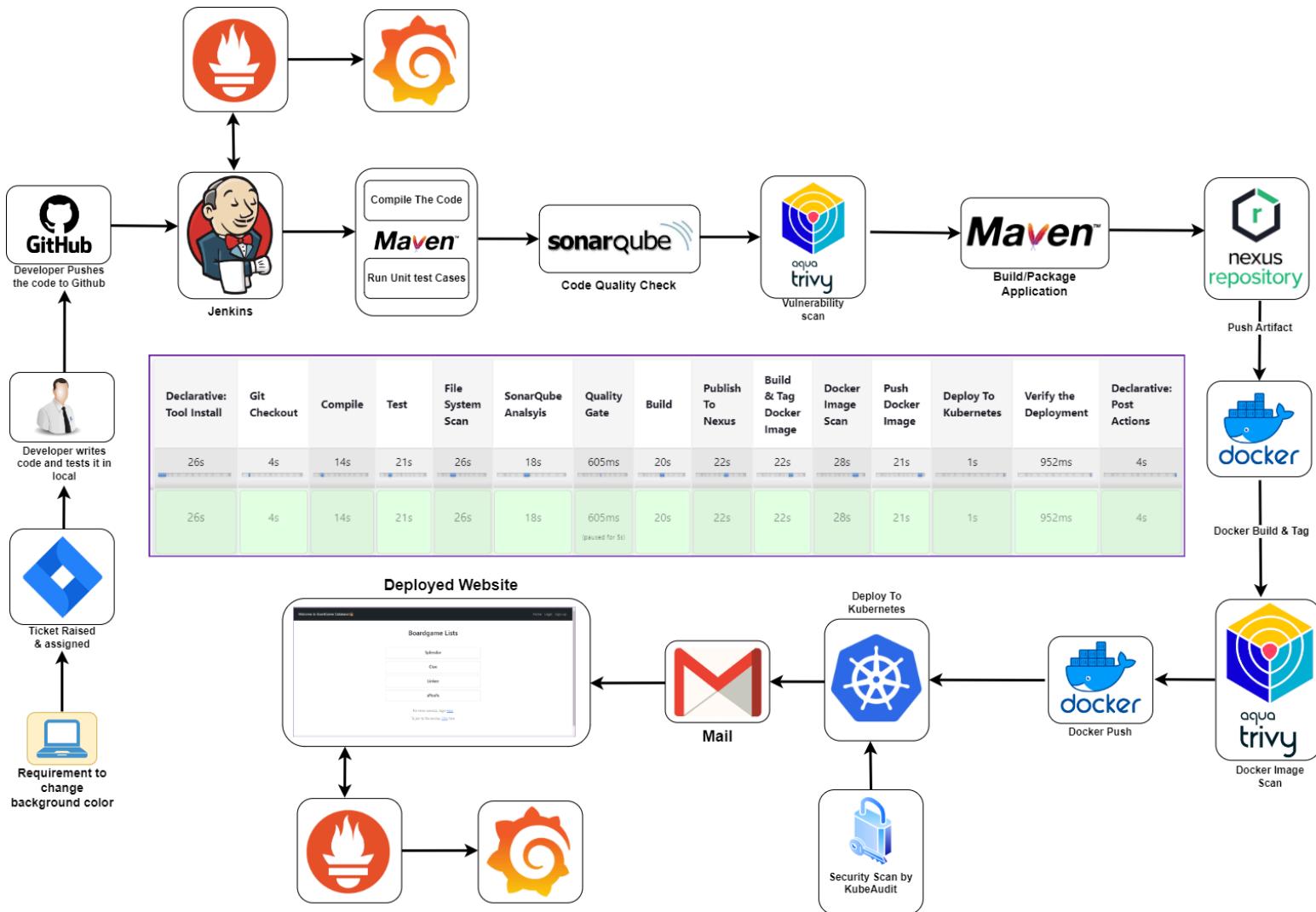
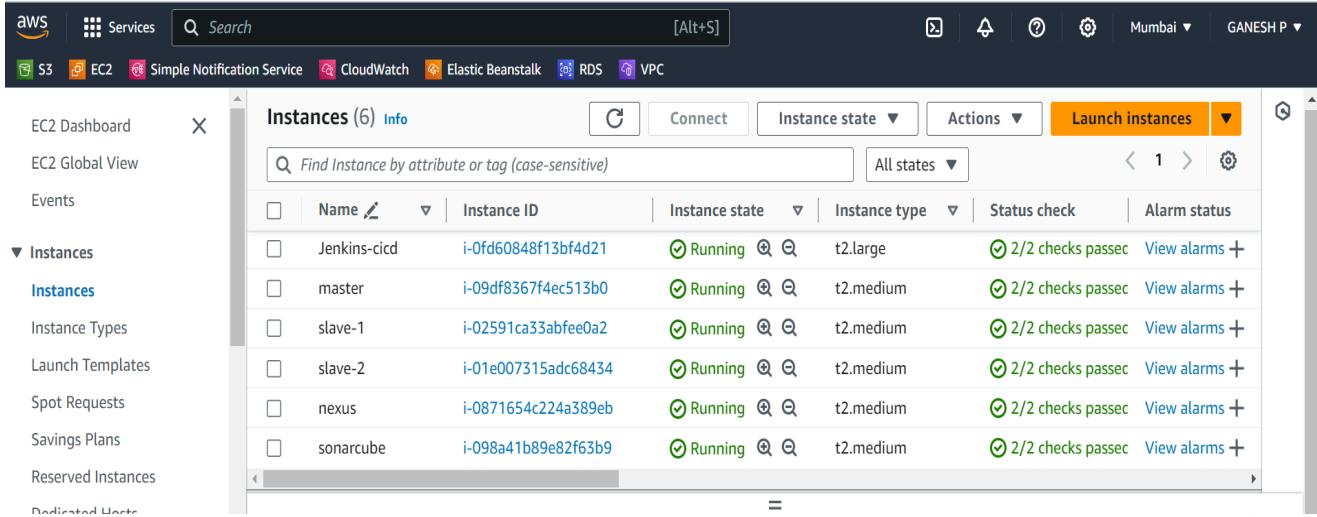


