Book My Show Clone Application Deployment using CI/CD Pipiline

In docker container and Eks cluster and accessing by load balancer url

Ellendula Shiva Teja

Launch 1 VM (Ubuntu, 24.04, t2.large, 28 GB, Name: BMS-Server)

Type	Protocol	Port range
SMTP	TCP	25
Custom TCP	TCP	3000-10000

(Used by various applications, such as Node.js (3000), Grafana (3000), Jenkins (8080), and custom web applications.

HTTP	TCP	80

Allows unencrypted web traffic. Used by web servers (e.g., Apache, Nginx) to serve websites over HTTP.

HTTPS	TCP	443

Allows secure web traffic using SSL/TLS.

SSH	TCP	22

Secure Shell (SSH) for remote server access.

Custom TCP	TCP	6443

Kubernetes API server port. Used for communication between kubectl, worker nodes, and the Kubernetes control plane.

SMTPS	TCP	465

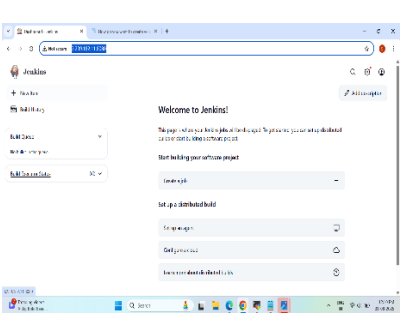
Secure Mail Transfer Protocol over SSL/TLS. Used for sending emails securely via SMTP with encryption.

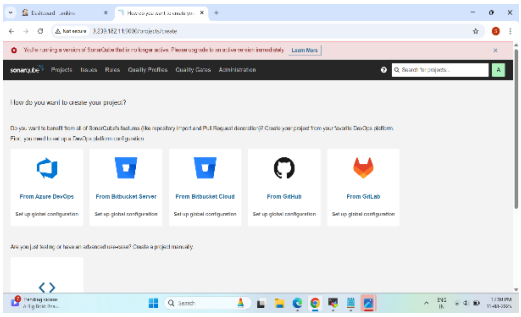
Custom TCP	TCP	30000-32767

INSTALL TOOLS LIKE DOCKER , JENKINS, TRIVY

Install SonarQube using docker with command

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





```
ubuntu@ip-10-0-1-41:~$ trivy --version
Version: 0.65.0
ubuntu@ip-10-0-1-41:~$
```

Install some plugings in Jenkins which is use for tools like docker, sonarqube,k8s etc

Plugings like : 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 the Jenkins

Add sonarqube, docker , email, creadentials is Jenkins

Add Jenkins creadentials in SonarQube for Code Quality Analysis

Configure system level tools in Jenkins in tools bar

Tools like: JDK, Sonar-Scanner, node.js, Docker, Dependency-Check

CREATION OF EKS CLUSTER

1.2.1. Creation of IAM user (To create EKS Cluster, its not recommended to create using Root Account)

1.2.2. Attach policies to the user

AmazonEC2FullAccess, AmazonEKS_CNI_Policy, AmazonEKSClusterPolicy, AmazonEKSWorkerNodePolicy, AWSCloudFormationFullAccess, IAMFullAccess

Attach the below inline policy also for the same user

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

1.2.3. Create Access Keys for the user created

With this we have created the IAM User with appropriate permissions to create the EKS Cluster

Connect to the 'BMS-Server' VM

```
sudo apt update
```

1.3.1. Install AWS CLI (to interact with AWS Account)

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

sudo apt install unzip

unzip awscliv2.zip

sudo ./aws/install

aws configure
```

Configure aws by executing below command

```
aws configure
```

1.3.2. Install KubeCTL (to interact with K8S)

```
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
```

1.3.3. Install EKS CTL (used to create EKS Cluster)

```
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
```

1.3.4. Create EKS Cluster

Execute the below commands as separate set

(a)

```
eksctl create cluster --name=kastro-eks \
        --region=us-east-1 \
        --zones=us-east-1a,us-east-1b \
```

```
        --version=1.30 \
        --without-nodegroup
```

It will take 5-10 minutes to create the cluster

Goto EKS Console and verify the cluster.

(b)

