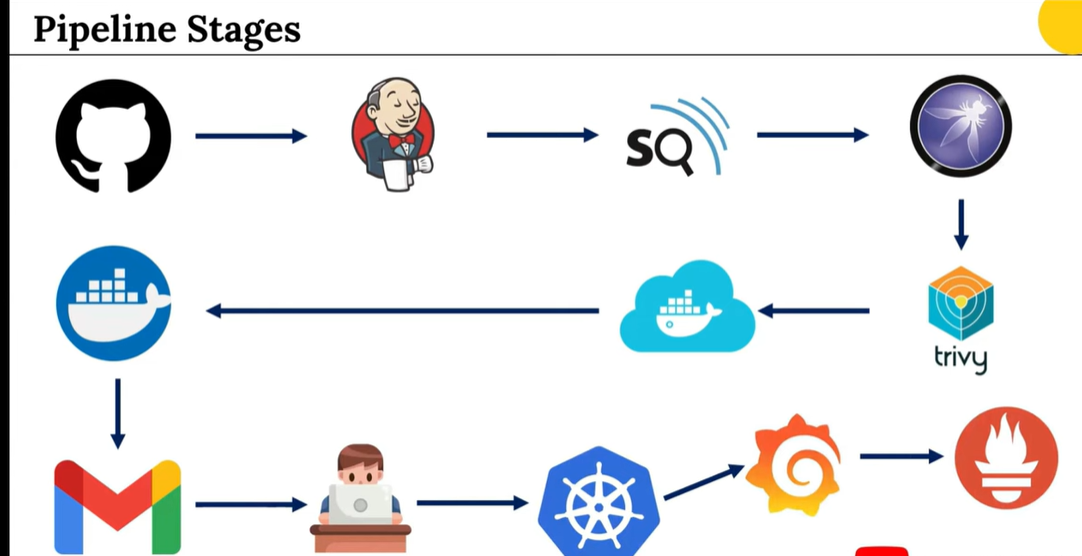
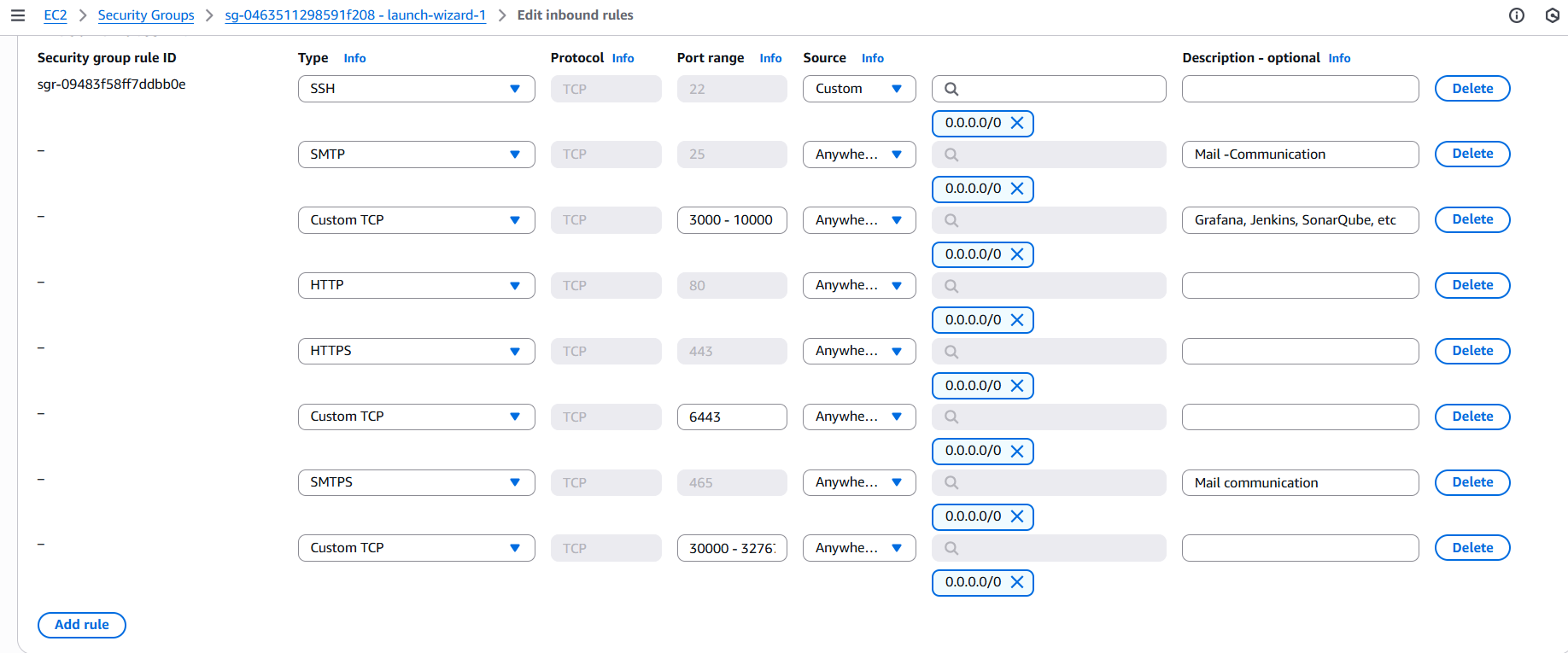
**Deployment of Book My Show App: -**

**====================================**

[**https://youtu.be/hBGVwa8MY4A?si=r7z2wOJBQxsqm6zC**](https://youtu.be/hBGVwa8MY4A?si=r7z2wOJBQxsqm6zC)

****

* **Go to EC2 -> select North Virginia region -> Launch Instance -> Name: BMS – Server -> Ubuntu image -> Instance Type: t2.large -> select key pair -> name: Juhi (.pem) -> create key pair -> storage: 28 GB -> create instance**
* **Open inbound rules in security group.**

****

* **Copy EC2 Public IP and connect to Mobaxterm or Putty.**
* **Execute below commands.**

**sudo apt update**

* **Install jenkins**

**vi jenkins.sh**

**#!/bin/bash**

**# Install OpenJDK 17 JRE Headless**

**sudo apt install openjdk-17-jre-headless -y**

**# Download Jenkins GPG key**

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

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

**# Add Jenkins repository to package manager sources**

**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 package manager repositories**

**sudo apt-get update**

**# Install Jenkins**

**sudo apt-get install jenkins -y**

**$ sudo chmod +x jenkins.sh**

**$ ls**

**$ ./jenkins.sh**

**$ java --version**

**$ systemctl status jenkins**

* **Access the jenkins server.**

**BMS-ServerPublicIP:8080**

* **Enter password and login.**

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

* **Create user id and password and login**
* **Install Docker**

**$ vi docker.sh**

**#!/bin/bash**

**# Update package manager repositories**

**sudo apt-get update**

**# Install necessary dependencies**

**sudo apt-get install -y ca-certificates curl**

**# Create directory for Docker GPG key**

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

**# Download Docker's GPG key**

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

**# Ensure proper permissions for the key**

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

**# Add Docker repository to Apt sources**

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

**$(. /etc/os-release && echo "$VERSION\_CODENAME") stable" | \**

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

**# Update package manager repositories**

**sudo apt-get update**

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

**$ sudo chmod +x docker.sh**

**$ ./docker.sh**

**$ docker –version**

**$ sudo chmod 666 /var/run/docker.sock**

* **Try to pull default image.**

**$ docker pull hello-world**

**$ docker images**

**$ docker rmi hello-world:latest**

* **Integrate docker hub account to EC2 instance.**

**$ docker login -u <username>**

* **Enter password, docker login will be succeeded.**
* **Install trivy. (trivy is a filesystem scanning tool.)**

**$ vi trivy.sh**

**#!/bin/bash**

**sudo apt-get install wget apt-transport-https gnupg**

**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 generic main" | sudo tee -a /etc/apt/sources.list.d/trivy.list**

**sudo apt-get update**

**sudo apt-get install trivy**

**$ sudo chmod +x trivy.sh**

**$ ./trivy.sh**

**$ trivy --version**

* **Sonar Qube setup**

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

**$ docker images**

**$ docker ps**

* **Access sonarqube server**

**PublicIP:9000/**

* **Login and update password.**

**Creation of EKS Cluster: -**

**=========================**

* **Go to AWS -> IAM -> create user -> name: eks-juhi-user -> Select: Provide user access to the AWS Management Console -> I want to create an IAM user -> custom password -> uncheck (user must set new password) -> Next (don’t attach any policies) -> create user**
* **View user -> permissions ->Add permissions -> Attach policies directly**
* [**AmazonEC2FullAccess**](https://us-east-1.console.aws.amazon.com/iam/home?region=us-east-1#/policies/details/arn%3Aaws%3Aiam%3A%3Aaws%3Apolicy%2FAmazonEC2FullAccess)
* [**IAMFullAccess**](https://us-east-1.console.aws.amazon.com/iam/home?region=us-east-1#/policies/details/arn%3Aaws%3Aiam%3A%3Aaws%3Apolicy%2FIAMFullAccess)
* [**AmazonEKS\_CNI\_Policy**](https://us-east-1.console.aws.amazon.com/iam/home?region=us-east-1#/policies/details/arn%3Aaws%3Aiam%3A%3Aaws%3Apolicy%2FAmazonEKS_CNI_Policy)
* [**AmazonEKSClusterPolicy**](https://us-east-1.console.aws.amazon.com/iam/home?region=us-east-1#/policies/details/arn%3Aaws%3Aiam%3A%3Aaws%3Apolicy%2FAmazonEKSClusterPolicy)
* [**AmazonEKSWorkerNodePolicy**](https://us-east-1.console.aws.amazon.com/iam/home?region=us-east-1#/policies/details/arn%3Aaws%3Aiam%3A%3Aaws%3Apolicy%2FAmazonEKSWorkerNodePolicy)
* [**AWSCloudFormationFullAccess**](https://us-east-1.console.aws.amazon.com/iam/home?region=us-east-1#/policies/details/arn%3Aaws%3Aiam%3A%3Aaws%3Apolicy%2FAWSCloudFormationFullAccess)
* **Click on next -> Add permissions.**
* **User -> Permissions -> Add permissions -> create inline policy -> Select JSON**
* **Attach the below inline policy also for the same user**

**{**

**"Version": "2012-10-17",**

**"Statement": [**

**{**

**"Sid": "VisualEditor0",**

**"Effect": "Allow",**

**"Action": "eks:\*",**

**"Resource": "\*"**

**}**

**]**

**}**

* **Click on next -> Policy name: eks-inline-policy -> create policy**
* **Create Access Keys for the user created**
* **With this we have created the IAM User with appropriate permissions to create the EKS Cluster.**
* **Install AWS CLI (to interact with AWS Account)**

**$ vi aws.sh**

**$ chmod +x aws.sh**

**$ ./aws.sh**

**curl "https://awscli.amazonaws.com/awscli-exe-linux-x86\_64.zip" -o "awscliv2.zip"**

**sudo apt install unzip**

**unzip awscliv2.zip**

**sudo ./aws/install**

* **Check the version**

**$ aws --version**

* **Configure aws by executing below command**

**$ aws configure**

* **Enter access key and secret key.**
* **Region: us-east-1**
* **Output format: json**

**# Install kubectl to interact with K8S: -**

**---------------------------------------------------**

**$ vi kubectl.sh**

**curl -o kubectl https://amazon-eks.s3.us-west-2.amazonaws.com/1.19.6/2021-01-05/bin/linux/amd64/kubectl**

**chmod +x ./kubectl**

**sudo mv ./kubectl /usr/local/bin**

**kubectl version --short --client**

* **Execute below commands.**

**$ chmod +x kubectl.sh**

**$ ./kubectl.sh**

* **Install EKS CTL (used to create EKS cluster)**

**$ vi eksctl.sh**

**curl --silent --location "https://github.com/weaveworks/eksctl/releases/latest/download/eksctl\_$(uname -s)\_amd64.tar.gz" | tar xz -C /tmp**

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

**eksctl version**

* **Give permissions.**

**$ sudo chmod +x eksctl.sh**

**$ ./eksctl.sh**

* **Create EKS cluster**

**eksctl create cluster --name=juhi-eks \**

**--region=us-east-1 \**

**--zones=us-east-1a,us-east-1b \**

**--version=1.30 \**

**--without-nodegroup**

* **Goto EKS Console and verify the cluster.**
* **Associate with IAM-OIDC provider.**