# THE ULTIMATE CICD DEVOPS PIPELINE PROJECT



# PHASE-1 | Setup Infra



The screenshot shows the AWS Management Console with the EC2 service selected. The main pane displays a list of EC2 instances, each with a checkbox, name, instance ID, state, type, status checks, and alarm status. All instances are currently running. The sidebar on the left shows other EC2-related options like Global View, Events, and Instance Types.

Name	Instance ID	Instance state	Instance type	Status check	Alarm status
Jenkins-cicd	i-0fd60848f13bf4d21	Running	t2.large	2/2 checks passed	<a href="#">View alarms +</a>
master	i-09df8367f4ec513b0	Running	t2.medium	2/2 checks passed	<a href="#">View alarms +</a>
slave-1	i-02591ca33abfee0a2	Running	t2.medium	2/2 checks passed	<a href="#">View alarms +</a>
slave-2	i-01e007315adc68434	Running	t2.medium	2/2 checks passed	<a href="#">View alarms +</a>
nexus	i-0871654c224a389eb	Running	t2.medium	2/2 checks passed	<a href="#">View alarms +</a>
sonarcube	i-098a41b89e82f63b9	Running	t2.medium	2/2 checks passed	<a href="#">View alarms +</a>

To create an Ubuntu EC2 instance in AWS, follow these steps:

**1. Sign in to the AWS Management Console:**

- Go to the AWS Management Console at <https://aws.amazon.com/console/>.
- Sign in with your AWS account credentials.

**2. Navigate to EC2:**

- Once logged in, navigate to the EC2 dashboard by typing "EC2" in the search bar at the top or by selecting "Services" and then "EC2" under the "Compute" section.

**3. Launch Instance:**

- Click on the "Instances" link in the EC2 dashboard sidebar.
- Click the "Launch Instance" button.

**4. Choose an Amazon Machine Image (AMI):**

- In the "Step 1: Choose an Amazon Machine Image (AMI)" section, select "Ubuntu" from the list of available AMIs.
- Choose the Ubuntu version you want to use. For example, "Ubuntu Server 24.04 LTS".
- Click "Select".

**5. Choose an Instance Type:**

- In the "Step 2: Choose an Instance Type" section, select the instance type that fits your requirements. The default option (usually a t2.micro instance) is suitable for testing and small workloads.
- Click "Next: Configure Instance Details".

## 6. Configure Instance Details:

- Optionally, configure instance details such as network settings, subnets, IAM role, etc. You can leave these settings as default for now.
- Click "Next: Add Storage".

## 7. Add Storage:

- Specify the size of the root volume (default is usually fine for testing purposes).
- Click "Next: Add Tags".

## 8. Add Tags:

- Optionally, add tags to your instance for better organization and management.
- Click "Next: Configure Security Group".

## 9. Configure Security Group:

- In the "Step 6: Configure Security Group" section, configure the security group to allow SSH access (port 22) from your IP address.
- You may also want to allow other ports based on your requirements (e.g., HTTP, HTTPS) as in this pic

Inbound rules (8)				
Type	Protocol	Port range	Source	
SMTP	TCP	25	0.0.0.0/0	
Custom TCP	TCP	3000 - 10000	0.0.0.0/0	
HTTP	TCP	80	0.0.0.0/0	
HTTPS	TCP	443	0.0.0.0/0	
SSH	TCP	22	0.0.0.0/0	
Custom TCP	TCP	6443	0.0.0.0/0	
SMTPS	TCP	465	0.0.0.0/0	
Custom TCP	TCP	30000 - 32767	0.0.0.0/0	

- Click "Review and Launch".

## 10. Review and Launch:

- Review the configuration of your instance.
- Click "Launch".

## 11. Select Key Pair:

- In the pop-up window, select an existing key pair or create a new one.
- Check the acknowledgment box.

- Click "Launch Instances".

12.

**Access Your Instance:**

- Use Mobaxterm

# **Setup K8-Cluster using kubeadm**

## **[K8 Version-->1.28.1]**

### **1. Update System Packages [On Master & Worker Node]**

```
sudo apt-get update
```

### **2. Install Docker[On Master & Worker Node]**

```
sudo apt install docker.io -y  
sudo chmod 666 /var/run/docker.sock
```

### **3. Install Required Dependencies for Kubernetes[On Master & Worker Node]**

```
sudo apt-get install -y apt-transport-https ca-certificates curl gnupg  
sudo mkdir -p -m 755 /etc/apt/keyrings
```

### **4. Add Kubernetes Repository and GPG Key[On Master & Worker Node]**

```
curl -fsSL https://pkgs.k8s.io/core:/stable:/v1.28/deb/Release.key | sudo gpg --  
dearmor -o /etc/apt/keyrings/kubernetes-apt-keyring.gpg  
echo 'deb [signed-by=/etc/apt/keyrings/kubernetes-apt-keyring.gpg]  
https://pkgs.k8s.io/core:/stable:/v1.28/deb/ /' | sudo tee  
/etc/apt/sources.list.d/kubernetes.list
```

