        }
    }

    stage('Quality Gate') {
        steps {
            waitForQualityGate abortPipeline: true
        }
    }
```

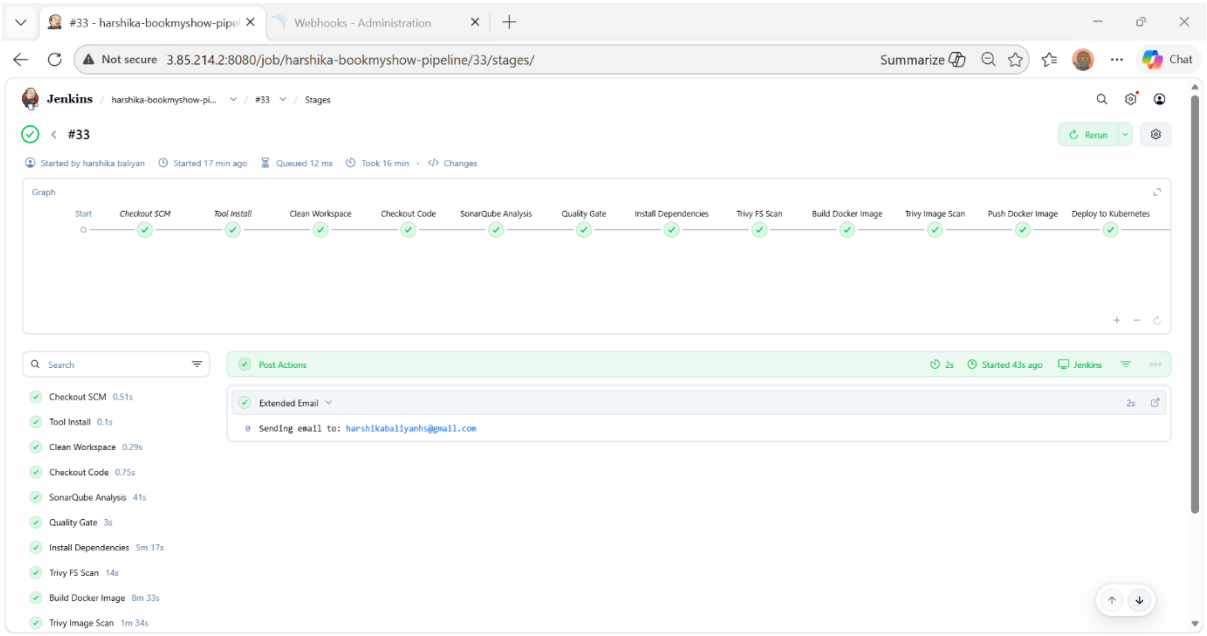
```
stage('Install Dependencies') {
    steps {
        dir('bookmyshow-app') {
            sh '''
            echo "Node version:"
            node -v
            npm -v

            npm config set registry https://registry.npmjs.org/
            npm config set fetch-retries 5
            npm config set fetch-retry-mintimeout 20000
            npm config set fetch-retry-maxtimeout 120000
            npm config set fetch-timeout 600000

            rm -rf node_modules package-lock.json
            npm cache clean --force

            npm install --legacy-peer-deps || npm install --legacy-peer-deps
            '''
        }
    }
}

stage('Trivy FS Scan') {
```

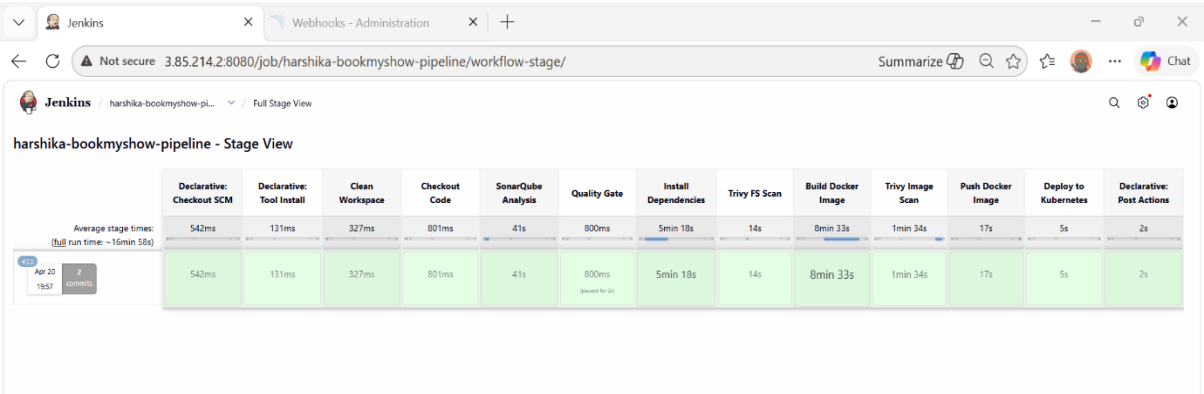
```
        steps {
            sh 'trivy fs bookmyshow-app || true'
        }
    }

    stage('Build Docker Image') {
        steps {
            sh 'docker build -t $DOCKER_IMAGE -f bookmyshow-app/Dockerfile bookmyshow-app'
        }
    }

    stage('Trivy Image Scan') {
        steps {
            sh 'trivy image $DOCKER_IMAGE || true'
        }
    }

    stage('Push Docker Image') {
        steps {
            script {
                docker.withRegistry('', DOCKER_CREDENTIALS) {
                    sh 'docker push $DOCKER_IMAGE'
                }
            }
```