**eksctl utils associate-iam-oidc-provider \**

**--region us-east-1 \**

**--cluster juhi-eks \**

**--approve**

**-> Execute below command.**

**eksctl create nodegroup --cluster=juhi-eks \**

**--region=us-east-1 \**

**--name=node2 \**

**--node-type=t3.medium \**

**--nodes=3 \**

**--nodes-min=2 \**

**--nodes-max=4 \**

**--node-volume-size=20 \**

**--ssh-access \**

**--ssh-public-key=Juhi \**

**--managed \**

**--asg-access \**

**--external-dns-access \**

**--full-ecr-access \**

**--appmesh-access \**

**--alb-ingress-access**

* **Setup the jenkins.**
* **Install below plugins.**
* **Go to manage jenkins -> 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 Credentials, Config File Provider, Prometheus metrics**
* **Restart jenkins**

**# SonarQube Token Creation: -**

* **Go to SonarQube -> Administration -> Security -> Users -> token -> name: token -> generate**
* **Add sonarqube token to jenkins**
* **Go to manage jenkins -> credentials -> global -> add credentials -> kind: secret text -> secret (paste token) -> ID & Desc: Sonar-token -> create**

**# Add dockerhub credentials**

* **Go to manage jenkins -> credentials -> global -> add credentials -> kind: username and password -> give username and password of dockerhub -> ID: docker and desc: docker-creds -> create**

**# Create sonarqube webhooks**

* **Go to sonarqube -> administration -> configurations -> webhook 🡪 create -> name: jenkins -> URL : jenkins-url (**[**http://3.93.73.64:8080/sonarqube-webhook**](http://3.93.73.64:8080/sonarqube-webhook)**/) -> create**

**# Configure Tools setup in jenkins**

* **Go to manage jenkins -> Tools -> JDK Installations -> Add JDK -> name: jdk17 -> Install automatically -> Install from adoptium.net -> version: jdk-17.0.8.1+1**
* **SonarQube Scanner installations -> Add sonarqube scanner -> Name: sonar-scanner**
* **NodeJS Installations -> Add NodeJS -> name: node23**
* **Dependency-Check installations -> Add dependency check -> name: DP-Check -> Install automatically -> Install from github.com**
* **Docker Installations -> add docker -> name: docker -> Install automatically -> install from docker.com**
* **Click on apply & save.**

**# Integrate Sonarqube with jenkins**

* **Go to manage jenkins -> system -> SonarQube Installations -> Add SonarQube -> name: sonar-server -> Server-URL:** [**http://3.93.73.64:9000**](http://3.93.73.64:9000) **(sonar-server IP) -> select sonar authentication token -> Apply and save**

**Email Integration**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**As soon as the build happens, I need to get an email notification to do that we have to configure our email.**

* **Goto Gmail ---> Click on Icon on top right ---> Click on 'Your google account' ---> in search box at top, search for 'App Passwords' and click on it, enter password of gmail ---> Next ---> App name: jenkins ---> Create ---> You can see the password (ex: fxssvxsvfbartxnt) ---> Copy it.**
* **Make sure to remove the spaces in the password. Let’s configure this password in Jenkins.**
* **Goto the Jenkins console ---> Manage Jenkins ---> Security ---> Credentials ---> Under 'Stores scoped to Jenkins', Click on 'Global' under 'Domains' ---> Add credentials ---> Kind: Username with Password, Scope: Global, Username: <ProvideEmail ID>, Password: <PasteTheToken>, ID: email-creds, Description: email-creds ---> Create**
* **You can see the email credentials got created.**
* **Manage Jenkins ---> System ---> Scroll down to 'Extended Email Notification' ---> SMTP Server: smtp.gmail.com ---> SMTP Port: 465, Click on 'Advanced' ---> Credentials: Select 'email creds' from drop down, 'Check' Use SSL and Use OAuth 2.0, Default content type: HTML**
* **Scroll down to 'Email Notification' ---> SMTP Server: smtp.gmail.com -> Click on 'Advanced' ---> 'Check' Use SMTP Authentication, Username: <ProvideEmailID>, Password: <PasteThePasswordToken>, 'Check' Use SSL, SMTP Port: 465, Reply-to-email: <ProvideEmail>, Charset: UTF-8,, Check 'Test configuration by sending test e-mail', Test Email Recepient: <provide-email-id>, Click on 'Test Configuration' ---> You can see 'email sent' ---> Goto email and check for test email**
* **Let’s make another configuration to get an email when build fails/success ---> Goto 'Default Triggers' drop down (If you cannot find this, try searching using control + F ---> 'Check' Always, Failure-Any, Success ---> Apply ---> Save**

**# Install NPM**

**$ sudo apt install npm**

**# Create and Build jenkins job**

* **Go to jenkins dashboard -> New Item -> name: BMS-App -> Pipeline -> OK**
* **Paste the below script**