### **5. Update Package List[On Master & Worker Node]**

```
sudo apt update
```

### **6. Install Kubernetes Components[On Master & Worker Node]**

```
sudo apt install -y kubeadm=1.28.1-1.1 kubelet=1.28.1-1.1 kubectl=1.28.1-1.1
```

kubeadm - Going to setup a Kubernetes cluster  
kubelet - Responsible for creating pods which we are going to deploy applications  
kubectl - will work as cli to interact with the k8s cluster

### **7. Initialize Kubernetes Master Node [On MasterNode]**

```
sudo kubeadm init --pod-network-cidr=10.244.0.0/16
```

After run the above command then our vm will acts as master node and it will generate token to connect this with slave node -copy the token and run the command in slave machines 1 & 2

## 8. Configure Kubernetes Cluster [On MasterNode]

```
mkdir -p $HOME/.kube  
sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config  
sudo chown $(id -u):$(id -g) $HOME/.kube/config
```

## 9. Deploy Networking Solution (Calico) [On MasterNode]

```
kubectl apply -f https://docs.projectcalico.org/v3.20/manifests/calico.yaml
```

## 10. Deploy Ingress Controller (NGINX) [On MasterNode]

```
kubectl apply -f https://raw.githubusercontent.com/kubernetes/ingress-nginx/controller-v0.49.0/deploy/static/provider/baremetal/deploy.yaml
```

11. We'll Scan Kubernetes Cluster For Any Kind Of Issues For That We Have Multiple Tools Like Trivy Also We Have But Trivy May Not Work Always So For That Reason We Are Just Going To Go With Cube Audit It Also A Tool That Can Be Used For Scanning

-><https://github.com/shopify/kubeaudit/releases> (Go To The Web Site Copy The Linux\_amd\_64

Link)

- >Paste It Using wget Command
- >Now Untar The File Using tar -xvf File Name
- >sudo mv kubeaudit /usr/local/bin/
- >kubeaudit all

# Installing Jenkins on Ubuntu

```
#!/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
```

Save this script in a file, for example, `install_jenkins.sh`, and make it executable using:

```
chmod +x install_jenkins.sh
```

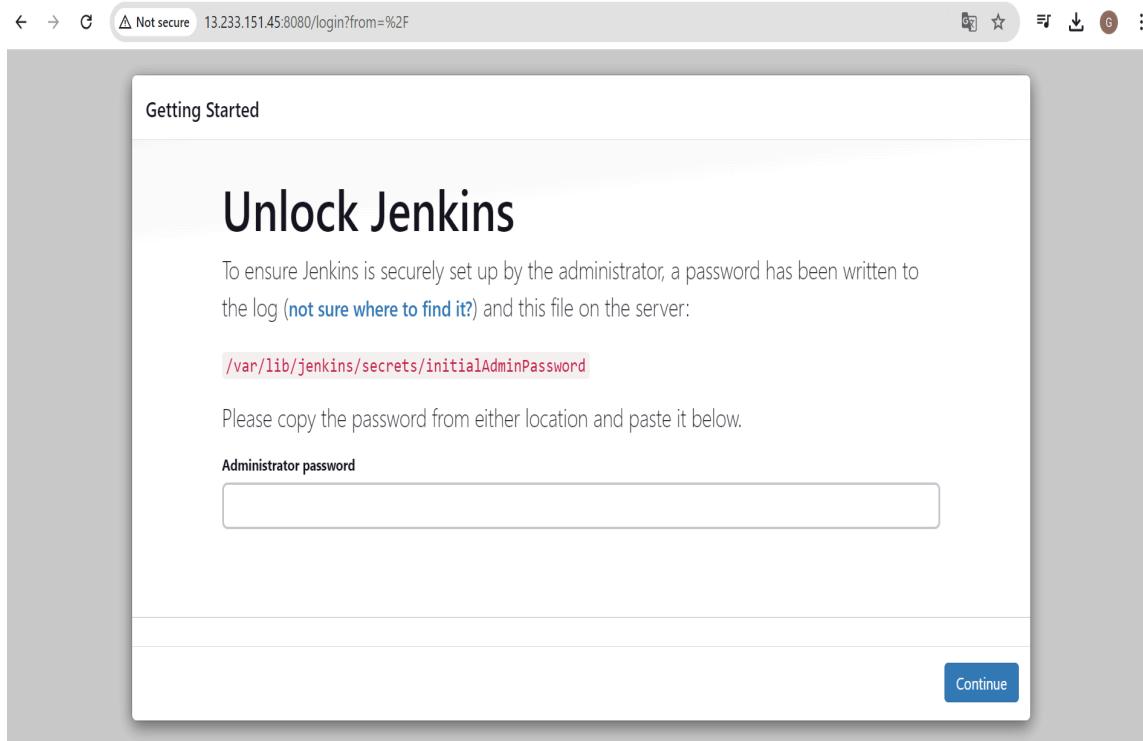
Then, you can run the script using:

```
./install_jenkins.sh
```

This script will automate the installation process of OpenJDK 17 JRE Headless and

Jenkins.