```
        }
    }

    stage('Deploy to Kubernetes') {
        steps {
            sh '''
            kubectl apply -f deployment.yml
            kubectl apply -f service.yml
            '''
        }
    }
}

post {
    success {
        emailext (
            subject: "SUCCESS: Build Passed",
            body: "Pipeline executed successfully 🚀 ",
            to: "harshi[redacted]@gmail.com"
        )
    }

    failure {
        emailext (
            subject: "FAILURE: Build Failed",
```

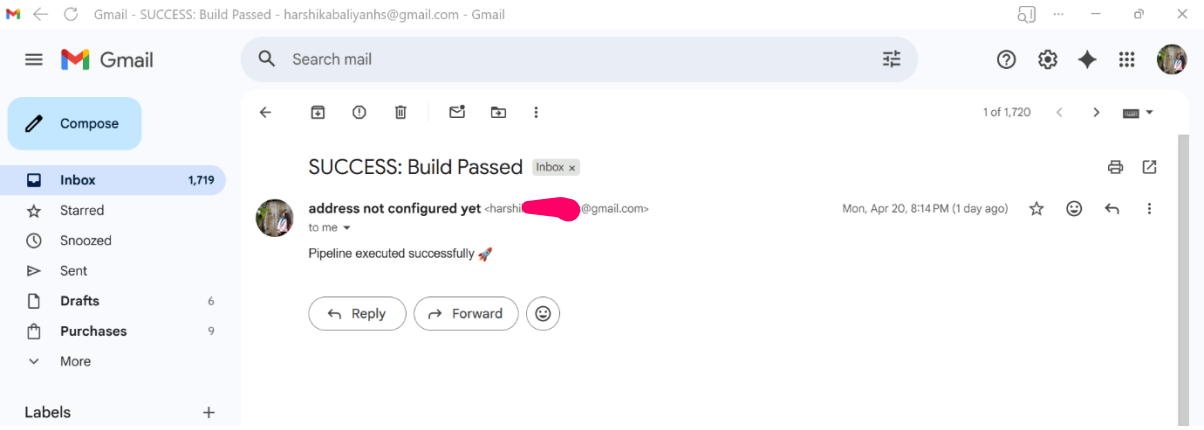
```
            body: "Pipeline failed ❌. Please check Jenkins logs.",
            to: "harshik[redacted]s@gmail.com"
        )
    }
  }
}
```







Notification of Successful Build:



Now

kubectl get nodes

kubectl get pods

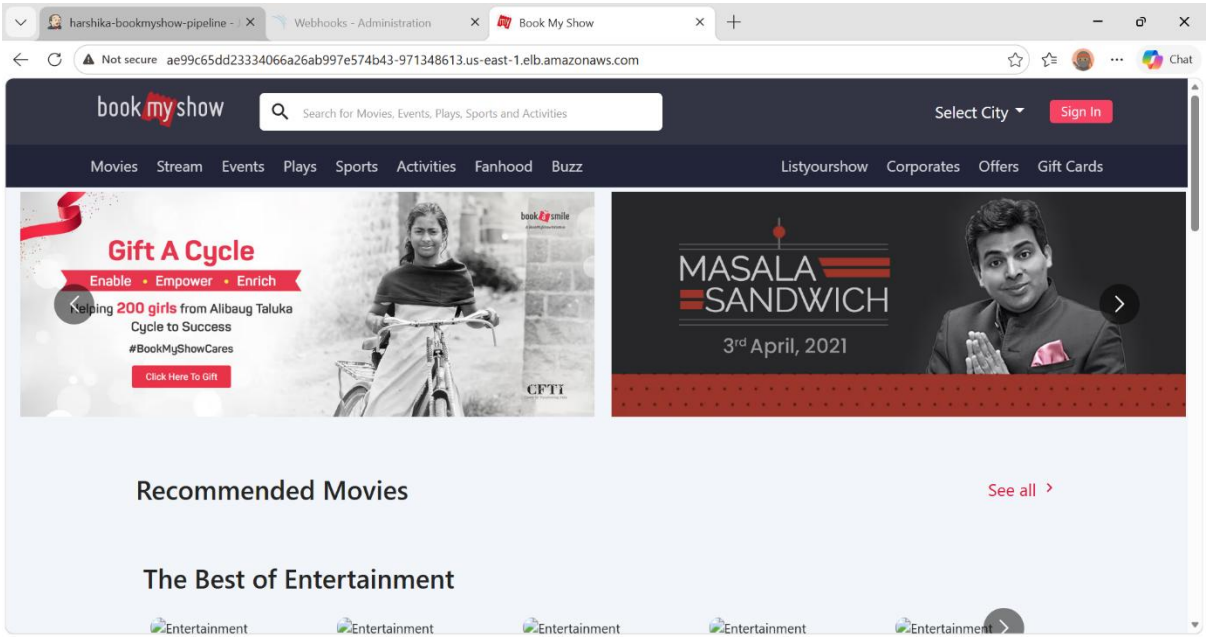
```
ubuntu@ip-172-31-71-135:~$ kubectl get pods
NAME                      READY   STATUS              RESTARTS   AG
E
bms-app-658cc79876-n6mhj  1/1     Running             0          14
s
bms-app-c7756889d-f2h8z   1/1     Running             0          14
s
bms-app-df6d7784d-tvvrx   0/1     ContainerCreating   0          13
s
ubuntu@ip-172-31-71-135:~$
```

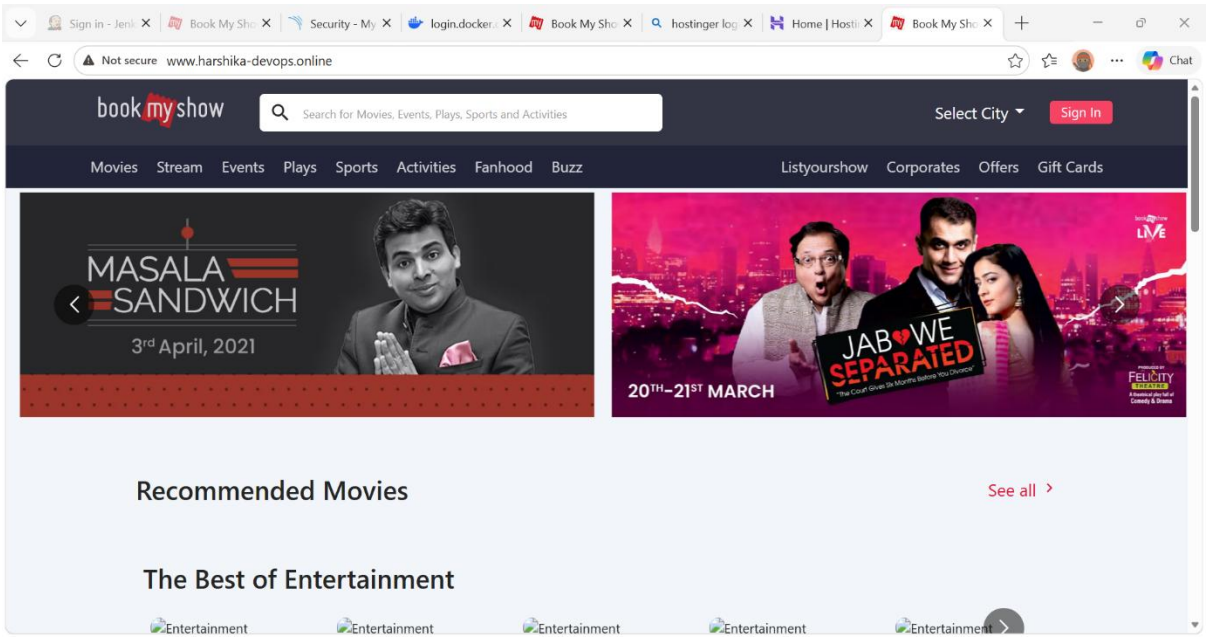
Your App URL

bms-service LoadBalancer
EXTERNAL-IP: ae99c65dd23334066a26ab997e574b43-971348613.us-east-1.elb.amazonaws.com
PORT: 80

Open browser

http://ae99c65dd23334066a26ab997e574b43-971348613.us-east-1.elb.amazonaws.co





Phase 4: Monitoring Setup using Prometheus & Grafana on Kubernetes

Step 0: Prerequisites

Make sure:

kubectl get nodes