```
eksctl utils associate-iam-oidc-provider \
  --region us-east-1 \
  --cluster kastro-eks \
  --approve
```

The above command is crucial when setting up an EKS cluster because it enables IAM roles for service accounts (IRSA)

Amazon EKS uses OpenID Connect (OIDC) to authenticate Kubernetes service accounts with IAM roles.

Associating the IAM OIDC provider allows Kubernetes workloads (Pods) running in the cluster to assume IAM roles securely.

Without this, Pods in EKS clusters would require node-level IAM roles, which grant permissions to all Pods on a node.

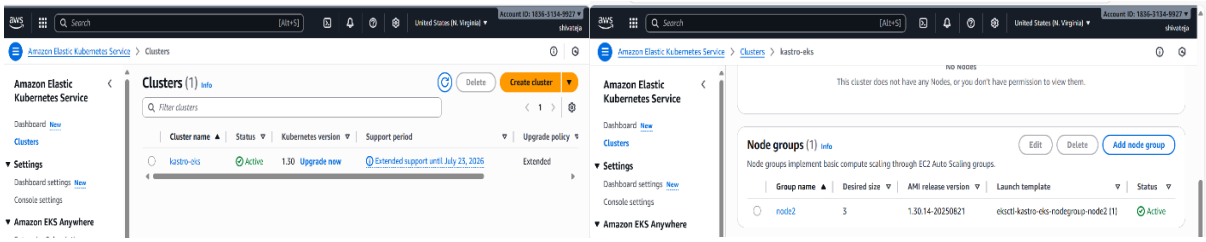
Without this, these services will not be able to access AWS resources securely.

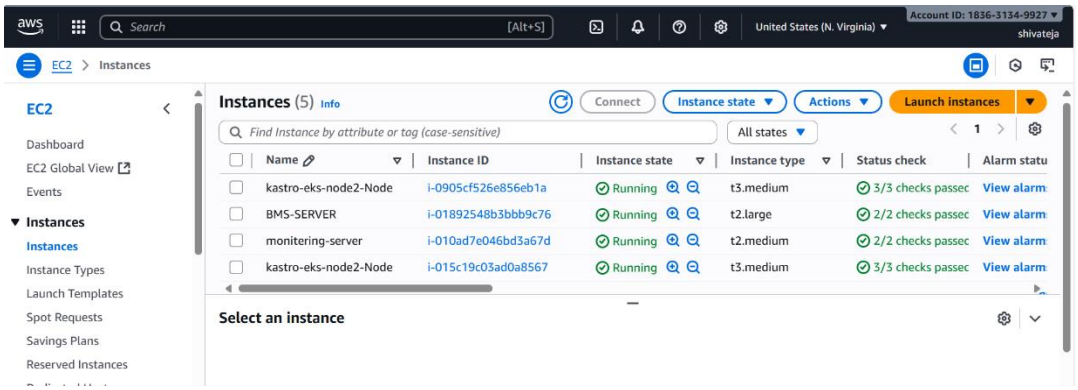
(c)

Before executing the below command, in the 'ssh-public-key' keep the '<PEM FILE NAME>' (dont give .pem. Just give the pem file name) which was used to create Jenkins Server

```
eksctl create nodegroup --cluster=kastro-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=BMSKEY.pem \
```