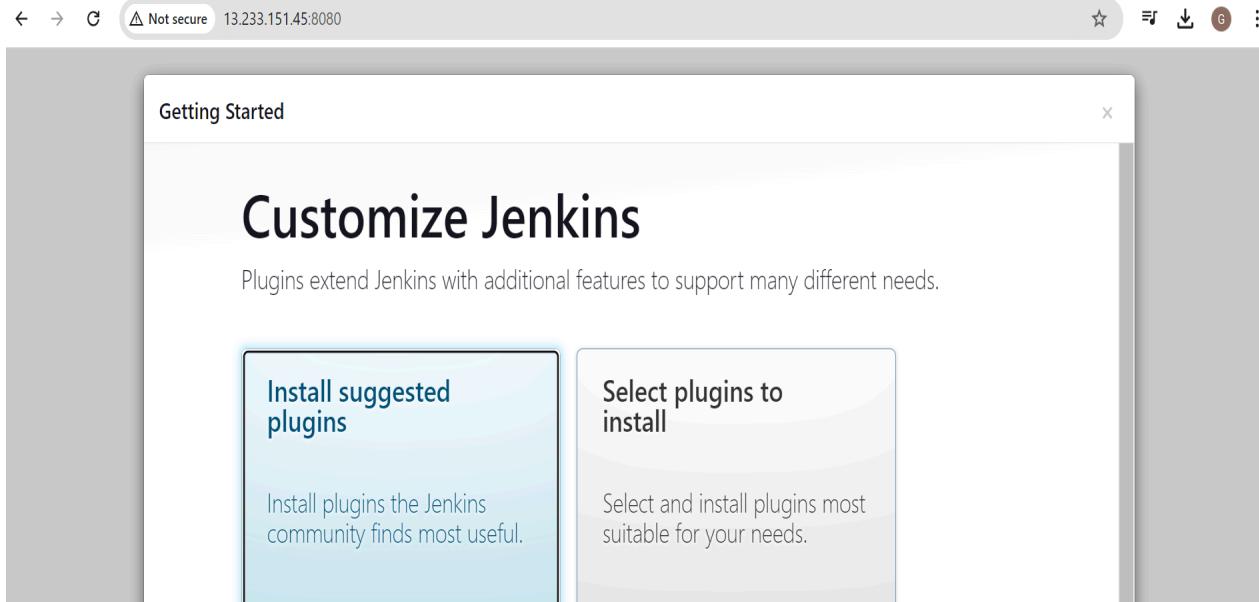
Now we can able to access Jenkins :using the public ip address <http://IP:8080>



Use the below command on your jenkins machine

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

Get password it will be like this —> a771e06575b54f91bc56a42ccdbb2f76



Not secure 13.233.151.45:8080

## Getting Started

Jenkins 2.462.1

Getting Started

Folders	Formatter		
Timestamper	Workspace Cleanup	Ant	Gradle
Pipeline	Github Branch Source	Pipeline: GitHub Groovy Libraries	Pipeline Graph View
Git	SSH Build Agents	Matrix Authorization Strategy	PAM Authentication
LDAP	Email Extension	Mailer	Dark Theme

Folders

- OWASP Markup Formatter
  - \*\* ASM API
  - \*\* JSON Path API
  - \*\* Structs
  - \*\* Pipeline: Step API
  - \*\* Token Macro

\*\* - required dependency

Not secure 13.233.151.45:8080

## Create First Admin User

Username  
ganesh

Password  
.....

Confirm password  
.....

Full name

Jenkins 2.462.1

Skip and continue as admin

Save and Continue

Not secure 13.233.151.45:8080

## Getting Started

# Instance Configuration

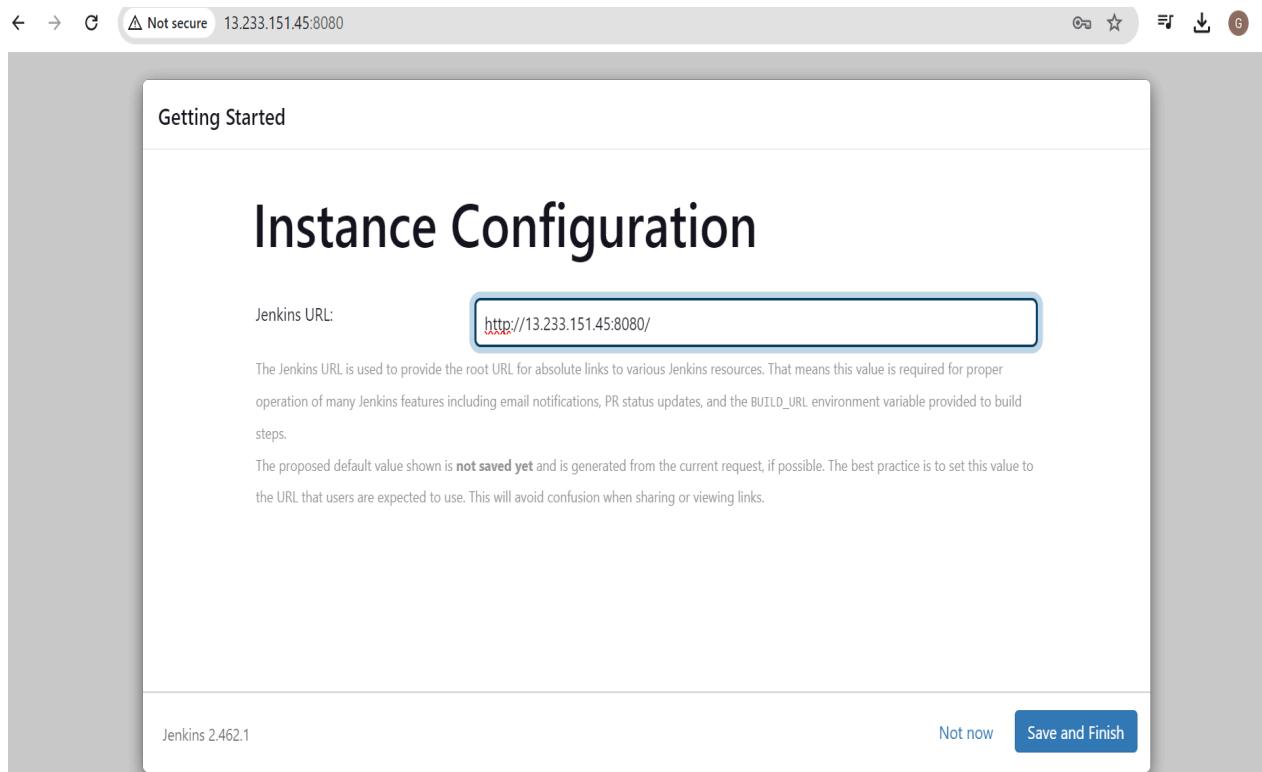
Jenkins URL: <http://13.233.151.45:8080/>