**pipeline {**

**agent any**

**tools {**

**jdk 'jdk17'**

**nodejs 'node23'**

**}**

**environment {**

**SCANNER\_HOME = tool 'sonar-scanner'**

**}**

**stages {**

**stage('Clean Workspace') {**

**steps {**

**cleanWs()**

**}**

**}**

**stage('Checkout from Git') {**

**steps {**

**git branch: 'main', url: '** **https://github.com/juhisinha422/Book-My-Show.git'**

**sh 'ls -la' // Verify files after checkout**

**}**

**}**

**stage('SonarQube Analysis') {**

**steps {**

**withSonarQubeEnv('sonar-server') {**

**sh '''**

**$SCANNER\_HOME/bin/sonar-scanner -Dsonar.projectName=BMS \**

**-Dsonar.projectKey=BMS**

**'''**

**}**

**}**

**}**

**stage('Quality Gate') {**

**steps {**

**script {**

**waitForQualityGate abortPipeline: false, credentialsId: 'Sonar-token'**

**}**

**}**

**}**

**stage('Install Dependencies') {**

**steps {**

**sh '''**

**cd bookmyshow-app**

**ls -la # Verify package.json exists**

**if [ -f package.json ]; then**

**rm -rf node\_modules package-lock.json # Remove old dependencies**

**npm install # Install fresh dependencies**

**else**

**echo "Error: package.json not found in bookmyshow-app!"**

**exit 1**

**fi**

**'''**

**}**

**}**

**stage('OWASP FS Scan') {**

**steps {**

**dependencyCheck additionalArguments: '--scan ./ --disableYarnAudit --disableNodeAudit', odcInstallation: 'DP-Check'**

**dependencyCheckPublisher pattern: '\*\*/dependency-check-report.xml'**

**}**

**}**

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

**steps {**

**sh 'trivy fs . > trivyfs.txt'**

**}**

**}**

**stage('Docker Build & Push') {**

**steps {**

**script {**

**withDockerRegistry(credentialsId: 'docker', toolName: 'docker') {**

**sh '''**

**echo "Building Docker image..."**

**docker build --no-cache -t juhisinha/bms:latest -f bookmyshow-app/Dockerfile bookmyshow-app**

**echo "Pushing Docker image to registry..."**

**docker push juhisinha/bms:latest**

**'''**

**}**

**}**

**}**

**}**

**stage('Deploy to Container') {**

**steps {**

**sh '''**

**echo "Stopping and removing old container..."**

**docker stop bms || true**

**docker rm bms || true**

**echo "Running new container on port 3000..."**

**docker run -d --restart=always --name bms -p 3000:3000 juhisinha/bms:latest**

**echo "Checking running containers..."**

**docker ps -a**

**echo "Fetching logs..."**

**sleep 5 # Give time for the app to start**

**docker logs bms**

**'''**

**}**

**}**

**}**

**post {**

**always {**

**emailext attachLog: true,**

**subject: "'${currentBuild.result}'",**

**body: "Project: ${env.JOB\_NAME}<br/>" +**

**"Build Number: ${env.BUILD\_NUMBER}<br/>" +**

**"URL: ${env.BUILD\_URL}<br/>",**

**to: 'jsinha1199@gmail.com',**

**attachmentsPattern: 'trivyfs.txt,trivyimage.txt'**

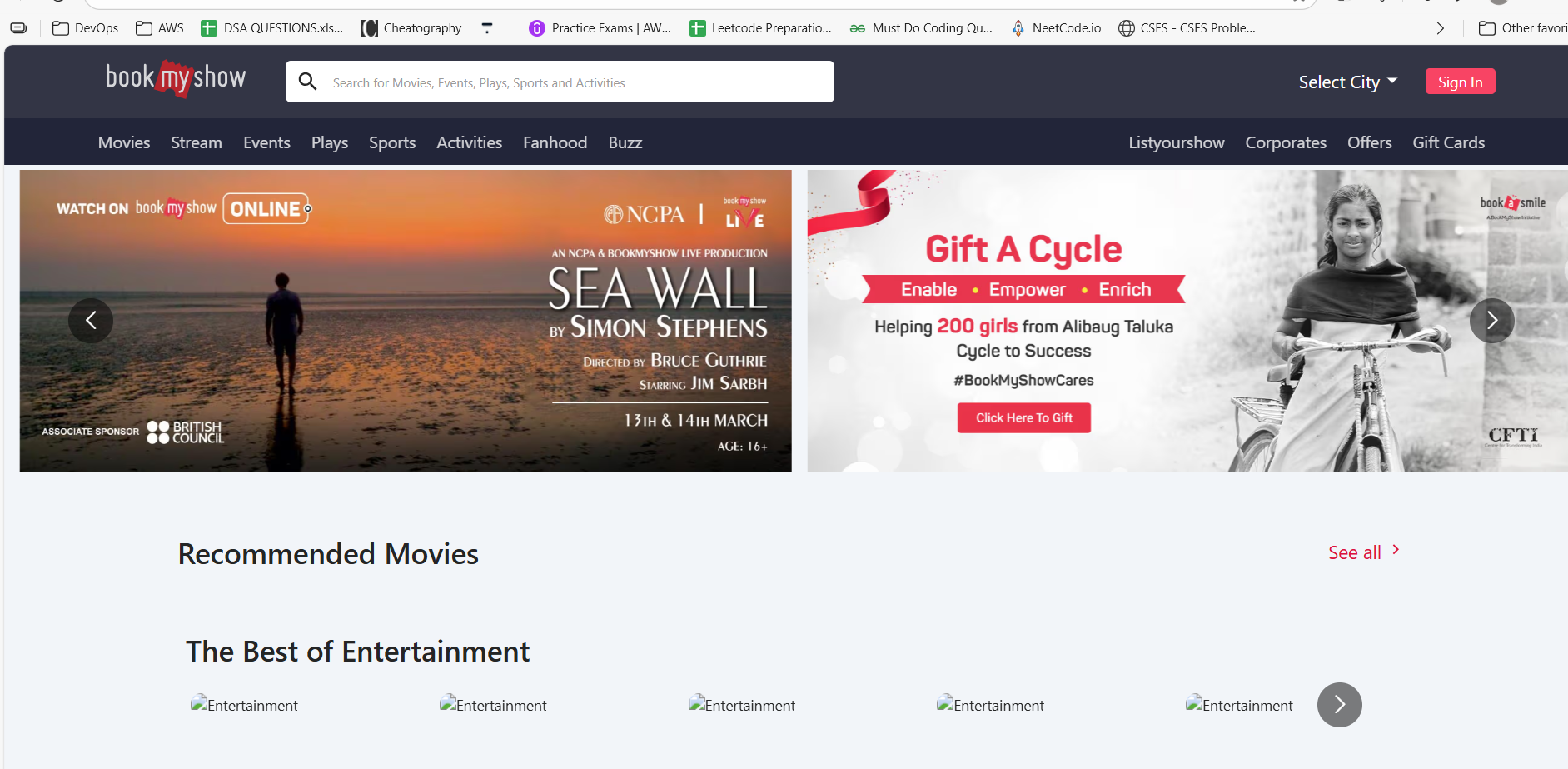
**}**

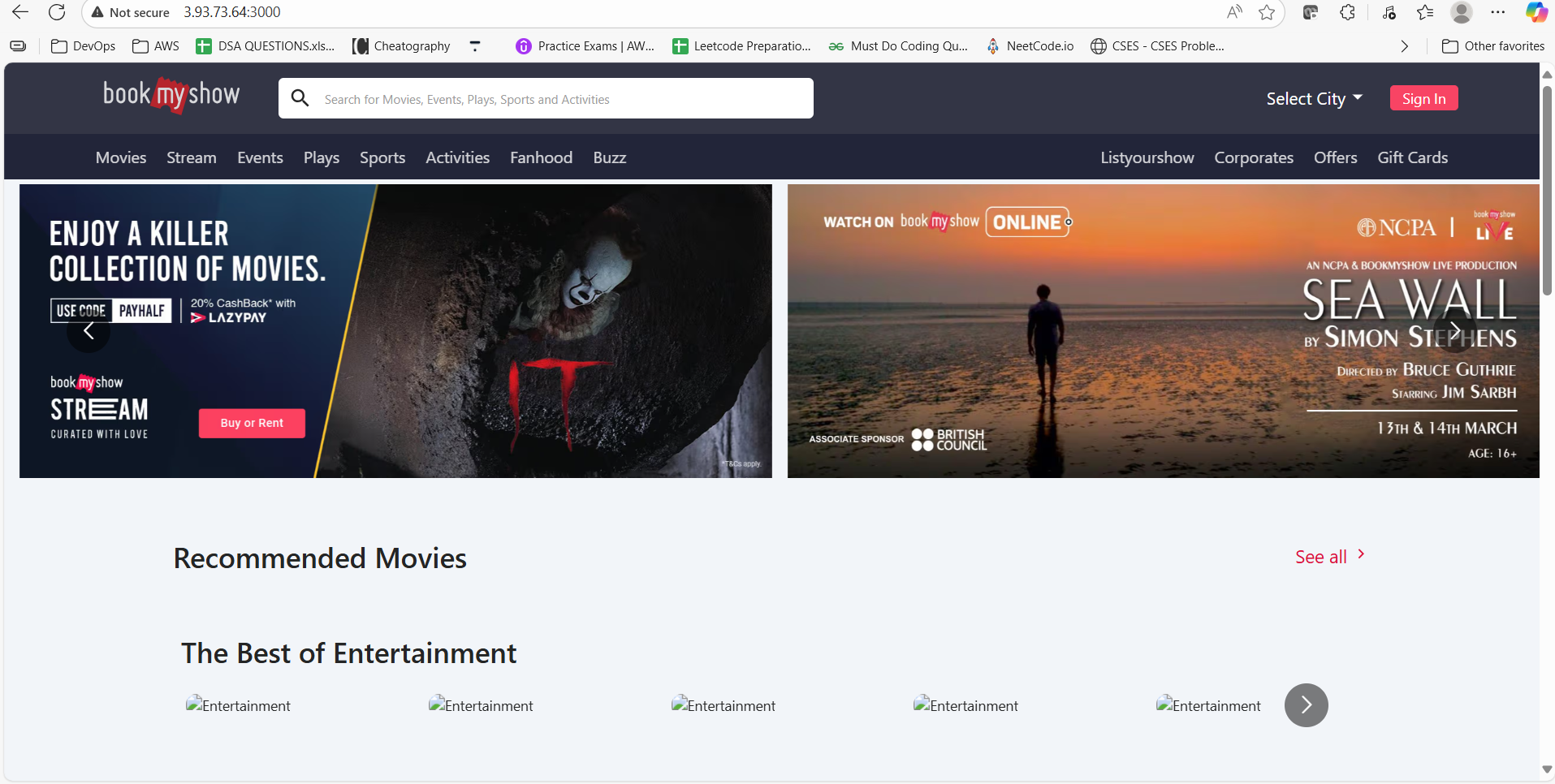
**}**

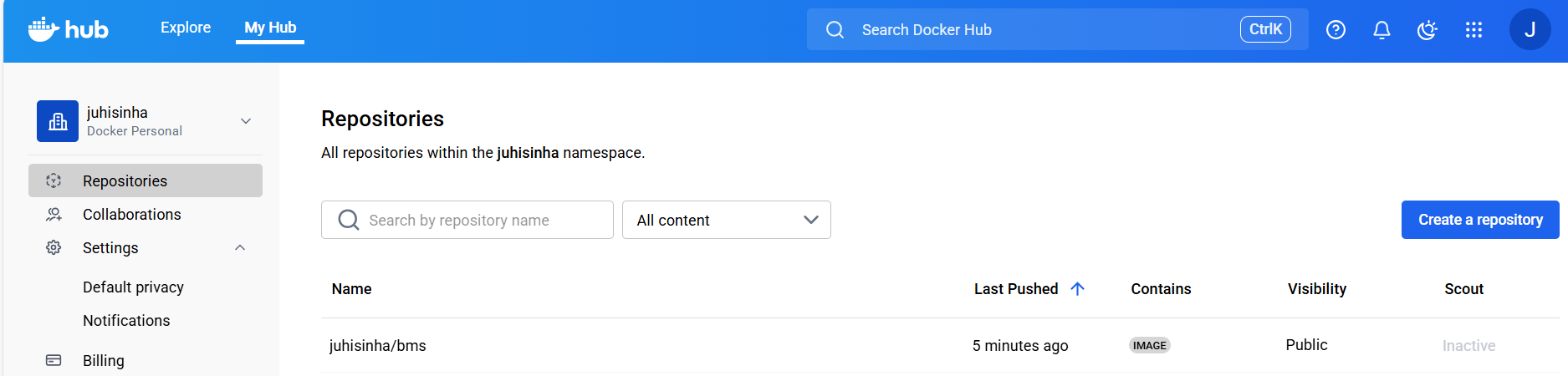
**}**

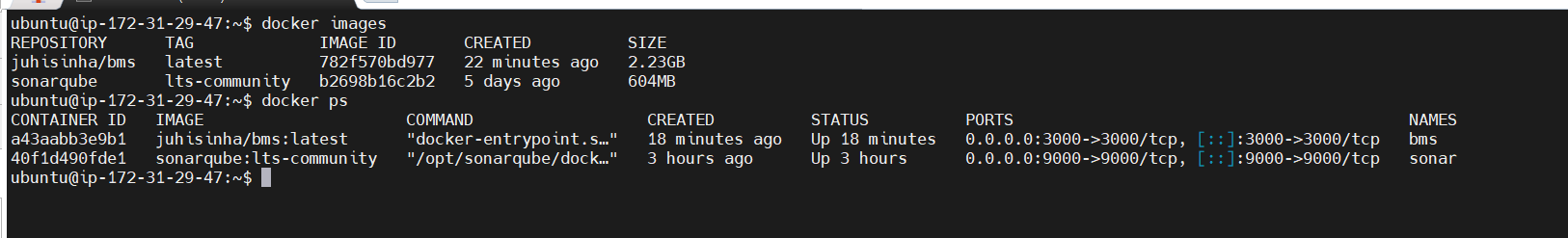
* **Access the application.**

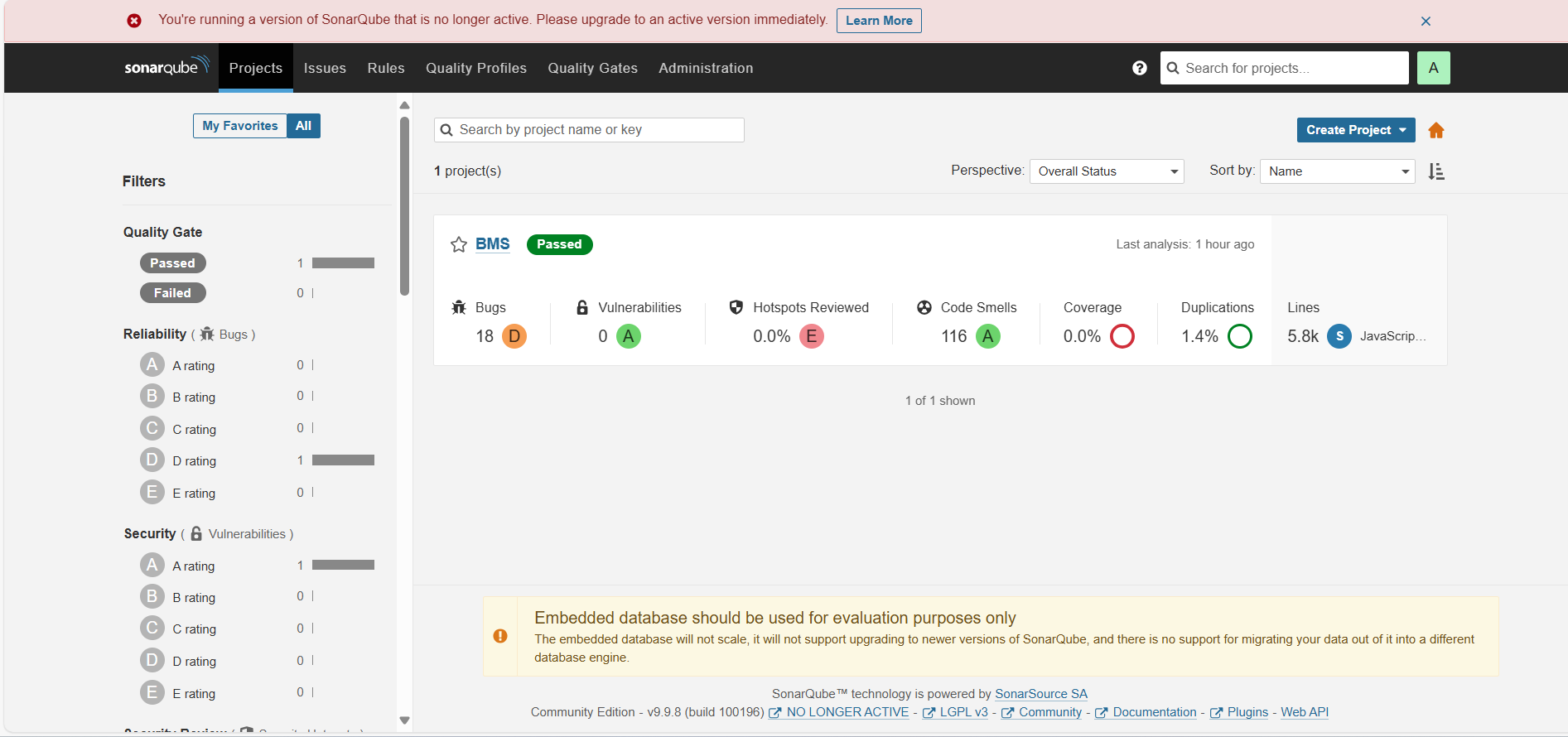
**$ PUBLIC\_IP:3000/**

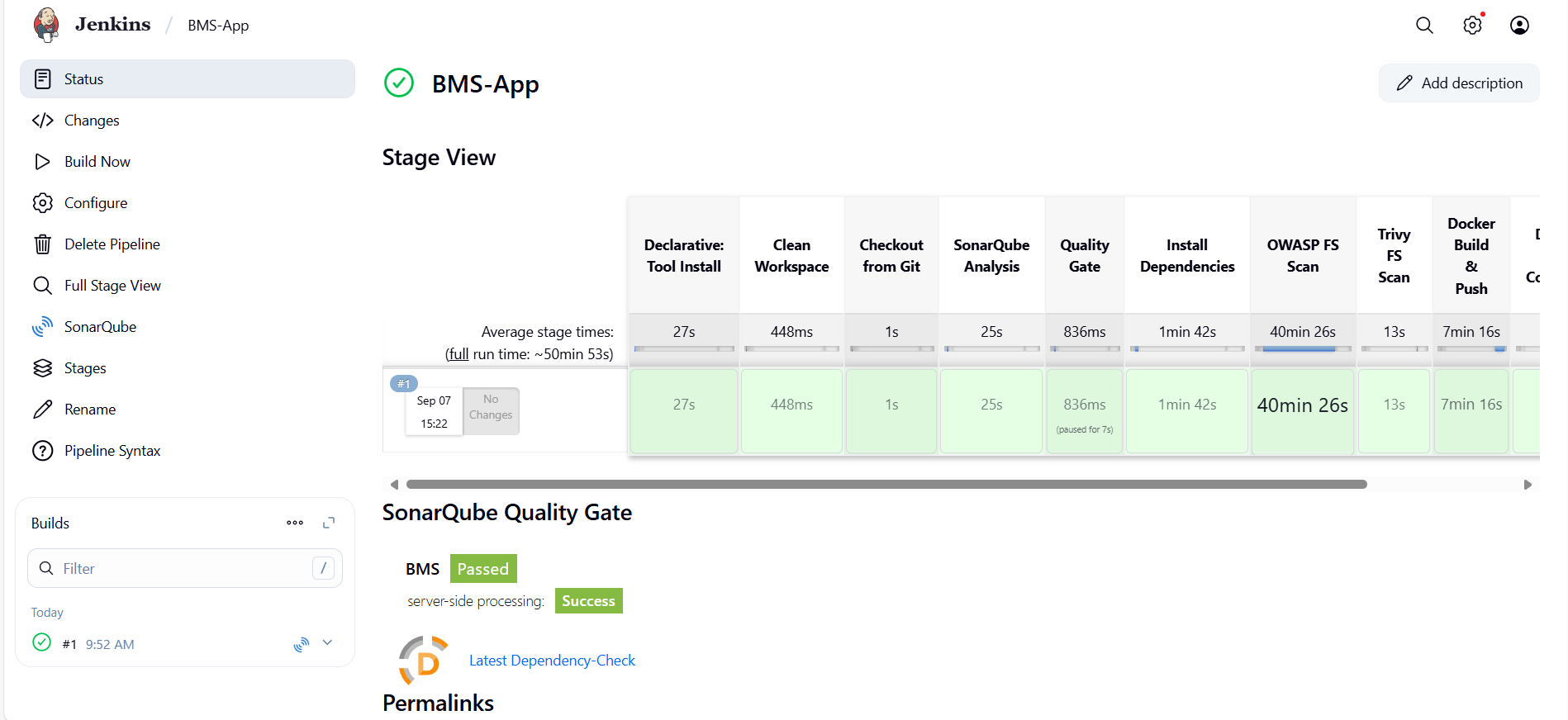
****

****

****

****

****

****

**# Deployment with K8S Stage:**

* **To know jenkins is running on which user.**

**$ ps aux | grep jenkins**

* **Switch to the jenkins user**

**sudo -su jenkins**

**pwd ---- /home/ubuntu**

**whoami ---- jenkins**

* **Configure AWS credentials:**

**Regin: us-east-1**

**aws configure ---> Configure with access and secret access keys**

**This will create the AWS credentials file at "/var/lib/jenkins/.aws/credentials"**

* **Verify the credentials (sts= simple token service)**

**aws sts get-caller-identity**

**If the credentials are valid, you should see output like this:**

**{**

**"UserId": "EXAMPLEUSERID",**

**"Account": "123456789012",**

**"Arn": "arn:aws:iam::123456789012:user/example-user"**

**}**

* **Comeout of the Jenkins user to Restart Jenkins**

**exit**

**sudo systemctl restart jenkins**

* **Switch to Jenkins user**

**sudo -su jenkins**

**aws eks update-kubeconfig --name juhi-eks --region us-east-1**

* **Go to jenkins dashboard -> BMS-App job and configure the script.**