```
ubuntu@ip-172-31-71-135:~$ kubectl get nodes
NAME                              STATUS   ROLES    AGE     VERSION
ip-192-168-35-177.ec2.internal   Ready    <none>   6m26s   v1.34.6-eks-bbe087e
ubuntu@ip-172-31-71-135:~$ kubectl get pods -n kube-system
NAME                              READY   STATUS    RESTARTS   AGE
aws-node-smdzq                    2/2     Running   0          6m36s
coredns-7d58d485c9-2wrl6          1/1     Running   0          3h35m
coredns-7d58d485c9-r9m95          1/1     Running   0          3h19m
kube-proxy-pb7gn                  1/1     Running   0          6m36s
metrics-server-6f49c4bc6c-5jw5z   1/1     Running   0          20h
metrics-server-6f49c4bc6c-9j8mh   1/1     Running   0          20h
ubuntu@ip-172-31-71-135:~$
```

Step 1: Create Namespace

kubectl create namespace monitoring

```
ubuntu@ip-172-31-71-135:~$ kubectl create namespace monitoring
namespace/monitoring created
```

Step 2: Create Prometheus Config

cat <<EOF > prometheus-config.yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: prometheus-config
  namespace: monitoring
data:
  prometheus.yml: |
    global:
      scrape_interval: 15s

    scrape_configs:
      - job_name: 'prometheus'

```
    static_configs:
      - targets: ['localhost:9090']

  - job_name: 'node-exporter'
    static_configs:
      - targets: ['node-exporter.monitoring.svc.cluster.local:9100']
EOF
```

Apply:

kubectl apply -f prometheus-config.yaml

```
ubuntu@ip-172-31-71-135:~$ cat <<EOF > prometheus.yml
global:
  scrape_interval: 30s

scrape_configs:
  - job_name: 'prometheus'
    static_configs:
      - targets: ['localhost:9090']
EOF
ubuntu@ip-172-31-71-135:~$ kubectl create configmap prometheus-config \
  --from-file=prometheus.yml \
  -n monitoring
configmap/prometheus-config created
ubuntu@ip-172-31-71-135:~$ cat <<EOF > prometheus-deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: prometheus
  namespace: monitoring
spec:
```

Step 3: Deploy Prometheus

```
cat <<EOF > prometheus-deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: prometheus
  namespace: monitoring
spec:
  replicas: 1
  selector:
    matchLabels:
      app: prometheus
  template:
```