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

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

Jenkins 2.462.1

Not now [Save and Finish](#)



Not secure 13.233.151.45:8080

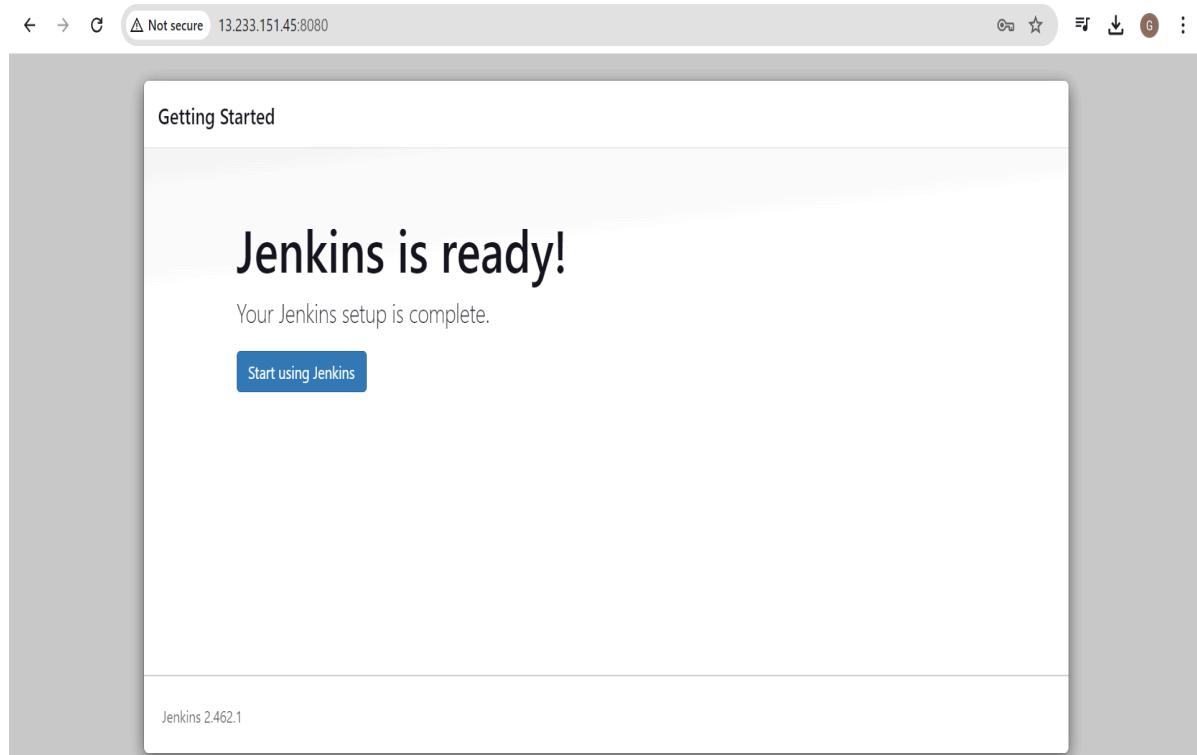
## Getting Started

# Jenkins is ready!

Your Jenkins setup is complete.

[Start using Jenkins](#)

Jenkins 2.462.1



The screenshot shows the Jenkins dashboard at the URL 13.233.151.45:8080. The top navigation bar includes a 'Not secure' warning, the IP address, and user information for 'Ganesh P'. The main content area features a 'Welcome to Jenkins!' message, a 'Start building your software project' button, and several management sections: 'Build Queue' (No builds in the queue), 'Build Executor Status' (1 Idle, 2 Idle), 'Create a job' (button with a plus sign), 'Set up a distributed build' (links to 'Set up an agent' and 'Configure a cloud').