**#(with K8S Stage)**

**pipeline {**

**agent any**

**tools {**

**jdk 'jdk17'**

**nodejs 'node23'**

**}**

**environment {**

**SCANNER\_HOME = tool 'sonar-scanner'**

**DOCKER\_IMAGE = 'juhisinha/bms:latest'**

**EKS\_CLUSTER\_NAME = 'juhi-eks'**

**AWS\_REGION = 'us-east-1'**

**}**

**stages {**

**stage('Clean Workspace') {**

**steps {**

**cleanWs()**

**}**

**}**

**stage('Checkout from Git') {**

**steps {**

**git branch: 'main', url: 'https://github.com/juhisinha422/Book-My-Show.git'**

**sh 'ls -la' // Verify files after checkout**

**}**

**}**

**stage('SonarQube Analysis') {**

**steps {**

**withSonarQubeEnv('sonar-server') {**

**sh '''**

**$SCANNER\_HOME/bin/sonar-scanner \**

**-Dsonar.projectName=BMS \**

**-Dsonar.projectKey=BMS**

**'''**

**}**

**}**

**}**

**stage('Quality Gate') {**

**steps {**

**script {**

**waitForQualityGate abortPipeline: false, credentialsId: 'Sonar-token'**

**}**

**}**

**}**

**stage('Install Dependencies') {**

**steps {**

**sh '''**

**cd bookmyshow-app**

**ls -la # Verify package.json exists**

**if [ -f package.json ]; then**

**rm -rf node\_modules package-lock.json # Remove old dependencies**

**npm install # Install fresh dependencies**

**else**

**echo "Error: package.json not found in bookmyshow-app!"**

**exit 1**

**fi**

**'''**

**}**

**}**

**stage('OWASP FS Scan') {**

**steps {**

**dependencyCheck additionalArguments: '--scan ./ --disableYarnAudit --disableNodeAudit', odcInstallation: 'DP-Check'**

**dependencyCheckPublisher pattern: '\*\*/dependency-check-report.xml'**

**}**

**}**

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

**steps {**

**sh 'trivy fs . > trivyfs.txt'**

**}**

**}**

**stage('Docker Build & Push') {**

**steps {**

**script {**

**withDockerRegistry(credentialsId: 'docker', toolName: 'docker') {**

**sh '''**

**echo "Building Docker image..."**

**docker build --no-cache -t $DOCKER\_IMAGE -f bookmyshow-app/Dockerfile bookmyshow-app**

**echo "Pushing Docker image to Docker Hub..."**

**docker push $DOCKER\_IMAGE**

**'''**

**}**

**}**

**}**

**}**

**stage('Deploy to EKS Cluster') {**

**steps {**

**script {**

**sh '''**

**echo "Verifying AWS credentials..."**

**aws sts get-caller-identity**

**echo "Configuring kubectl for EKS cluster..."**

**aws eks update-kubeconfig --name $EKS\_CLUSTER\_NAME --region $AWS\_REGION**

**echo "Verifying kubeconfig..."**

**kubectl config view**

**echo "Deploying application to EKS..."**

**kubectl apply -f deployment.yml**

**kubectl apply -f service.yml**

**echo "Verifying deployment..."**

**kubectl get pods**

**kubectl get svc**

**'''**

**}**

**}**

**}**

**}**

**post {**

**always {**

**emailext attachLog: true,**

**subject: "'${currentBuild.result}'",**

**body: "Project: ${env.JOB\_NAME}<br/>" +**

**"Build Number: ${env.BUILD\_NUMBER}<br/>" +**

**"URL: ${env.BUILD\_URL}<br/>",**

**to: 'jsinha1199@gmail.com',**

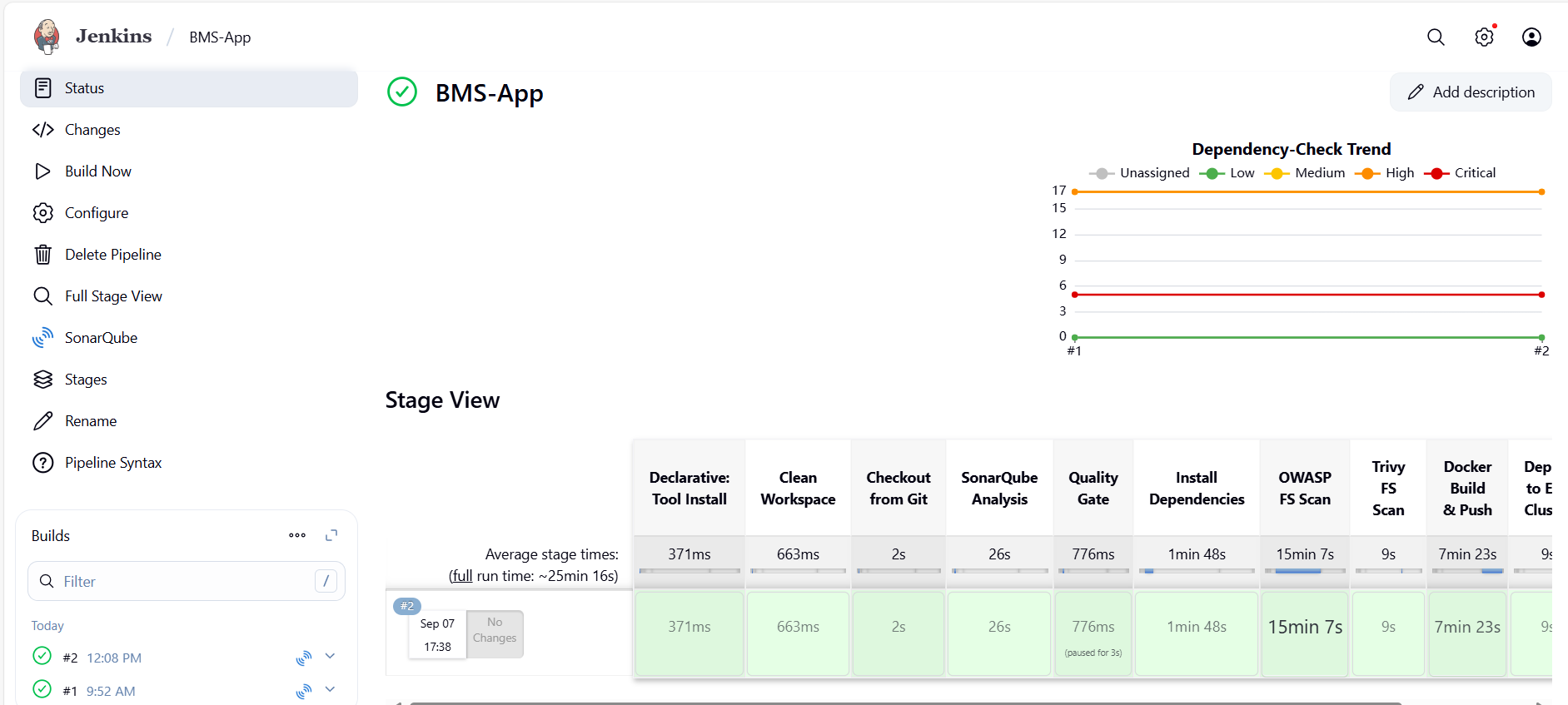
**attachmentsPattern: 'trivyfs.txt'**

**}**

**}**

**}**

* **Build the job.**

****

**# Monitoring the application**

* **Go to AWS -> EC2 -> Launch Instance -> name: Monitoring Server -> ubuntu and t2.medium -> Select 22.04 LTS -> Select existing keypair -> select create new security group -> storage: 28 GB -> Launch Instance.**
* **Connect to the monitoring server VM.**
* **Execute below commands.**

**sudo apt update**

* **Create a dedicated Linux user sometimes called a 'system' account for Prometheus**

**sudo useradd \**

**--system \**

**--no-create-home \**

**--shell /bin/false prometheus**

* **Download the Prometheus.**

**sudo wget https://github.com/prometheus/prometheus/releases/download/v2.47.1/prometheus-2.47.1.linux-amd64.tar.gz**

**tar -xvf prometheus-2.47.1.linux-amd64.tar.gz**

**sudo mkdir -p /data /etc/prometheus**

**cd prometheus-2.47.1.linux-amd64/**

* **Move the Prometheus binary and a promtool to the /usr/local/bin/. promtool is used to check configuration files and Prometheus rules.**

**sudo mv prometheus promtool /usr/local/bin/**

* **Move console libraries to the Prometheus configuration directory**

**sudo mv consoles/ console\_libraries/ /etc/prometheus/**

* **Move the example of the main Prometheus configuration file**

**sudo mv prometheus.yml /etc/prometheus/prometheus.yml**

* **Set the correct ownership for the /etc/prometheus/ and data directory**

**sudo chown -R prometheus:prometheus /etc/prometheus/ /data/**

* **Delete the archive and a Prometheus tar.gz file**

**cd**

**You are in ~ path**

**rm -rf prometheus-2.47.1.linux-amd64.tar.gz**

**prometheus --version**

**You will see as "version 2.47.1"**

**prometheus --help**

* **We’re going to use Systemd, which is a system and service manager for Linux operating systems. For that, we need to create a Systemd unit configuration file.**