```
  metadata:
    labels:
      app: prometheus
  spec:
    containers:
    - name: prometheus
      image: prom/prometheus
      args:
        - "--config.file=/etc/prometheus/prometheus.yml"
      ports:
        - containerPort: 9090
      resources:
        requests:
          memory: "100Mi"
          cpu: "100m"
        limits:
          memory: "200Mi"
          cpu: "200m"
      volumeMounts:
        - name: config
          mountPath: /etc/prometheus
    volumes:
      - name: config
        configMap:
          name: prometheus-config
EOF
```

Apply:

kubectl apply -f prometheus-deployment.yaml

```
ubuntu@ip-172-31-71-135:~$ cat <<EOF > prometheus-deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: prometheus
  namespace: monitoring
spec:
```

Step 4: Expose Prometheus

```
kubectl expose deployment prometheus \
  --type=LoadBalancer \
  --port=9090 \
  -n monitoring
```

```
ubuntu@ip-172-31-71-135:~$ kubectl expose deployment prometheus \
  --type=LoadBalancer \
  --port=9090 \
  --name=prometheus-lb \
  -n monitoring
service/prometheus-lb exposed
ubuntu@ip-172-31-71-135:~$ kubectl get svc -n monitoring
NAME            TYPE           CLUSTER-IP       EXTERNAL-IP
                                                PORT(S)          AGE
prometheus      NodePort       10.100.183.191   <none>
                                                9090:30930/TCP   28m
prometheus-lb   LoadBalancer   10.100.118.179   aaf45a575143046a08bcd0f0f49802
4d-808355246.us-east-1.elb.amazonaws.com   9090:32389/TCP   13s
ubuntu@ip-172-31-71-135:~$
```

Step 5: Deploy Node Exporter

```
cat <<EOF > node-exporter.yaml
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: node-exporter
  namespace: monitoring
spec:
  selector:
    matchLabels:
      app: node-exporter
  template:
    metadata:
      labels:
        app: node-exporter
    spec:
      containers:
      - name: node-exporter
        image: prom/node-exporter:v1.8.1
```

```
        ports:
          - containerPort: 9100
        resources:
          requests:
            memory: "30Mi"
            cpu: "20m"
          limits:
            memory: "50Mi"
            cpu: "50m"
EOF
```

Apply:

```
kubectl apply -f node-exporter.yaml
```

Step 6: Node Exporter Service

```
kubectl expose daemonset node-exporter \
  --type=ClusterIP \
  --port=9100 \
  -n monitoring
```

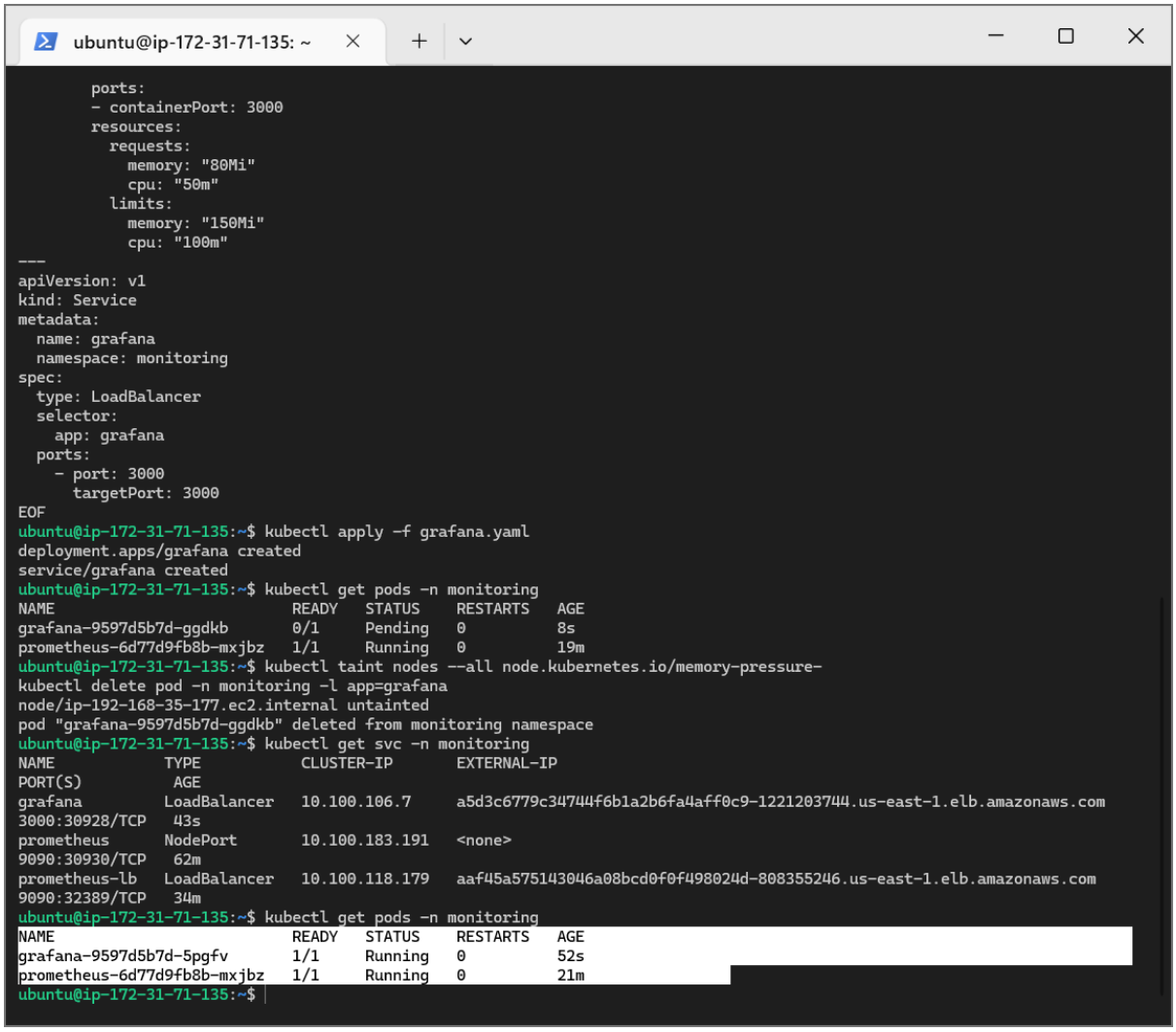
Step 7: Deploy Grafana

