**Book My Show App - EC2 Setup and CI/CD Configuration**

**1. Launch an EC2 Instance**

* **Instance Type**: Ubuntu, **t2.large**
* **Instance Name**: BMS-Server

**2. Configure Security Group (Inbound Rules)**

| **Protocol** |  | **Port** | **Purpose** |
| --- | --- | --- | --- |
| SSH |  | 22 | Secure Shell access |
| HTTP |  | 80 | Web traffic |
| HTTPS |  | 443 | Secure web traffic |
| Node.js |  | 3000 | Application service |
| Jenkins |  | 8080 | CI/CD Pipeline |
| SonarQube |  | 9000 | Code quality analysis |

**3. Connect to the Instance**

* Open a terminal and use SSH to connect:

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

**4. Install Required Tools**

**4.1 Install Jenkins**

* Create a script to automate Jenkins installation:

vim jenkins.sh

* Paste the following content:

#!/bin/bash

sudo apt update

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

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

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

echo deb [signed-by=/usr/share/keyrings/jenkins-keyring.asc] \

https://pkg.jenkins.io/debian-stable binary/ | sudo tee \

/etc/apt/sources.list.d/jenkins.list > /dev/null

sudo apt-get update

sudo apt-get install jenkins -y

* Save and exit (ESC → :wq)
* Grant execution permission and run the script:

sudo chmod +x jenkins.sh

./jenkins.sh

**4.2 Install Docker**

* Create a script:

vim docker.sh

* Paste the following:

#!/bin/bash

sudo apt-get update

sudo apt-get install -y ca-certificates curl

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

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

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

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

sudo apt-get update

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

* Save and exit (ESC → :wq)
* Grant execution permission and run:

sudo chmod +x docker.sh

./docker.sh

* If you encounter permission issues when pulling images, run:

sudo chmod 666 /var/run/docker.sock

**4.3 Install Trivy (Security Scanner)**

* Create a script:

vim trivy.sh

* Paste the following:

#!/bin/bash

sudo apt-get install -y 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 -y trivy

* Save and exit (ESC → :wq)
* Grant execution permission and run:

sudo chmod +x trivy.sh

./trivy.sh

* Verify the installation:

trivy --version

**5. Run SonarQube Container**

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

docker images

docker ps

**6. Clone GitHub Repository**

git clone https://github.com/rageshvk/Book-My-Show-App.git

**7. Configure Docker Hub**

docker login -u rageshvk

# Enter your password when prompted

**8. Access SonarQube Dashboard**

* Open http://your-ec2-ip:9000
* Default credentials:
  + **Username**: admin
  + **Password**: admin
* Set a new password after the first login.

**9. Access Jenkins Dashboard**

* Open http://your-ec2-ip:8080
* Get the initial admin password:

cat /var/lib/jenkins/secrets/initialAdminPassword

* Copy the password and paste it into the Jenkins setup page.
* Create a new **admin user**.

**10. Install Jenkins Plugins**

* Go to **Manage Jenkins** → **Plugins** → **Install** the following:
  + Eclipse Temurin Installer
  + SonarQube Scanner
  + NodeJS
  + Docker
  + Docker Commons
  + Docker Pipeline
  + Docker API
  + docker-build-step
  + Pipeline Stage View
  + Prometheus Metrics

**11. Configure SonarQube Token in Jenkins**

1. Go to SonarQube: http://your-ec2-ip:9000
2. **Administration** → **Security** → **Users**
3. Click on the **four dash to create a token**.
4. **Name**: sonar-token
5. Click **Generate** and copy the token.
6. **In Jenkins**:
   * **Manage Jenkins** → **Security** → **Credentials** → **Global**
   * **Add Credential**
     + **Kind**: Secret Text
     + **Scope**: Global
     + **Secret**: Paste the token
     + **ID**: sonar-token
     + **Description**: SonarQube Token
     + **Create**

**12. Configure SonarQube Webhook**

* In **SonarQube Dashboard**:
  + **Administration** → **Configuration** → **Webhooks** → **Create**
  + **Name**: Jenkins
  + **URL**: http://your-ec2-ip:8080/sonarqube-webhook/
  + **Create**

*(Ensure the trailing / at the end of the webhook URL, otherwise, it won’t work.)*

**13. Configure Docker Hub Credentials in Jenkins**

* **Manage Jenkins** → **Security** → **Credentials** → **Global**
* **Add Credential**
  + **Kind**: Username & Password
  + **Scope**: Global
  + **Username**: your-dockerhub-username
  + **Password**: your-dockerhub-password
  + **ID**: docker
  + **Description**: Docker Hub Credentials
  + **Create**

**14. Configure Jenkins Tools**

* **Manage Jenkins** → **System Configuration** → **Tools**
* **JDK**:
  + Name: jdk17
  + Install automatically from adoptium.net
  + Version: JDK 17.0.8.1+1
* **SonarQube Scanner**:
  + Name: sonar-scanner
  + Install automatically
  + Version: 7.0.1.4817
* **NodeJS**:
  + Name: node23
  + Install automatically
  + Version: 23.7.0
* **Docker**:
  + Name: docker
  + Install automatically from docker.com
  + Version: latest

**15. Configure SonarQube in Jenkins**

* **Manage Jenkins** → **System** → **SonarQube Servers**
  + **Add SonarQube**
  + **Name**: sonar-server
  + **Server URL**: http://your-ec2-ip:9000
  + **Authentication Token**: sonar-token
  + **Save & Apply**

**Create dockerfile:**

Create docker fille:

FROM node:18

WORKDIR /app

COPY package.json package-lock.json ./

RUN npm install postcss@8.4.21 postcss-safe-parser@6.0.0 --legacy-peer-deps

RUN npm install

COPY . .