## Install docker and trivy on Jenkins machine for future use

```
#!/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
```

Save this script in a file, for example, `install_docker.sh`, and make it executable using:  
chmod +x `install_docker.sh`

Then, you can run the script using:

```
./install_docker.sh
```

Trivy Installation Steps:

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

```
sudo apt-get install wget apt-transport-https gnupg lsb-release
```

```
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 -a /etc/apt/sources.list.d/trivy.list
```

```
sudo apt-get update
```

```
sudo apt-get install trivy -y
```

After creating the script change the permission to executable file

```
chmod +x trivy.sh
```

```
ls
```

```
./trivy.sh
```

```
trivy -version
```

## SetUp Nexus

```
#!/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
```

**Save this script in a file, for example, `install_docker.sh`, and make it executable using:**

```
chmod +x install_docker.sh
```

**Then, you can run the script using:**

```
./install_docker.sh
```

## Create Nexus using docker container

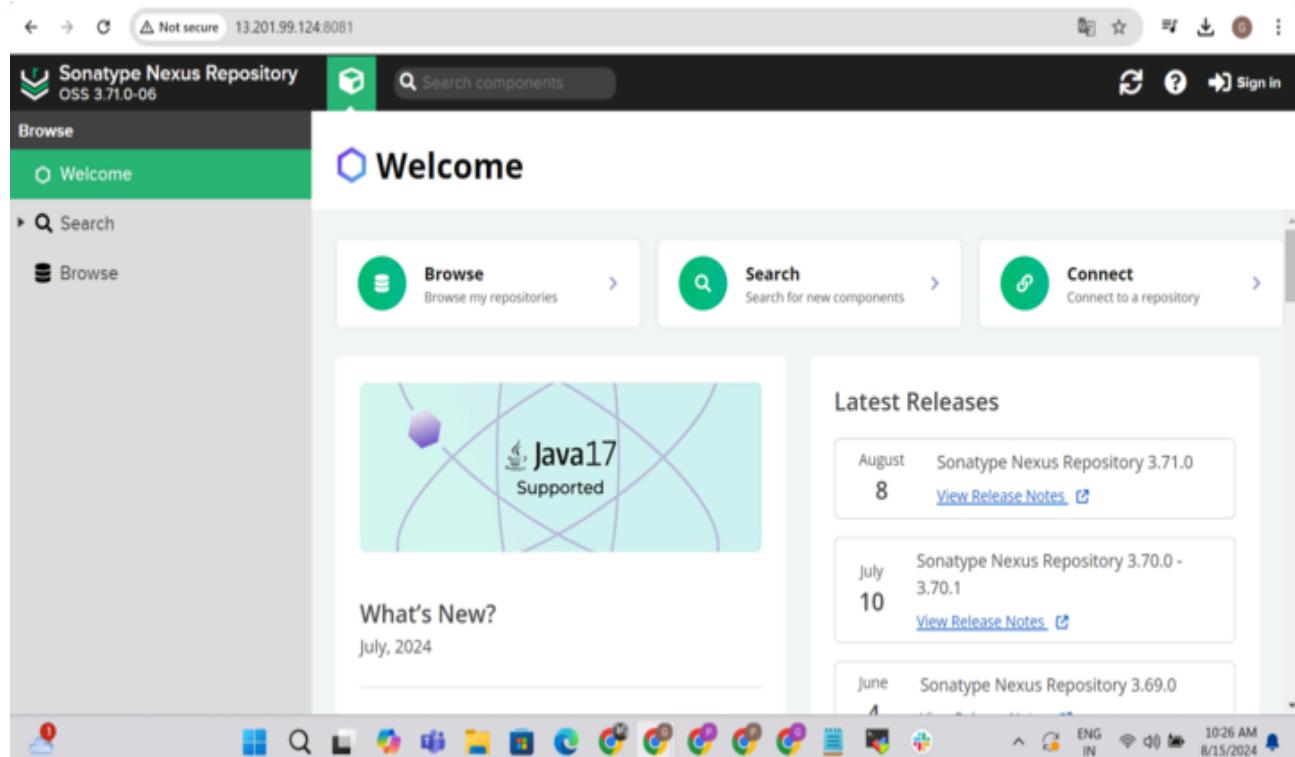
To create a Docker container running Nexus 3 and exposing it on port 8081, you can use the following command:

```
docker run -d --name nexus -p 8081:8081 sonatype/nexus3:latest
```

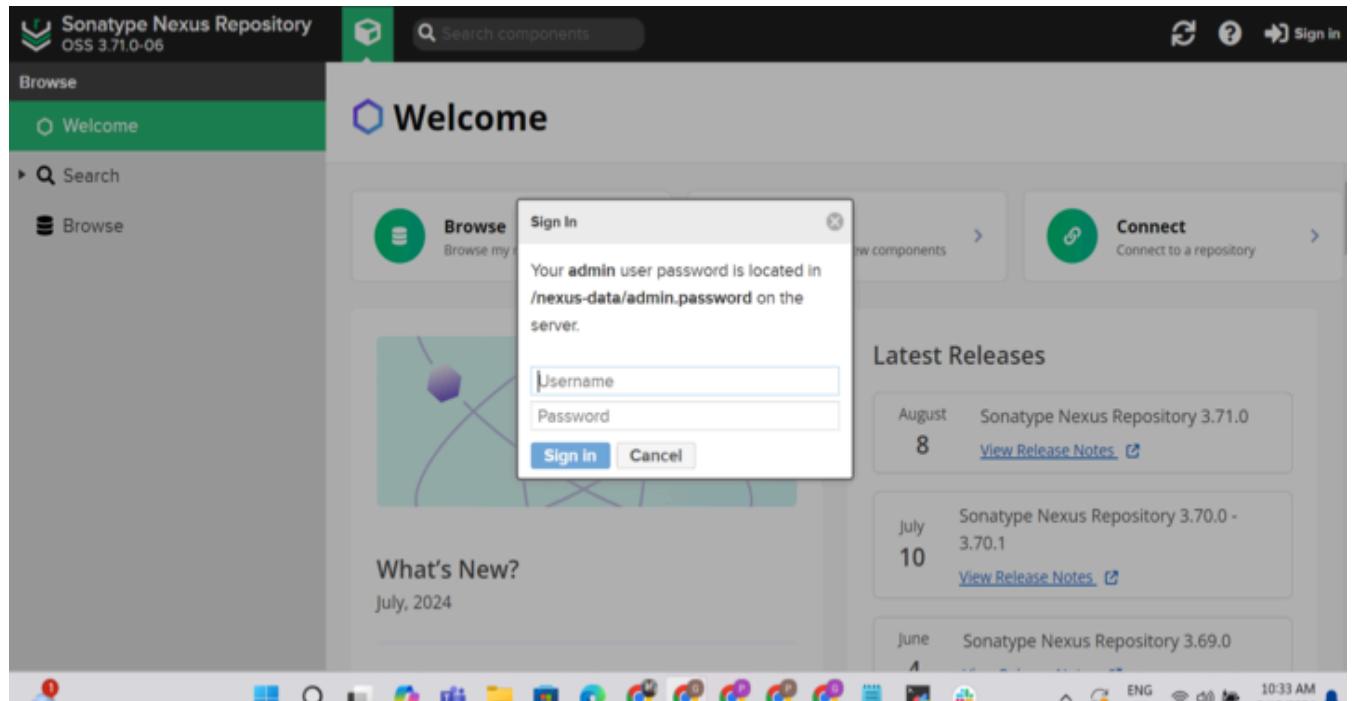
This command does the following:

- `-d`: Detaches the container and runs it in the background.
- `--name nexus`: Specifies the name of the container as "nexus".
- `-p 8081:8081`: Maps port 8081 on the host to port 8081 on the container, allowing access to Nexus through port 8081.
- `sonatype/nexus3:latest`: Specifies the Docker image to use for the container, in this case, the latest version of Nexus 3 from the Sonatype repository.

After running this command, Nexus will be accessible on your host machine at <http://IP:8081>.



## Get Nexus initial password



Your provided commands are correct for accessing the Nexus password stored in the container. Here's a breakdown of the steps:

1. **Get Container ID:** You need to find out the ID of the Nexus container. You can do this by running:

```
docker ps
```

This command lists all running containers along with their IDs, among other information.

2. **Access Container's Bash Shell:** Once you have the container ID, you can execute the `docker exec` command to access the container's bash shell:

```
docker exec -it <container_ID> /bin/bash
```

Replace `<container_ID>` with the actual ID of the Nexus container.

3. **Navigate to Nexus Directory:** Inside the container's bash shell, navigate to the directory where Nexus stores its configuration:

```
cd sonatype-work/nexus3
```

4. **View Admin Password:** Finally, you can view the admin password by displaying the contents of the `admin.password` file:

```
cat admin.password
```

5. **Exit the Container Shell:** Once you have retrieved the password, you can exit the container's bash shell:

```
exit
```

This process allows you to access the Nexus admin password stored within the container. Make sure to keep this password secure, as it grants administrative access to your Nexus instance.

Not secure 13.201.99.124:8081

Sonatype Nexus Repository OSS 3.71.0-06

Search components

admin Sign out

Welcome

Complete 4 of 4

The setup tasks have been completed, enjoy using Nexus Repository Manager!

Requests Per Day

20,000 Threshold

last 24 hours

Highest Recorded Count (30 days)

Finish

Sonatype will start to collect anonymous, non-sensitive usage metrics and performance information to shape the future of Nexus Repository. [Learn more](#) about the information we collect or [Decline](#).

OK

10:39 AM 8/15/2024

# SetUp SonarQube

```
#!/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
```

**Save this script in a file, for example, `install_docker.sh`, and make it executable using:**

```
chmod +x install_docker.sh
```

**Then, you can run the script using:**

```
./install_docker.sh
```

## Create Sonarqube Docker container

To run SonarQube in a Docker container with the provided command, you can follow these steps:

1. Open your terminal or command prompt.
2. Run the following command:

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

This command will download the `sonarqube:lts-community` Docker image from Docker Hub if it's not already available locally. Then, it will create a container named "sonar"

from this image, running it in detached mode (-d flag) and mapping port 9000 on the host machine to port 9000 in the container (-p 9000:9000 flag).

3. Access SonarQube by opening a web browser and navigating to <http://VmIP:9000>.

The image contains two screenshots of the SonarQube web interface. The top screenshot shows the 'Log in to SonarQube' page, which includes fields for 'Login' and 'Password', and a 'Log in' button. The bottom screenshot shows the 'Create Project' page, which displays options for importing projects from various platforms: 'From Azure DevOps', 'From Bitbucket Server', 'From Bitbucket Cloud', 'From GitHub', and 'From GitLab'. Each option has a corresponding icon and a 'Set up global configuration' link below it.

This will start the SonarQube server, and you should be able to access it using the provided URL. If you're running Docker on a remote server or a different port, replace `localhost` with the appropriate hostname or IP address and adjust the port accordingly.

# PHASE-2 | Private Git Setup

Steps to create a private Git repository, generate a personal access token, connect to the repository, and push code to it:

## 1. Create a Private Git Repository:

- Go to your preferred Git hosting platform (e.g., GitHub, GitLab, Bitbucket).
- Log in to your account or sign up if you don't have one.
- Create a new repository and set it as private.

## 2. Generate a Personal Access Token:

- Navigate to your account settings or profile settings.
- Look for the "Developer settings" or "Personal access tokens" section.
- Generate a new token, providing it with the necessary permissions (e.g., repo access).