```
--managed \
--asg-access \
--external-dns-access \
--full-ecr-access \
--appmesh-access \
--alb-ingress-access
```





Install Node Port Manager

Using command sudo apt install npm

Now copy the pipeline script without k8s because 1st we are deploying in docker container from location and make changes if required then paste in Jenkins pipeline and build the pipepline and wait for the output

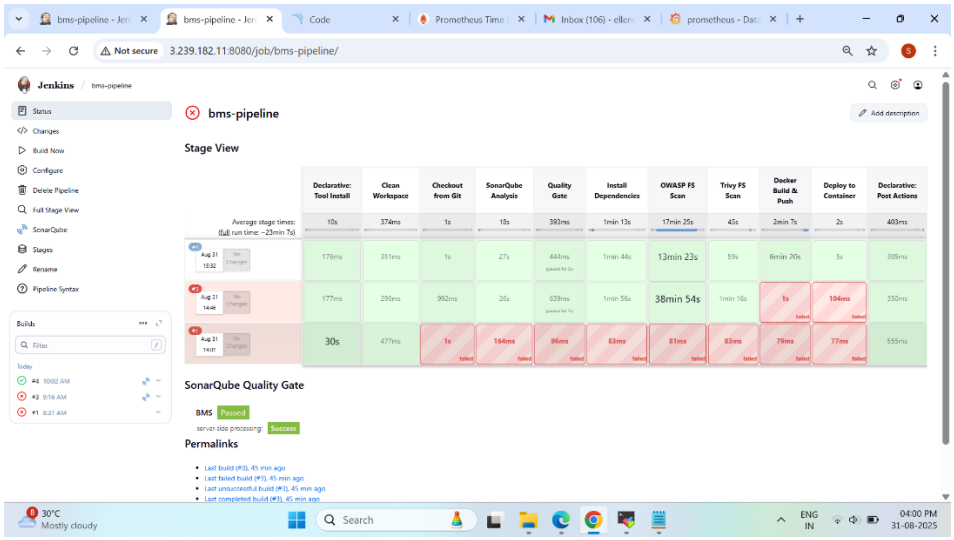
```
pipeline {
  agent any
  tools {
    jdk 'jdk17'
    nodejs 'node24'
```

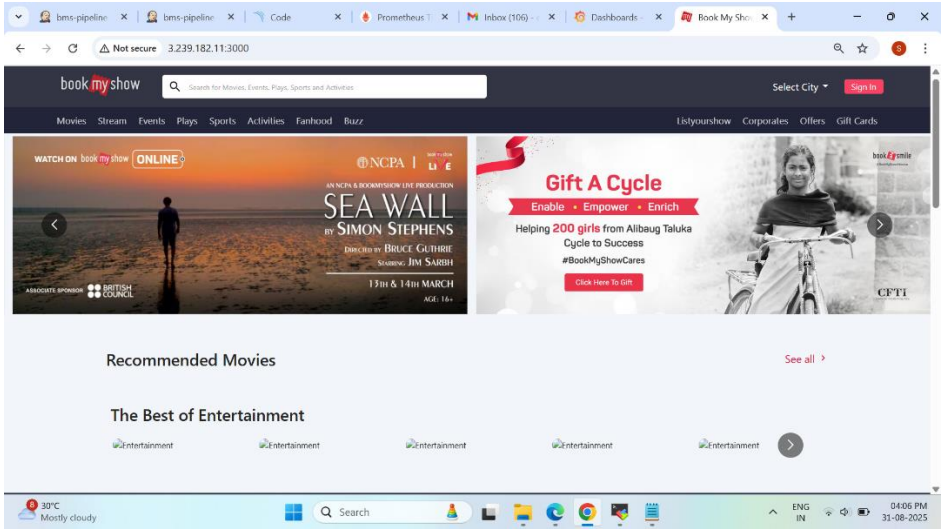
```
}
environment {
    SCANNER_HOME = tool 'sonar-scanner'
}
stages {
    stage('Clean Workspace') {
        steps {
            cleanWs()
        }
    }
    stage('Checkout from Git') {a
        steps {
          git branch: 'main', url: 'https://github.com/shivateja1234/Book-My-Show.git'
            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-hub', toolName: 'docker') {
                    sh '''
                    echo "Building Docker image..."
                    docker build --no-cache -t shivateja12/bms:latest -f bookmyshow-
app/Dockerfile bookmyshow-app

                    echo "Pushing Docker image to registry..."
                    docker push shivateja12/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 shivateja12/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: 'ellendulashivateja@gmail.com',
            attachmentsPattern: 'trivyfs.txt,trivyimage.txt'
    }
  }
}
```





Access the application using EC2 instance ip:3000

APPLICATION IS DEPLOYED IN DOCKER CONTAINER SUCCESSFULLY

2. DEPLOYING BMS CLONE IN EKS CLUSTER

Switch to Jenkins user and again configure AWS

To check wheather it is running in Jenkins or not

ps aux | grep jenkins

Switch to the jenkins user

sudo -su jenkins

pwd ---- /home/ubuntu

whoami ---- Jenkins

Verify the credentials

aws sts get-caller-identity

Comeout of the Jenkins user to Restart Jenkins

Exit

sudo systemctl restart jenkins

Switch to Jenkins user

sudo -su Jenkins

update the kubeconfig-file

aws eks update-kubeconfig --name kastro-eks --region us-east-1

Copy the pipeline with k8s and paste in Jenkins pipeline and build the pipeline

(with K8S Stage)