```
cat <<EOF > grafana.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: grafana
  namespace: monitoring
spec:
  replicas: 1
  selector:
    matchLabels:
      app: grafana
  template:
    metadata:
      labels:
```

```
      app: grafana
  spec:
    containers:
    - name: grafana
      image: grafana/grafana:10.4.2
      ports:
       - containerPort: 3000
      resources:
       requests:
         memory: "80Mi"
         cpu: "50m"
       limits:
         memory: "150Mi"
         cpu: "100m"
---
apiVersion: v1
kind: Service
metadata:
 name: grafana
 namespace: monitoring
spec:
 type: LoadBalancer
 selector:
  app: grafana
 ports:
  - port: 3000
    targetPort: 3000
EOF
```

Apply:

```
kubectl apply -f grafana.yaml
```



Step 8: Verify Pods

kubectl get pods -n monitoring

Expected:

grafana Running
prometheus Running
node-exporter Running

Step 9: Get External URLs

kubectl get svc -n monitoring

Use:

- Prometheus → http://<EXTERNAL-IP>:9090
- Grafana → http://<EXTERNAL-IP>:3000

```
ubuntu@ip-172-31-71-135:~$ kubectl get svc -n monitoring
NAME            TYPE           CLUSTER-IP       EXTERNAL-IP
PORT(S)          AGE
grafana         LoadBalancer   10.100.106.7     a5d3c6779c34744f6b1a2b6fa4aff0c9-1221203744.us-east-1.elb.amazonaws.com
3000:30928/TCP   43s