**$ sudo vi /etc/systemd/system/prometheus.service**

* **Paste the below content.**

**[Unit]**

**Description=Prometheus**

**Wants=network-online.target**

**After=network-online.target**

**StartLimitIntervalSec=500**

**StartLimitBurst=5**

**[Service]**

**User=prometheus**

**Group=prometheus**

**Type=simple**

**Restart=on-failure**

**RestartSec=5s**

**ExecStart=/usr/local/bin/prometheus \**

**--config.file=/etc/prometheus/prometheus.yml \**

**--storage.tsdb.path=/data \**

**--web.console.templates=/etc/prometheus/consoles \**

**--web.console.libraries=/etc/prometheus/console\_libraries \**

**--web.listen-address=0.0.0.0:9090 \**

**--web.enable-lifecycle**

**[Install]**

**WantedBy=multi-user.target**

* **To automatically start the Prometheus after reboot run the below command.**

**sudo systemctl enable prometheus**

* **Start the Prometheus**

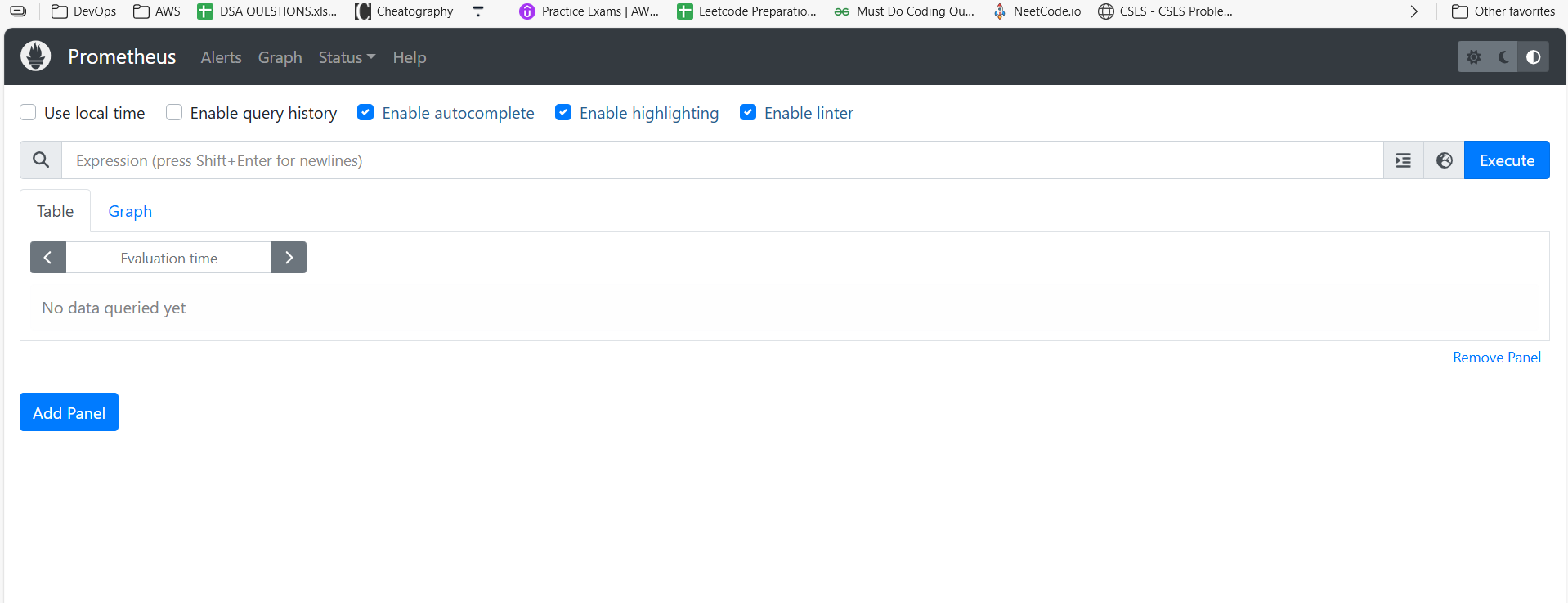
**sudo systemctl start prometheus**

* **Check the status of Prometheus**

**sudo systemctl status prometheus**

* **Open Port No. 9090 for Monitoring Server VM and Access Prometheus**

**<public-ip:9090>**

****

* **Install Node Exporter (Execute in Monitoring Server VM)**

**You are in ~ path now**

* **Create a system user for Node Exporter and download Node Exporter:**

**sudo useradd --system --no-create-home --shell /bin/false node\_exporter**

**wget** [**https://github.com/prometheus/node\_exporter/releases/download/v1.6.1/node\_exporter-1.6.1.linux-amd64.tar.gz**](https://github.com/prometheus/node_exporter/releases/download/v1.6.1/node_exporter-1.6.1.linux-amd64.tar.gz)

* **Extract Node Exporter files, move the binary, and clean up:**

**tar -xvf node\_exporter-1.6.1.linux-amd64.tar.gz**

**sudo mv node\_exporter-1.6.1.linux-amd64/node\_exporter /usr/local/bin/**

**rm -rf node\_exporter\***

**node\_exporter --version**

* **Create a systemd unit configuration file for Node Exporter:**

**sudo vi /etc/systemd/system/node\_exporter.service**

* **Add the following content to the node\_exporter.service file:**

**[Unit]**

**Description=Node Exporter**

**Wants=network-online.target**

**After=network-online.target**

**StartLimitIntervalSec=500**

**StartLimitBurst=5**

**[Service]**

**User=node\_exporter**

**Group=node\_exporter**

**Type=simple**

**Restart=on-failure**

**RestartSec=5s**

**ExecStart=/usr/local/bin/node\_exporter --collector.logind**

**[Install]**

**WantedBy=multi-user.target**

**Note: Replace --collector.logind with any additional flags as needed.**

* **Enable and start Node Exporter:**

**sudo systemctl enable node\_exporter**

**sudo systemctl start node\_exporter**

* **Verify the Node Exporter's status:**

**sudo systemctl status node\_exporter**

* **You can see "active (running)" in green colour**

**Press control+c to come out of the file**

**# Configure Prometheus Plugin Integration**

* **As of now we created Prometheus service, but we need to add a job in order to fetch the details by node exporter. So for that we need to create 2 jobs, one with 'node exporter' and the other with 'jenkins' as shown below;**
* **Integrate Jenkins with Prometheus to monitor the CI/CD pipeline.**

**# Prometheus Configuration:**

* **To configure Prometheus to scrape metrics from Node Exporter and Jenkins, you need to modify the prometheus.yml file.**
* **The path of prometheus.yml is;**

**cd /etc/prometheus/ ----> ls -l ----> You can see the "prometheus.yml"**

* **file ----> sudo vi prometheus.yml ----> You will see the content and also there is a default job called "Prometheus" Paste the below content at the end of the file;**

**- job\_name: 'node\_exporter'**

**static\_configs:**

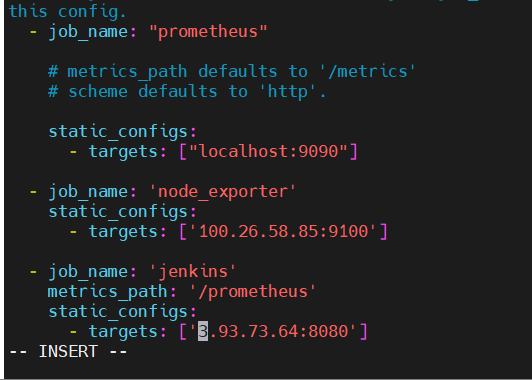
**- targets: ['<MonitoringVMip>:9100']**

**- job\_name: 'jenkins'**

**metrics\_path: '/prometheus'**

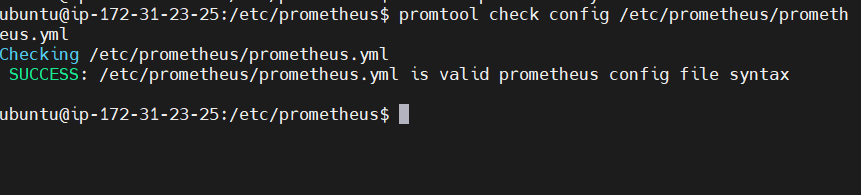
**static\_configs:**

**- targets: ['<your-jenkins-ip>:<your-jenkins-port>']**

****

* **In the above, replace <your-jenkins-ip> and <your-jenkins-port> with the appropriate IPs ----> esc ----> :wq**
* **Also replace the public ip of monitorting VM. Dont change 9100. Even though the Monitoring server is running on 9090, dont change 9100 in the above script**
* **Check the validity of the configuration file:**

**promtool check config /etc/prometheus/prometheus.yml**

****

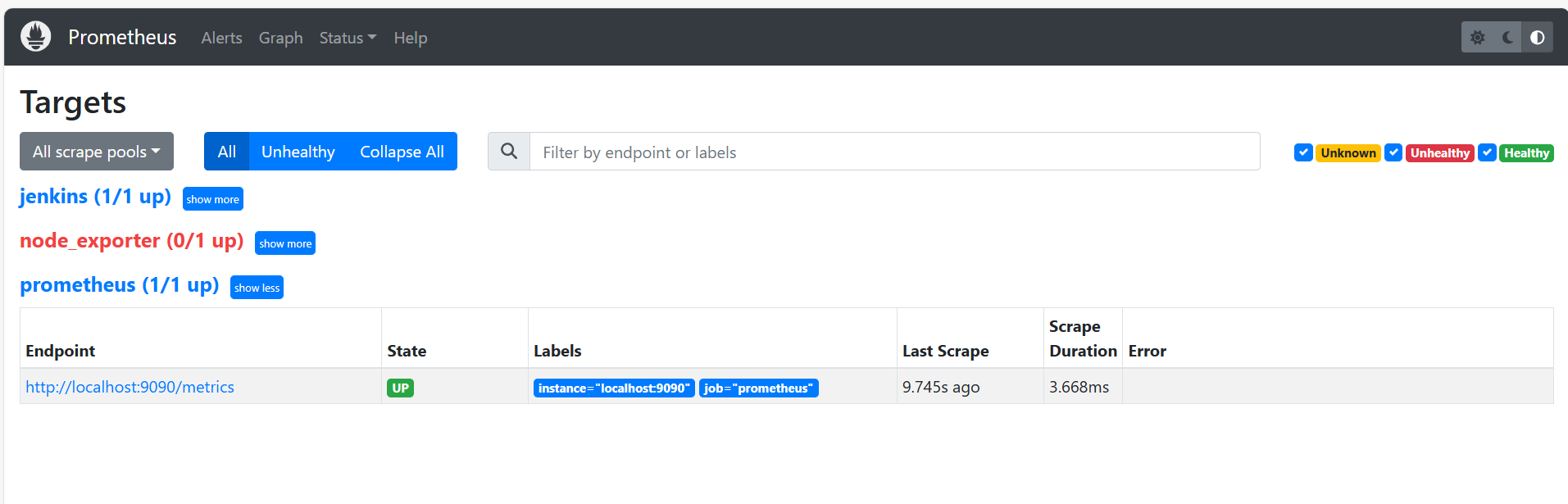
* **You should see "SUCCESS" when you run the above command, it means every configuration made so far is good.**
* **Reload the Prometheus configuration without restarting:**

**curl -X POST http://localhost:9090/-/reload**

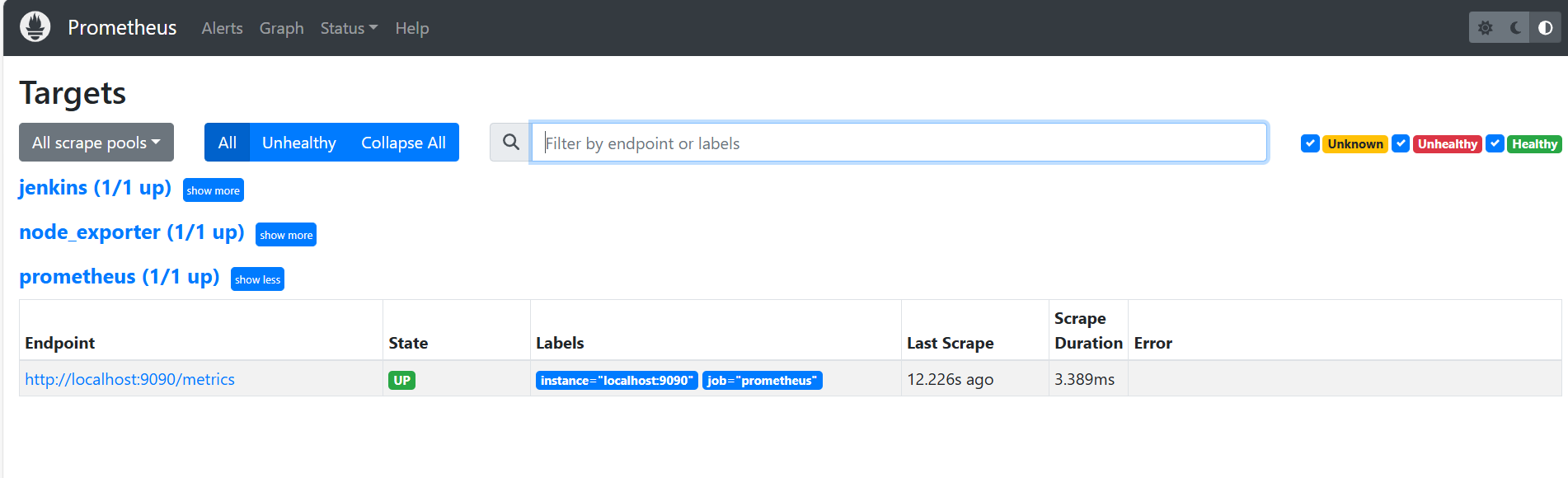
* **Access Prometheus in browser (if already opened, just reload the page):**