```
pipeline {
   agent any

   tools {
      jdk 'jdk17'
      nodejs 'node24'
   }

   environment {
      SCANNER_HOME = tool 'sonar-scanner'
      DOCKER_IMAGE = 'shivateja12/bms:latest'
      EKS_CLUSTER_NAME = 'kastro-eks'
      AWS_REGION = 'us-east-1'
   }

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

      stage('Checkout from Git') {
         steps {
            git branch: 'main', url: 'https://github.com/shivateja1234/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-hub', 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: 'ellendulashivateja@gmail.com',
            attachmentsPattern: 'trivyfs.txt'
    }
}
}
```

Meanwhile install monitering tools like promethus and Grafana

To install create EC2 instance with t2.medium

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

```
sudo apt update

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

----> esc ----> :wq ---->

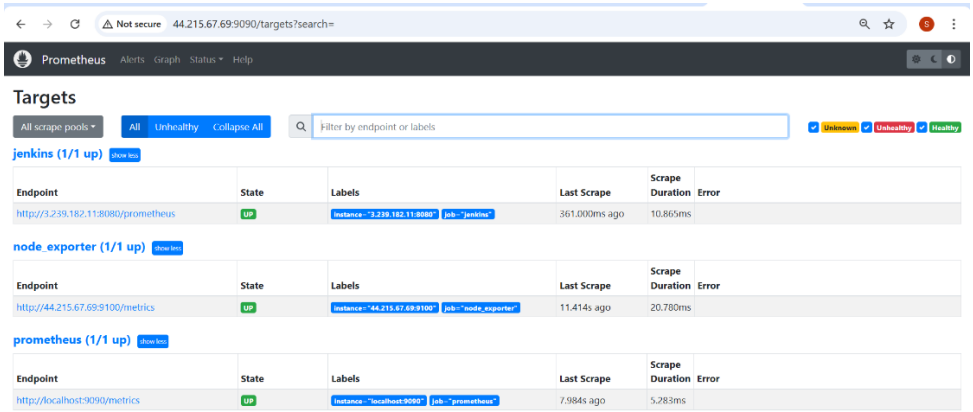
To automatically start the Prometheus after reboot run the below command

sudo systemctl enable prometheus

sudo systemctl start Prometheus

sudo systemctl status Prometheus

Access the Prometheus by EC2 ip:9090



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
```

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

sudo systemctl status node_exporter

we should integrate Jenkins with Prometheus for that we should add job in Prometheus.yaml file

vi Prometheus.yaml