prometheus      NodePort       10.100.183.191   <none>
9090:30930/TCP   62m
prometheus-lb   LoadBalancer   10.100.118.179   aaf45a575143046a08bcd0f0f498024d-808355246.us-east-1.elb.amazonaws.com
9090:32389/TCP   34m
ubuntu@ip-172-31-71-135:~$ kubectl get pods -n monitoring
NAME                          READY   STATUS    RESTARTS   AGE
grafana-9597d5b7d-5pgfv       1/1     Running   0          52s
prometheus-6d77d9fb8b-mxjbz   1/1     Running   0          21m
ubuntu@ip-172-31-71-135:~$
```

Step 10: Verify Prometheus

Open:

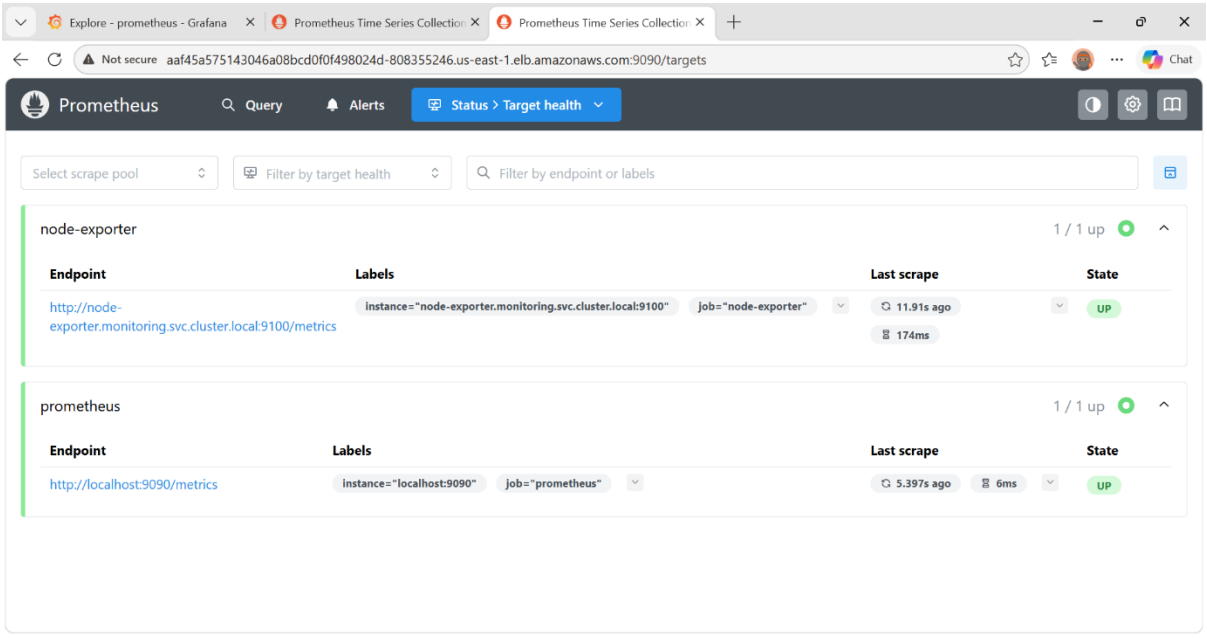
http://<PROMETHEUS-IP>:9090/targets

Check:

- prometheus → UP
- node-exporter → UP

Run query:

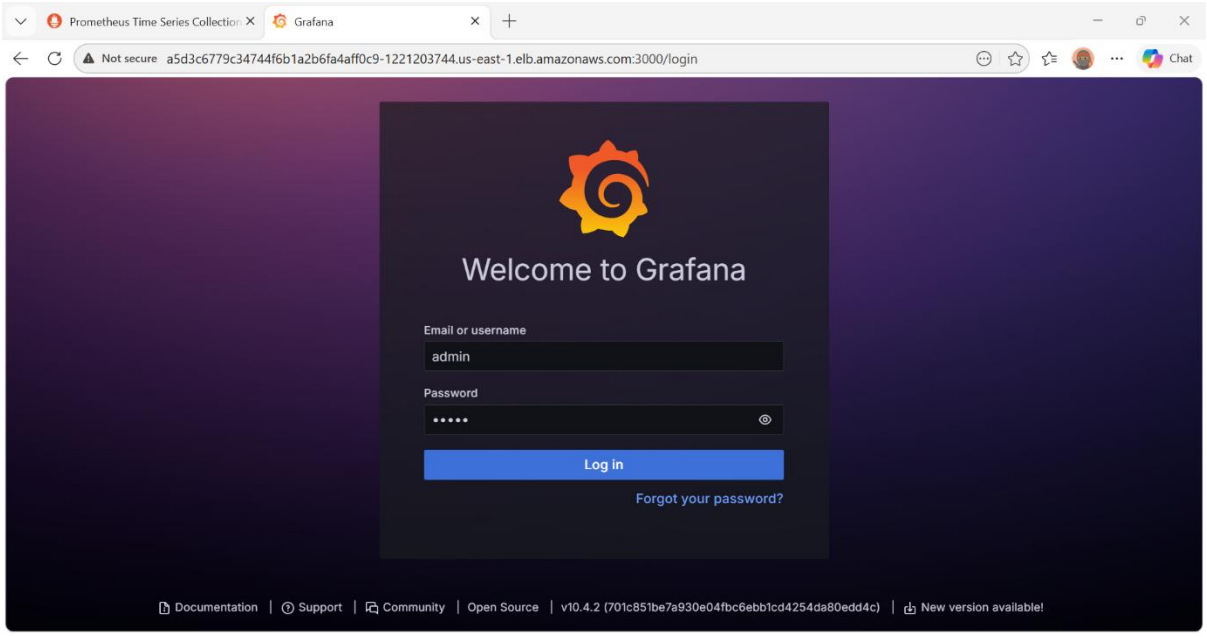
up



Step 11: Configure Grafana

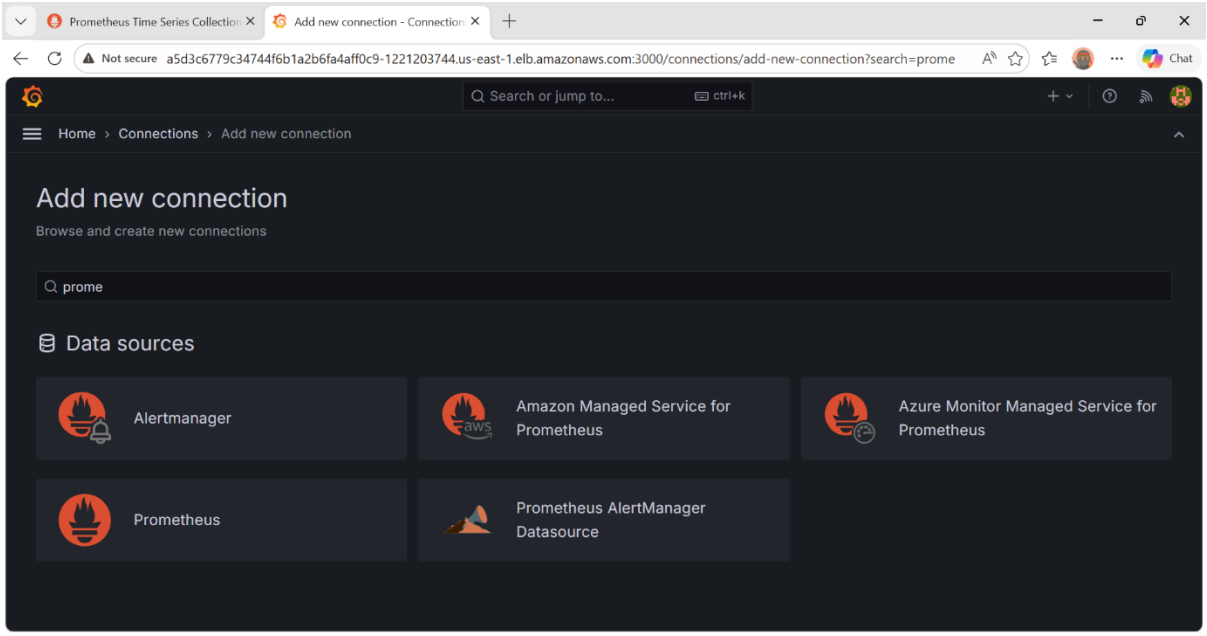
Login:

username: admin
password: admin



Add Data Source

- Type: Prometheus

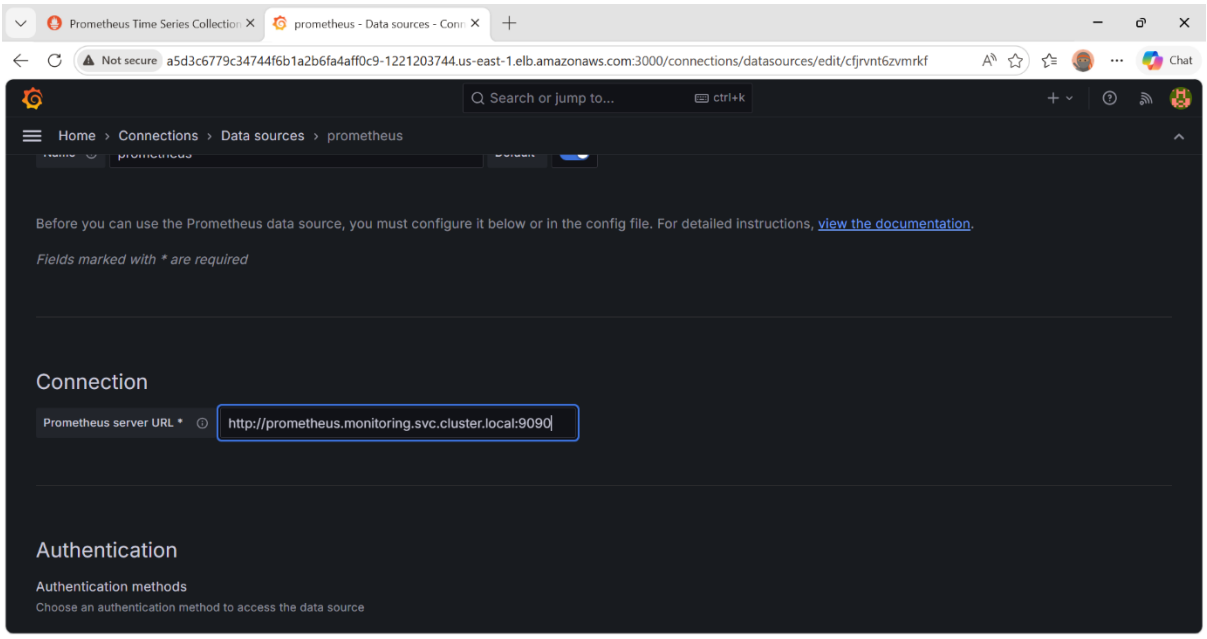


URL:

http://prometheus.monitoring.svc.cluster.local:9090

Click:

Save & Test



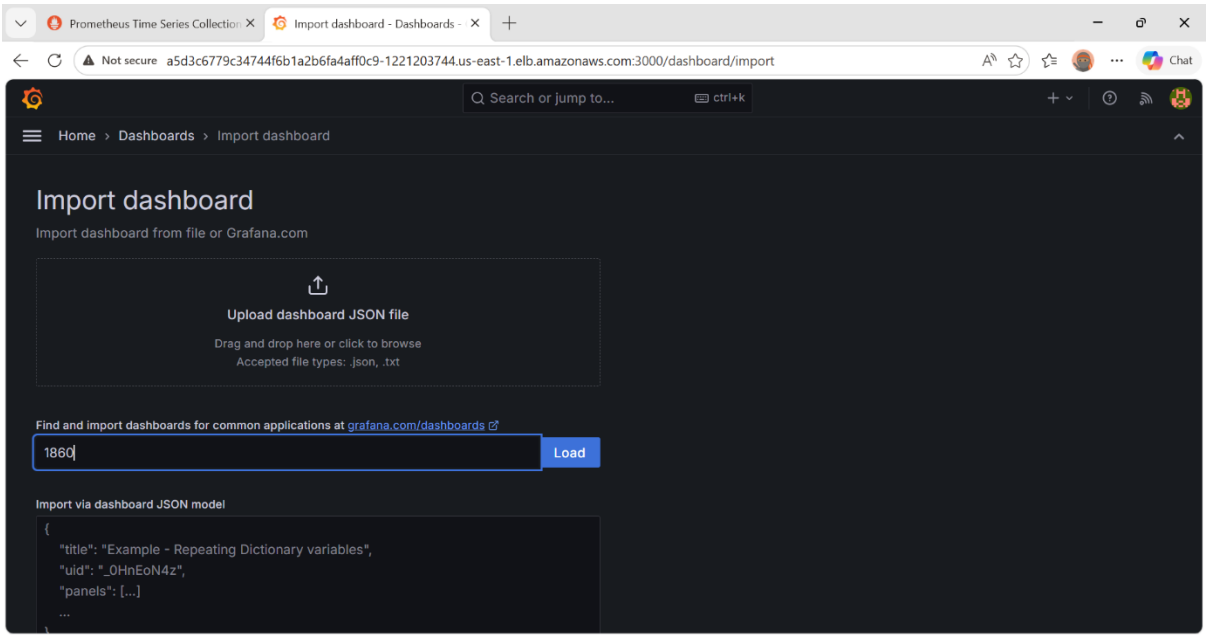
Step 12: Import Dashboard

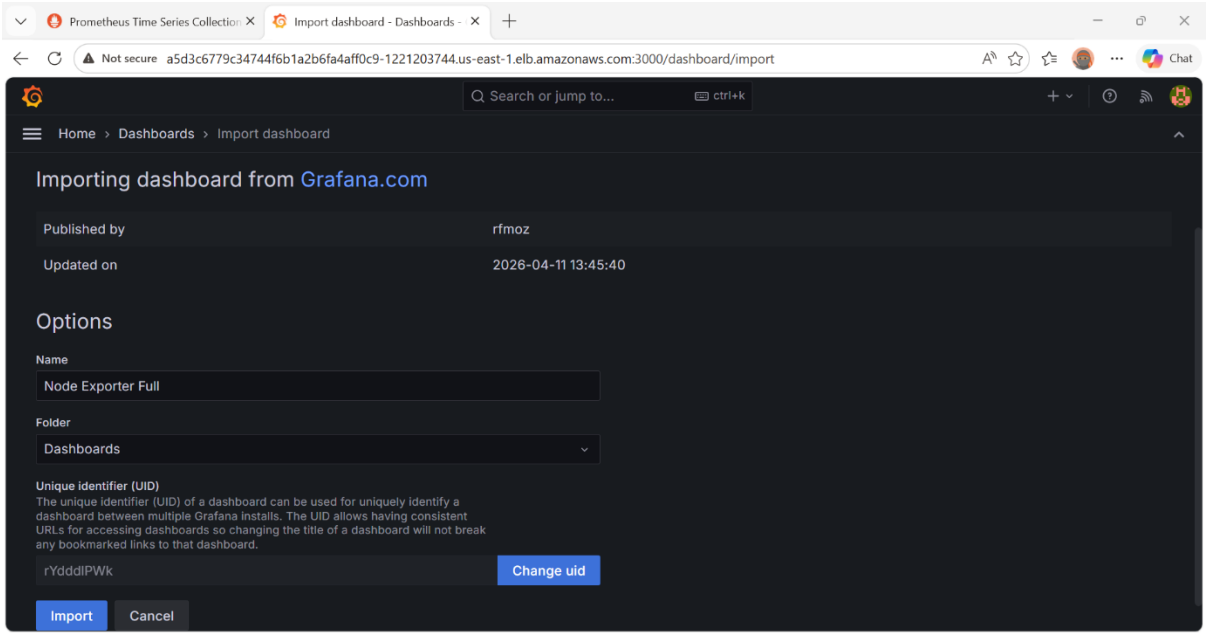
Go to:

- Dashboards → Import

Enter ID:

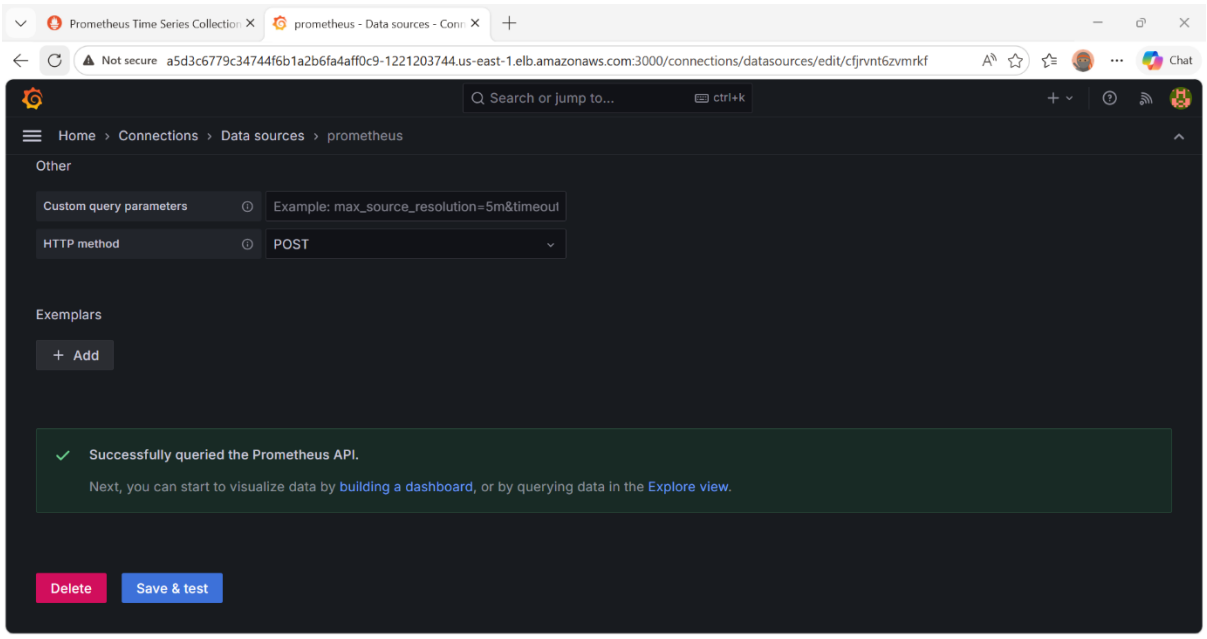
1860





Select:

- Data Source → Prometheus



Step 13: Final Output

Dashboard shows:

- CPU usage
- Memory usage
- Disk usage
- Network stats