**http://<your-prometheus-ip>:9090/targets**

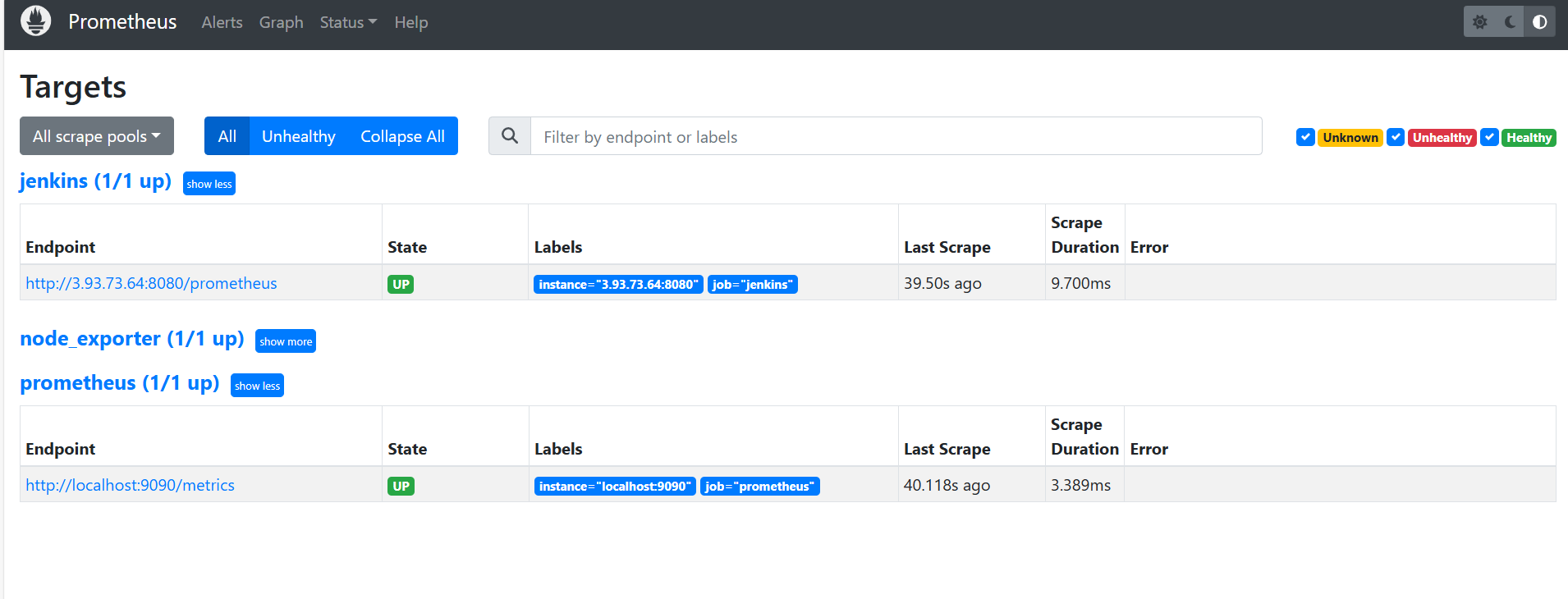
* **For Node Exporter you will see (0/1) in red colour. To resolve this, open Port number 9100 for Monitoring VM**

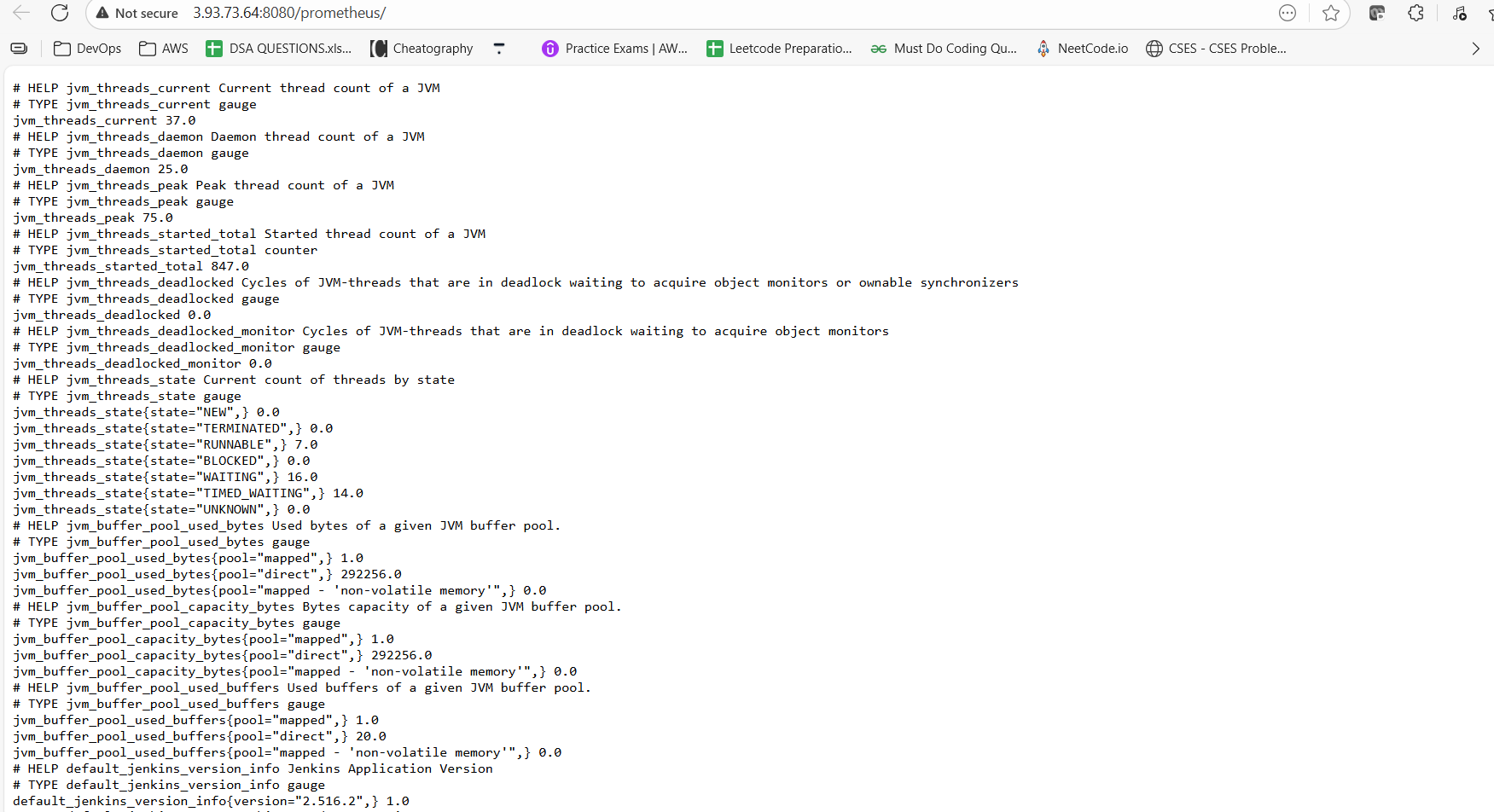
****

* **You should now see "Jenkins (1/1 up)" "node exporter (1/1 up)" and "prometheus (1/1 up)" in the prometheus browser.**

****

* **Click on "showmore" next to "jenkins." You will see a link. Open the link in new tab, to see the metrics that are getting scraped**