EXPOSE 3000

ENV NODE\_OPTIONS=--openssl-legacy-provider

ENV PORT=3000

CMD ["npm", "start"]

**Create Jenkinsfile:**

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 'https://github.com/rageshvk/Book-My-Show-App.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('Trivy FS Scan') {

steps {

sh 'trivy fs bookmyshow-app'

}

}

stage('Docker Build & Push') {

steps {

script {

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

sh '''

echo "Building Docker image..."

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

echo "Pushing Docker image to registry..."

docker push rageshvk/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 rageshvk/bms:latest

echo "Checking running containers..."

docker ps -a

echo "Fetching logs..."

sleep 5 # Give time for the app to start

docker logs bms

'''

}

}

}

}

**Create Pipeline Job:**

1. Go to **Jenkins Dashboard** > Click **New Item**
2. Enter **Job Name**: BMS-APP > Select **Pipeline** > Click **OK**
3. Under **Build Triggers**, check **GitHub hook trigger for GITScm polling**
4. Go to **Pipeline** > Under **Definition**, select **Pipeline script from SCM**
5. Under **SCM**, choose **Git** and enter the **Git Repository URL**
6. Set **Branch** to master
7. Set **Script Path** to Jenkinsfile
8. Click **Apply** and **Save**
9. Go to **BMS-APP Job Page** and click **Build Now** to run the pipeline

**Create Load Balancer and Auto Scaling Group:**

**Create Target Group:**

1. Go to **EC2 Dashboard** > **Load Balancing** > **Target Groups**
2. Click **Create Target Group**
3. Under **Basic Configuration**:
   * Choose **Target Type**: **Instance**
   * Enter **Target Group Name**: bms-server-tg
   * **Protocol**: HTTP, **Port**: 3000
   * **VPC**: Default
4. Click **Next**
5. Under **Available Instances**, select bms-server
6. Click **Include as Pending Below**
7. Click **Create Target Group**

**Create Load Balancer:**

1. Navigate to **Load Balancers** > Click **Create Load Balancer**
2. Select **Application Load Balancer** > Click **Create**
3. Enter **Load Balancer Name**: bms-server-alb
4. Select **Internet-facing**
5. Choose **Two Availability Zones and Subnets**
6. Select **bms-server Security Group**
7. Under **Listeners and Routing**, set **Default Action** to the previously created **Target Group**
8. Click **Create Load Balancer**
9. Once the load balancer becomes active, you can access the application using the **Load Balancer DNS Name**

**Create Auto Scaling Group:**

**Create an AMI (Image) and Launch Template:**

1. Go to **EC2 Dashboard** > Select bms-server
2. Click **Actions** > **Create Image (AMI)**
3. Enter **Image Name**: bms-ami > Click **Create**

**Create Launch Template:**

1. Navigate to **Launch Templates** > Click **Create Launch Template**
2. Enter **Launch Template Name**: bms-server-tmplt
3. Select **AMI**: bms-ami
4. Choose **Instance Type**: t2.large
5. Select **Key Pair** and **Security Group**
6. Click **Create Launch Template**

**Create Auto Scaling Group:**

1. Navigate to **Auto Scaling Groups** > Click **Create Auto Scaling Group**
2. Enter **Auto Scaling Group Name**: bms-server-asg
3. Select **Launch Template**: bms-server-tmplt
4. Click **Next**
5. Choose **VPC**: Default
6. Select **Availability Zones and Subnets**
7. Click **Next**
8. Click **Attach to an Existing Load Balancer**
9. Select the **Existing Target Group** (bms-server-tg)
10. Click **Next**
11. Set:
    * **Desired Capacity**: 2
    * **Minimum Capacity**: 2
    * **Maximum Capacity**: 4
12. Click **Next** > **Next**
13. Under **Tags**, add:
    * **Key**: Name, **Value**: bms-server
14. Click **Next** > **Create Auto Scaling Group**

Now, you will see **two instances being created**.

**Monitoring the application:**

Launch Ubuntu VM, 22.04, t2.medium,

Name the VM as Monitoring Server

**1. Connect to the Monitoring Server VM (Execute in Monitoring Server VM)**

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

With the above command, we have created a 'Prometheus' user

Explanation of above command

–system – Will create a system account.

–no-create-home – We don’t need a home directory for Prometheus or any other system accounts in our case.

–shell /bin/false – It prevents logging in as a Prometheus user.

Prometheus – Will create a Prometheus user and a group with the same name.

**2. 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

**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>

If it doesn't work, in the web link of browser, remove 's' in 'https'. Keep only 'http' and now you will be able to see.

You can see the Prometheus console.

Click on 'Status' dropdown ---> Click on 'Targets' ---> You can see 'Prometheus (1/1 up)' ----> It scrapes itself every 15 seconds by default.

**10. 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

**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

**3. 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

**4. 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

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

**10.2. 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;

**10.3. Adding Dashboards in Grafana**

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

Lets add another dashboard for Jenkins;

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

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