```
 - job_name: 'node_exporter'
   static_configs:
     - targets: ['<MonitoringVMip>:9100']
 - job_name: 'jenkins'
   metrics_path: '/prometheus'
   static_configs:
     - targets: ['<your-jenkins-ip>:<your-jenkins-port>']
```

Reload the Prometheus by

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

Install grafana

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

To check

sudo systemctl enable grafana-server

sudo systemctl start grafana-server

sudo systemctl status grafana-server

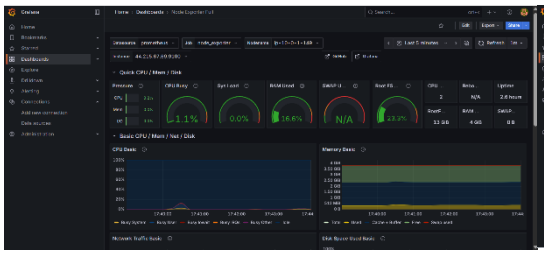
Step 7: Access Grafana Web Interface:

The default port for Grafana is 3000

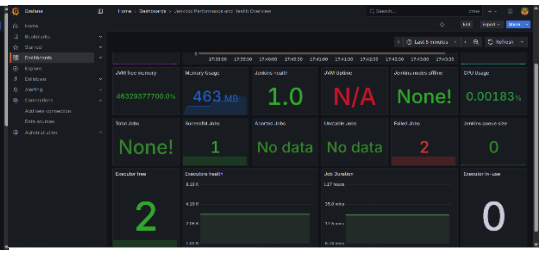
http://<monitoring-server-ip>:3000

Add dashboards for Prometheus and jenkins

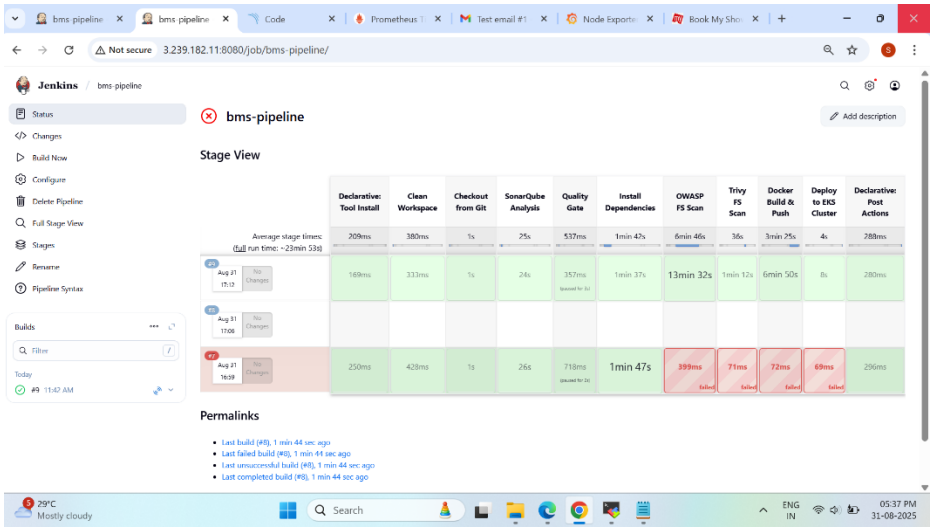
Node exporter



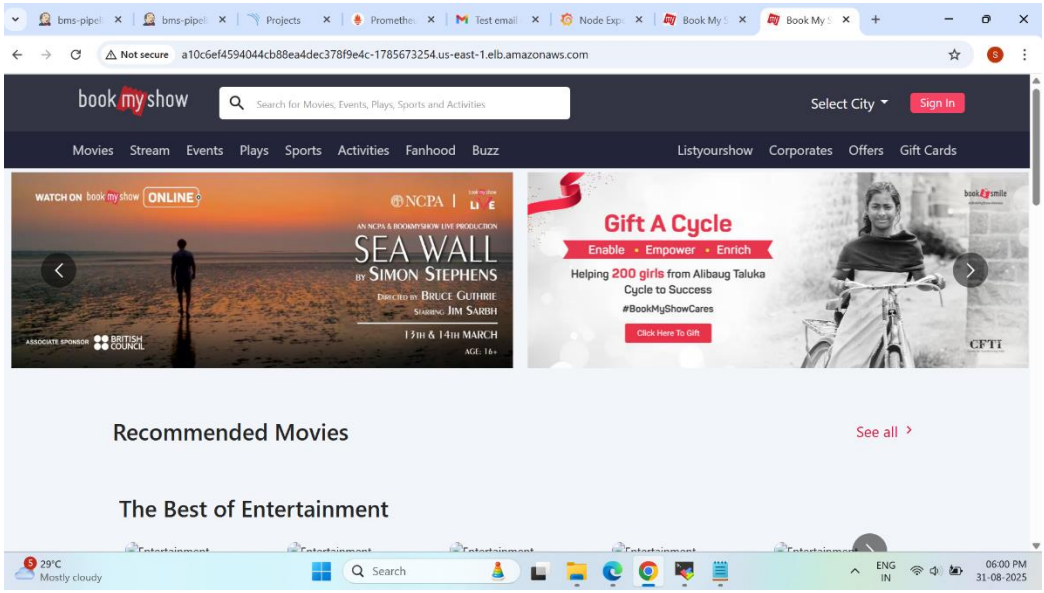
jenkins



The application is successfully deployed in EKS Cluster



The application accessed by load balancer url



To get load balancer url we use

Kubectl get nodes

kubectl get svc

```
ubuntu@ip-10-0-1-41:~$ kubectl get svc
NAME          TYPE           CLUSTER-IP       EXTERNAL-IP                                                              PORT(S
)        AGE
bms-service   LoadBalancer   10.100.171.194   a10c6ef4594044cb88ea4dec378f9e4c-1785673254.us-east-1.elb.amazonaws.com   80:309
36/TCP   4m18s
kubernetes    ClusterIP      10.100.0.1       <none>                                                                   443/TC
P        4h57m
ubuntu@ip-10-0-1-41:~$
```

THANK YOU