****

****

**Install Grafana (Execute in Monitoring Server VM)**

**-------------------------------------------------------------------**

**You are currently in /etc/Prometheus path.**

**Install Grafana on Monitoring Server;**

**Step 1: Install Dependencies:**

**First, ensure that all necessary dependencies are installed:**

**sudo apt-get update**

**sudo apt-get install -y apt-transport-https software-properties-common**

**Step 2: Add the GPG Key:**

**cd ---> You are now in ~ path**

**Add the GPG key for Grafana:**

**wget -q -O - https://packages.grafana.com/gpg.key | sudo apt-key add -**

**You should see OK when executed the above command.**

**Step 3: Add Grafana Repository:**

**Add the repository for Grafana stable releases:**

**echo "deb https://packages.grafana.com/oss/deb stable main" | sudo tee -a /etc/apt/sources.list.d/grafana.list**

**Step 4: Update and Install Grafana:**

**Update the package list and install Grafana:**

**sudo apt-get update**

**sudo apt-get -y install grafana**

**Step 5: Enable and Start Grafana Service:**

**To automatically start Grafana after a reboot, enable the service:**

**sudo systemctl enable grafana-server**

**Start Grafana:**

**sudo systemctl start grafana-server**

**Step 6: Check Grafana Status:**

**Verify the status of the Grafana service to ensure it's running correctly:**

**sudo systemctl status grafana-server**

**You should see "Active (running)" in green colour**

**Press control+c to come out**

**Step 7: Access Grafana Web Interface:**

**The default port for Grafana is 3000**

**Enable port no 3000 in inbound rules.**

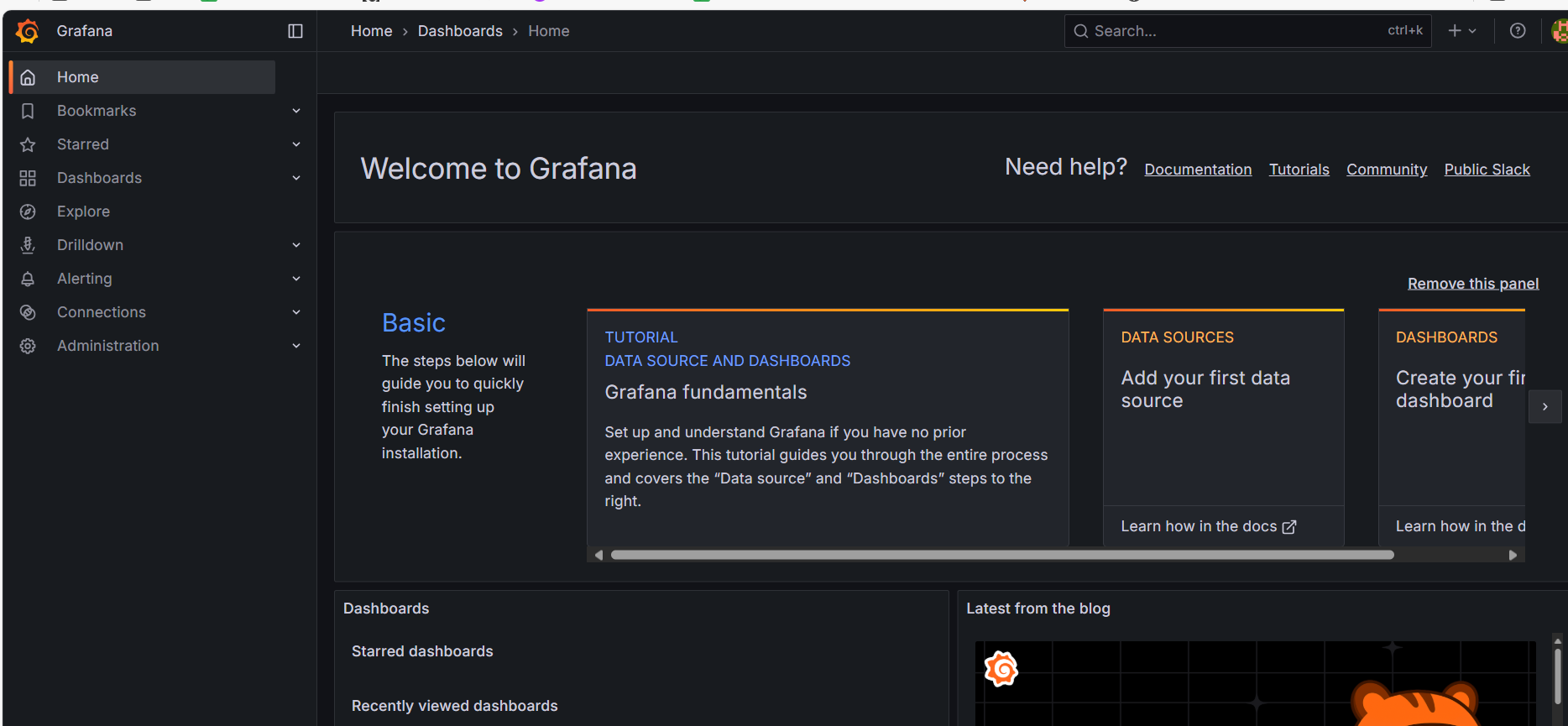
**http://<monitoring-server-ip>:3000**

**Default id and password is "admin"**

**You can Set new password or you can click on "skip now".**

**Click on "skip now" (If you want you can create the password)**

**You will see the Grafana dashboard**

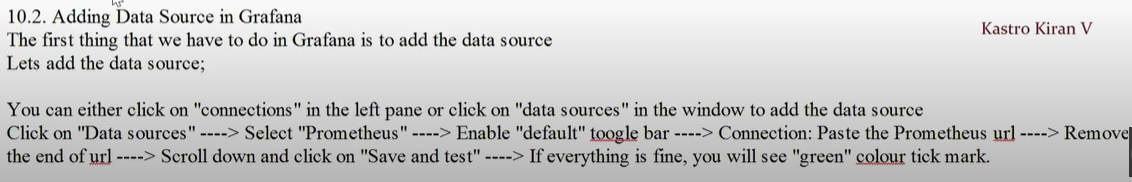
****

**# Adding Data Source in Grafana**

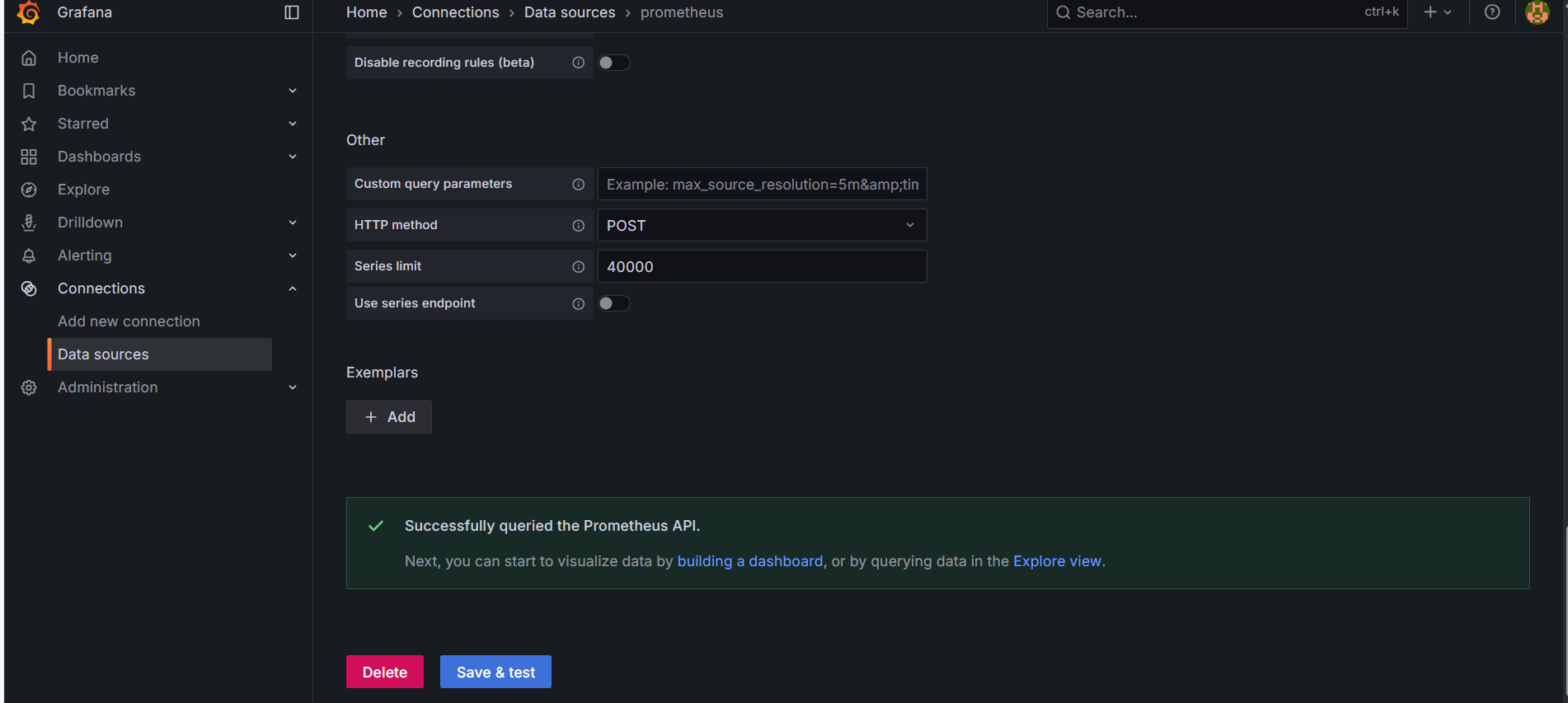
**The first thing that we have to do in Grafana is to add the data source**

**Add the data source.**

**Click on data source (home page)-> Import dashboard.**

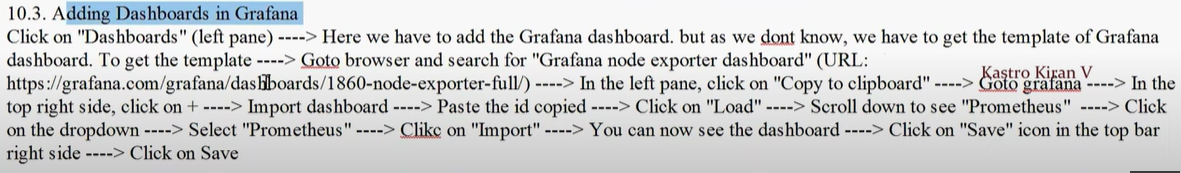
****

[**http://100.26.58.85:9090**](http://100.26.58.85:9090)

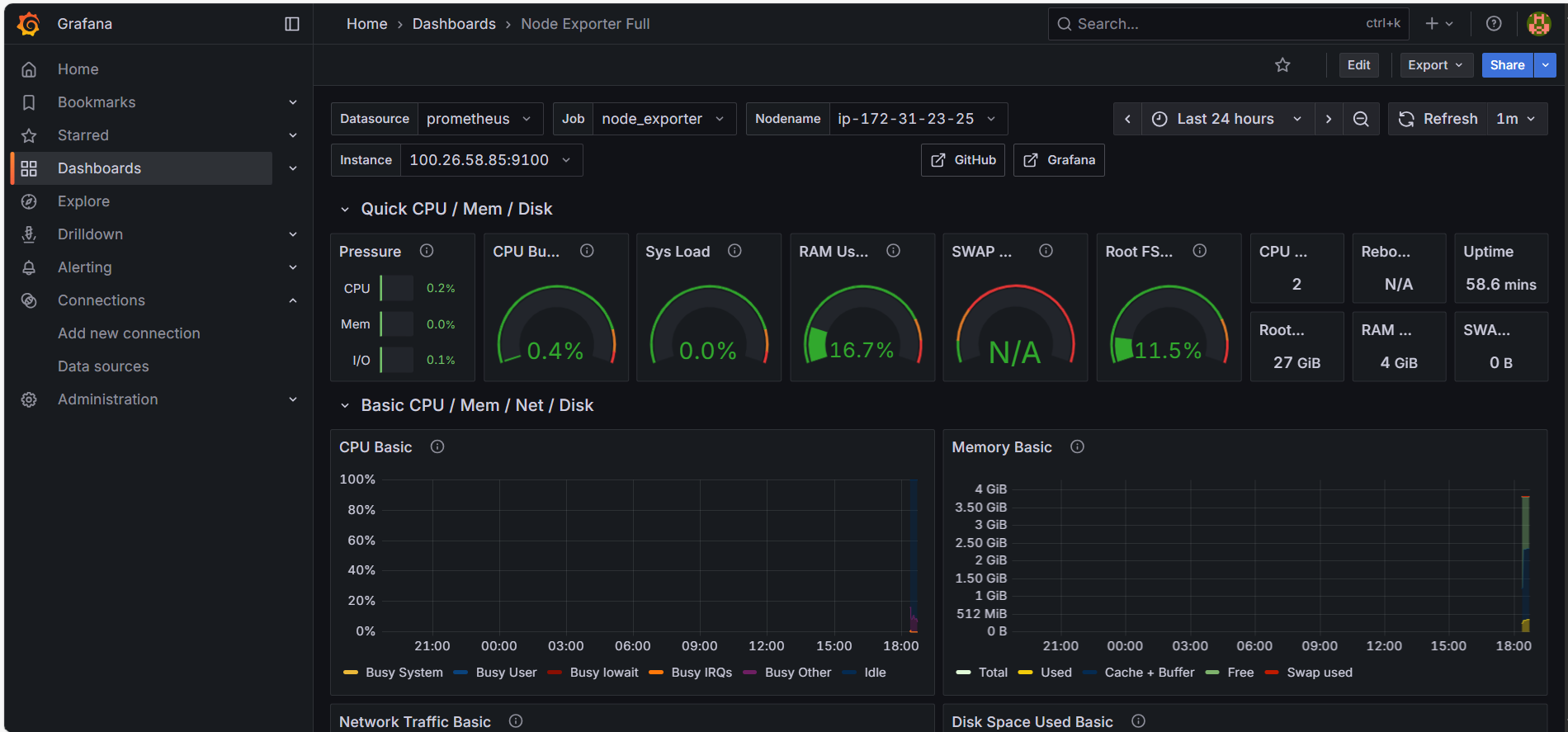
****

**# Adding Dashboards in Grafana**

**(URL:** [**https://grafana.com/grafana/dashboards/1860-node-exporter-full/**](https://grafana.com/grafana/dashboards/1860-node-exporter-full/)**)**

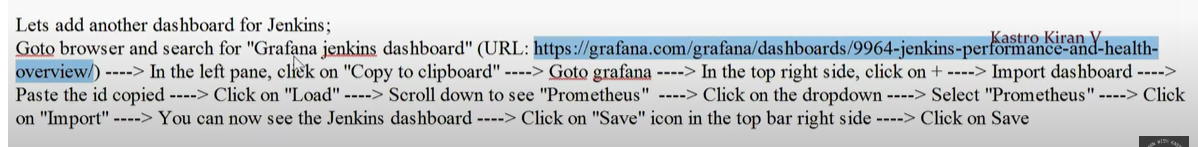
****

* **Go to dashboard -> Import dashboard -> Find and import dashboard : 1860 -> click on Load -> select data source (bottom) : prometheus -> Import**

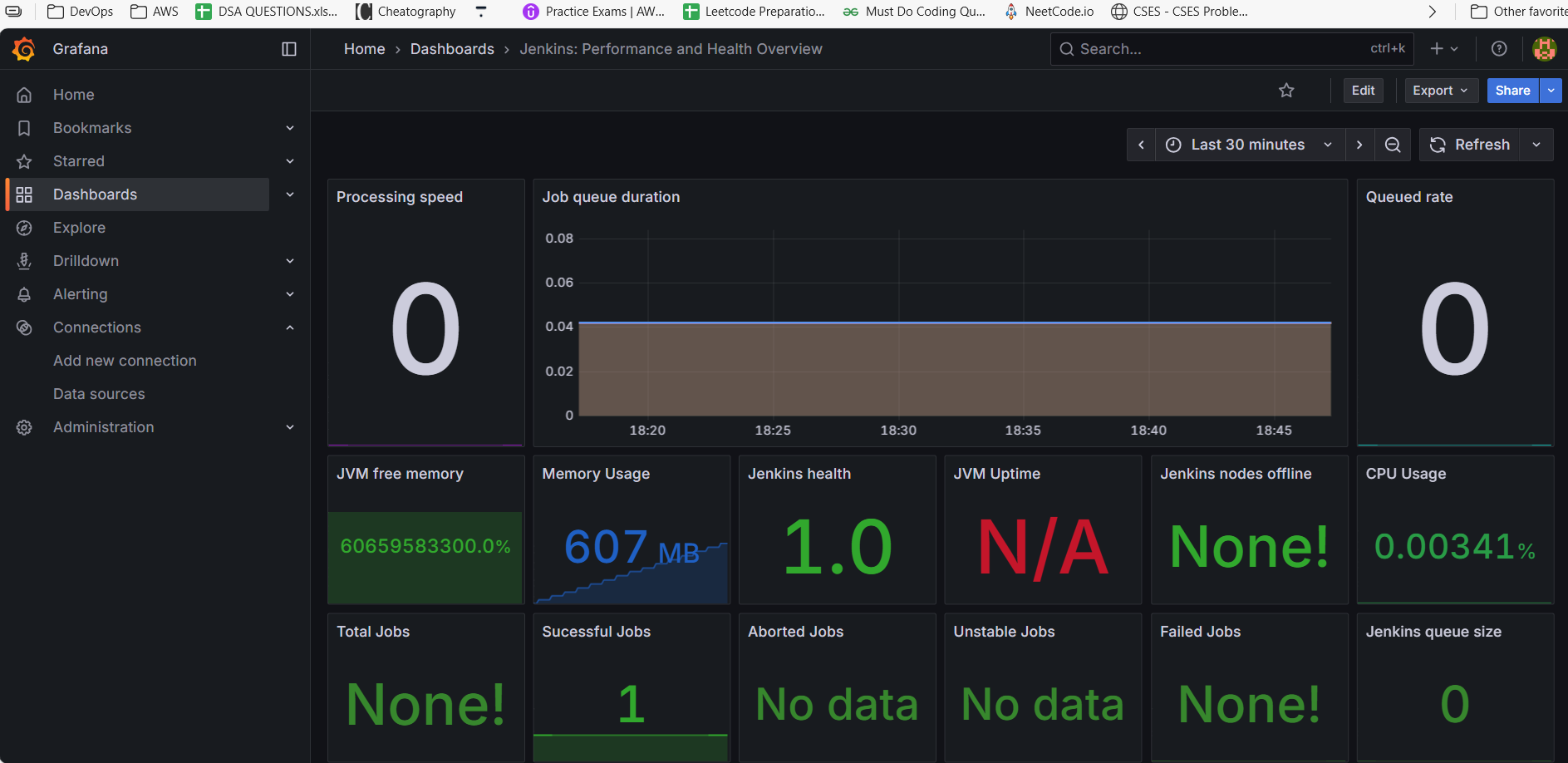
****

**Lets add another dashboard for Jenkins;**

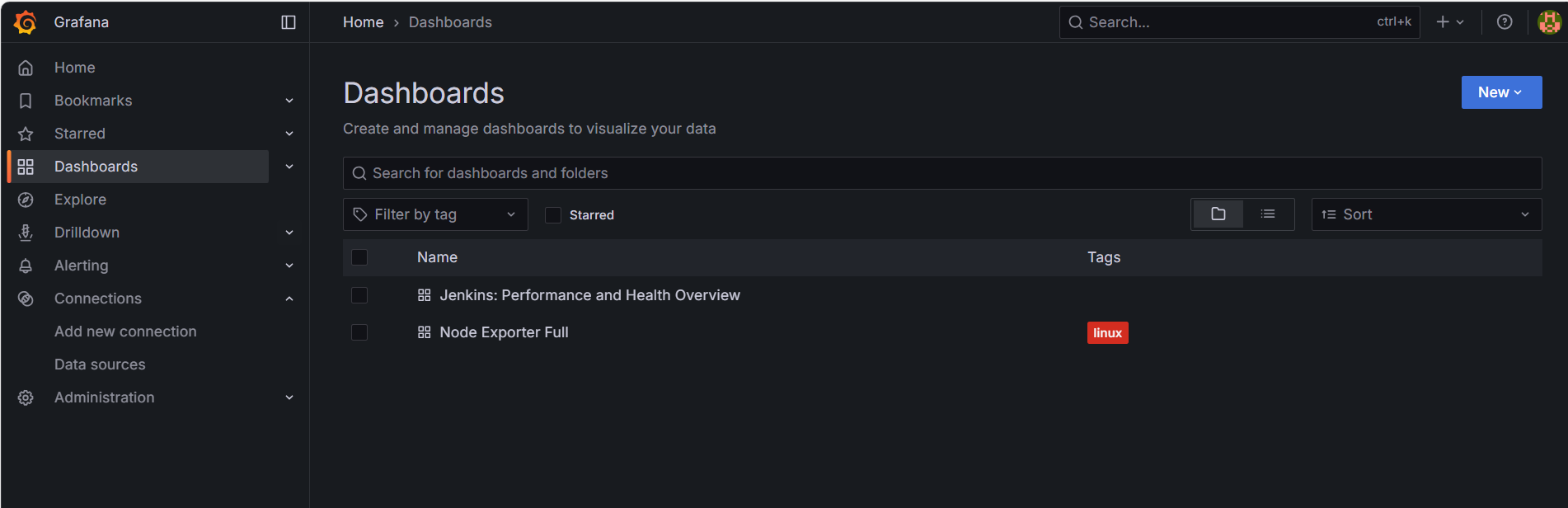
**(URL: https://grafana.com/grafana/dashboards/9964-jenkins-performance-and-health-overview/)**

****

* **Go to dashboard -> Import dashboard -> Find and import dashboard: 9964 -> click on load -> Select Prometheus data source -> Import**

****

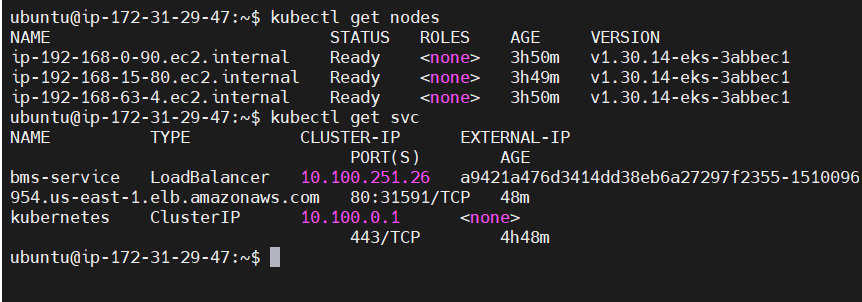
**Click on Dashboards in the left pane, you can see both the dashboards you have just added.**

****

* **Go to EC2 -> Load balancer -> we can see load balancer url, but we will not access this URL.**
* **Instead we will get from BMS -Server**
* **Execute below commands.**

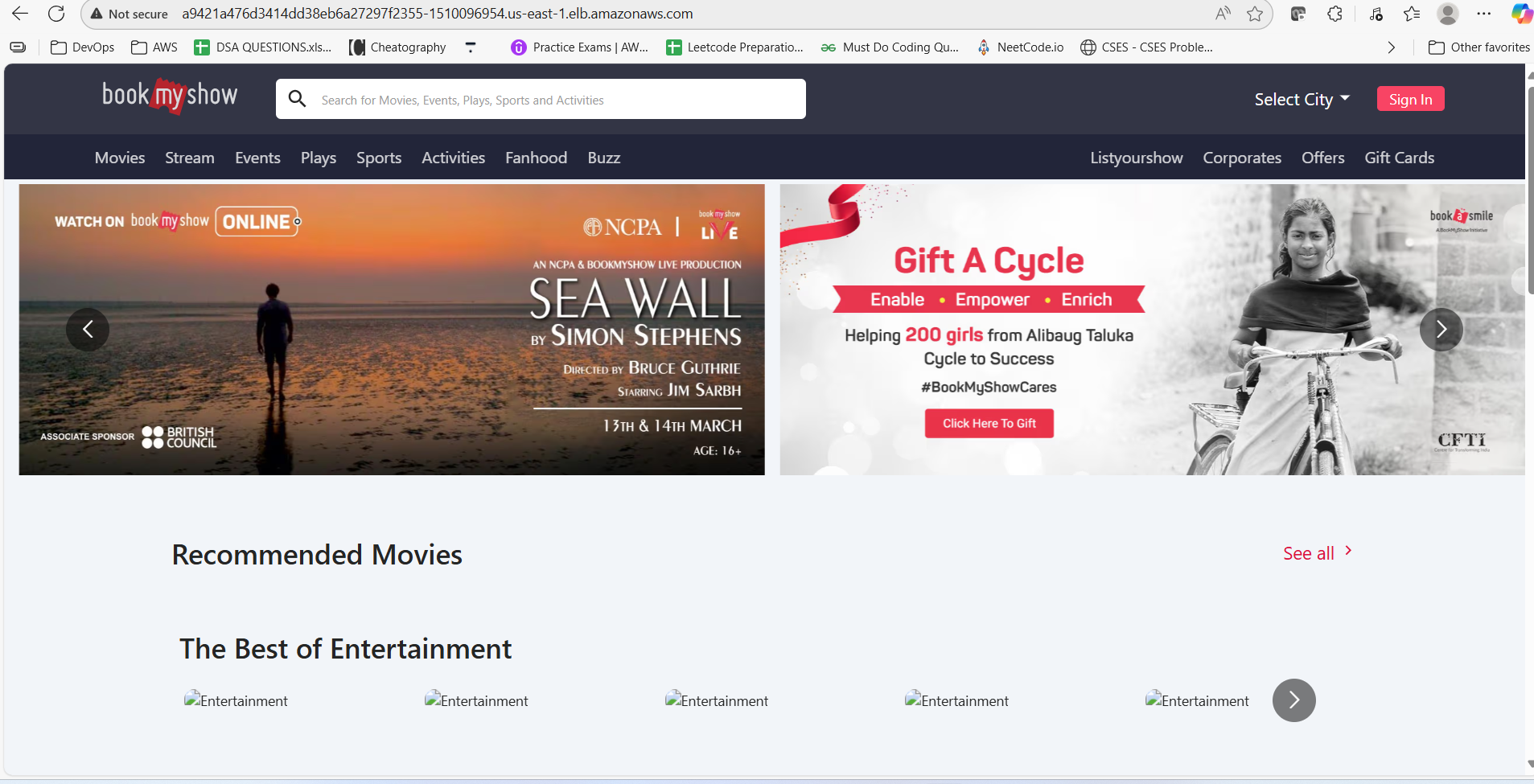
**$ kubectl get nodes**

**$ kubectl get svc**

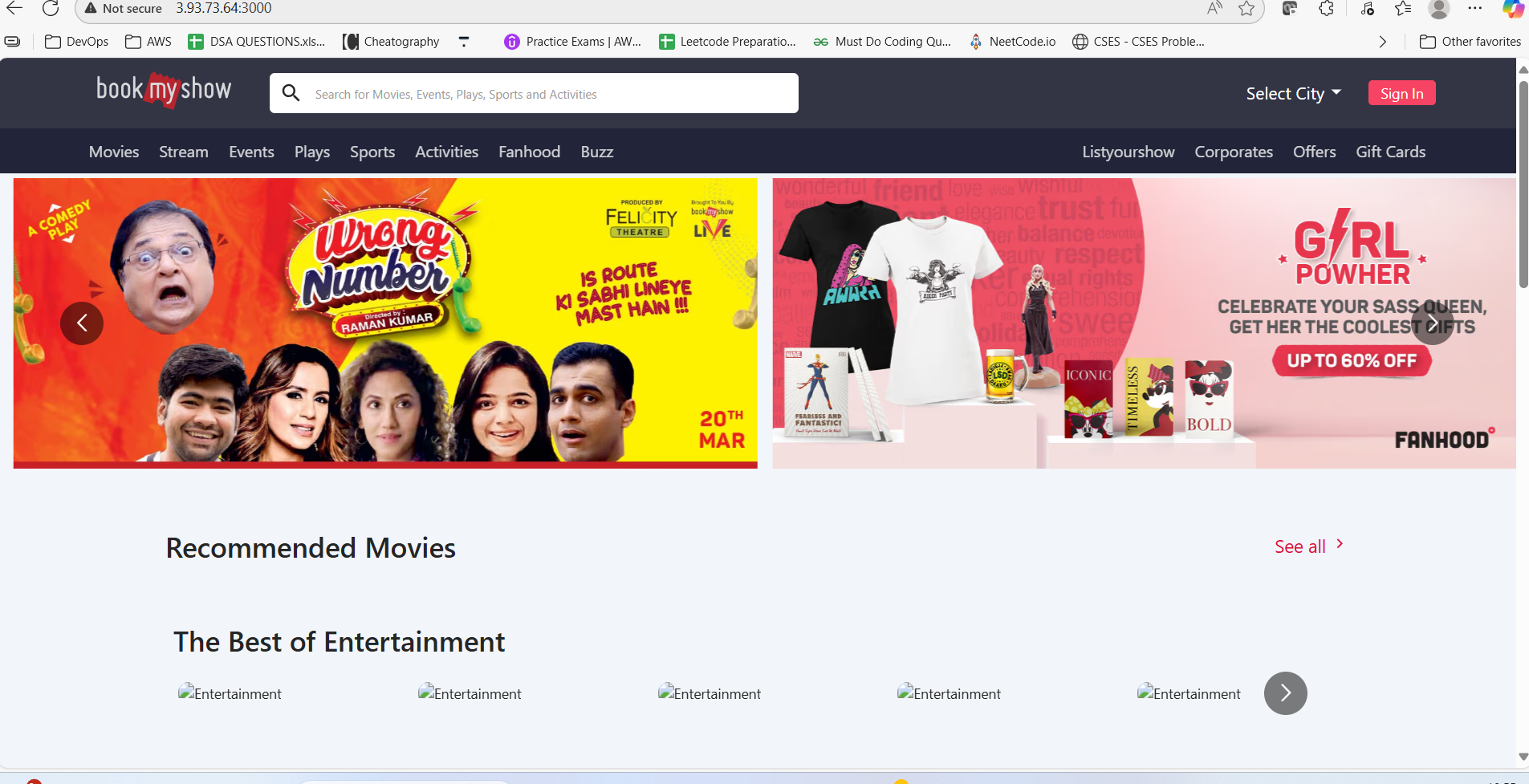
****

* **We will take the Load Balancer URL from here.**
* **Access the App through load balancer URL.**

**a9421a476d3414dd38eb6a27297f2355-1510096954.us-east-1.elb.amazonaws.com**

****

* **We have successfully deployed our application on K8S cluster.**
* **We can also access through IP address.**

****

**Final:**

**Delete all the resources created.**

**Firstly delete the node group, then delete the cluster, then terminate both EC2 instances (Monitoring server and BMS-server).**