## 3. Clone the Repository Locally:

- Open Git Bash or your terminal.
- Navigate to the directory where you want to clone the repository.
- Use the `git clone` command followed by the repository's URL. For example:

```
git clone <repository_URL>
[+] Tops_Ganesh@LP-CHN-GP MINGW64 ~
$ git clone https://github.com/ganeshperumal007/Board-game.git
Cloning into 'Board-game'...
```

- 4. Replace `<repository_URL>` with the URL of your private repository.

## 5. Add Your Source Code Files:

- Navigate into the cloned repository directory.
- Paste your source code files or create new ones inside this directory.

## 6. Stage and Commit Changes:

- Use the `git add` command to stage the changes:

- `git add .`
- Use the `git commit` command to commit the staged changes along with a meaningful message:

```
git commit -m "Your commit message here"
```

## 7. Push Changes to the Repository:

- Use the `git push` command to push your committed changes to the remote repository  
`git push`
- If it's your first time pushing to this repository, you might need to specify the remote and branch:

```
git push -u origin master
```

## 8. Replace `master` with the branch name if you're pushing to a different branch.

## 9. Enter Personal Access Token as Authentication:

- When prompted for credentials during the push, enter your username (usually your email) and use your personal access token as the password.

By following these steps, you'll be able to create a private Git repository, connect to it using Git Bash, and push your code changes securely using a personal access token for authentication.

# PHASE-3 | CICD

Install below Plugins in Jenkins

The screenshot shows the Jenkins Plugin Manager interface. The left sidebar has tabs for 'Updates', 'Available plugins' (which is selected), 'Installed plugins', 'Advanced settings', and 'Download progress'. The main area is titled 'Plugins' and contains a search bar. Below the search bar is a table listing various Jenkins plugins:

Plugin	Description	Last Updated
Eclipse Temurin installer 1.5	Provides an installer for the JDK tool that downloads the JDK from https://adoptium.net	1 yr 10 mo ago
Config File Provider 973.vb_a_80ecb_9a_4d0	Ability to provide configuration files (e.g. settings.xml for maven, XML, groovy, custom files,...) loaded through the UI which will be copied to the job workspace.	3 mo 28 days ago
Pipeline Maven Integration 1421.v610fa_b_e2d60e	This plugin provides integration with Pipeline, configures maven environment to use within a pipeline job by calling sh mvn or bat mvn. The selected maven installation will be configured and prepended to the path.	2 mo 9 days ago
SonarQube Scanner 2.17.2	This plugin allows an easy integration of SonarQube, the open source platform for Continuous Inspection of code quality.	5 mo 27 days ago
Docker 1.6.2	This plugin integrates Jenkins with Docker	2 mo 12 days ago
Docker Pipeline 580.vc0c340686b_54	Build and use Docker containers from pipelines.	2 mo 25 days ago
Kubernetes 4285.v50ed5f1624918	This plugin integrates Jenkins with Kubernetes	2 days 17 hr ago
Kubernetes CLI 1.12.1	Configure kubectl for Kubernetes	11 mo ago

## 1. Eclipse Temurin Installer:

- This plugin enables Jenkins to automatically install and configure the Eclipse Temurin JDK (formerly known as AdoptOpenJDK).
- To install, go to Jenkins dashboard → Manage Jenkins → Manage Plugins → Available tab.
- Search for "Eclipse Temurin Installer" and select it.
- Click on the "Install without restart" button.

## 2. Pipeline Maven Integration:

- This plugin provides Maven support for Jenkins Pipeline.
- It allows you to use Maven commands directly within your Jenkins Pipeline scripts.
- To install, follow the same steps as above, but search for "Pipeline Maven Integration" instead.

## 3. Config File Provider:

- This plugin allows you to define configuration files (e.g., properties, XML,

- JSON centrally in Jenkins.
- These configurations can then be referenced and used by your Jenkins jobs.
- Install it using the same procedure as mentioned earlier.

#### 4. SonarQube Scanner:

- SonarQube is a code quality and security analysis tool.
- This plugin integrates Jenkins with SonarQube by providing a scanner that analyzes code during builds.
- You can install it from the Jenkins plugin manager as described above.

#### 5. Kubernetes CLI:

- This plugin allows Jenkins to interact with Kubernetes clusters using the Kubernetes command-line tool (`kubectl`).
- It's useful for tasks like deploying applications to Kubernetes from Jenkins jobs.
- Install it through the plugin manager.

#### 6. Kubernetes:

- This plugin integrates Jenkins with Kubernetes by allowing Jenkins agents to run as pods within a Kubernetes cluster.
- It provides dynamic scaling and resource optimization capabilities for Jenkins builds.
- Install it from the Jenkins plugin manager.

#### 7. Docker:

- This plugin allows Jenkins to interact with Docker, enabling Docker builds and integration with Docker registries.
- You can use it to build Docker images, run Docker containers, and push/pull images from Docker registries.
- Install it from the plugin manager.

#### 8. Docker Pipeline Step:

- This plugin extends Jenkins Pipeline with steps to build, publish, and run Docker containers as part of your Pipeline scripts.
- It provides a convenient way to manage Docker containers directly from Jenkins Pipelines.
- Install it through the plugin manager like the others.

After installing these plugins, you may need to configure them according to your specific environment and requirements. This typically involves setting up credentials, configuring paths, and specifying options in Jenkins global configuration or individual job configurations. Each plugin usually comes with its own set of documentation to guide you through the configuration process.

## Configure Above Plugins in Jenkins Pipeline

Configure the tools choose manage jenkins → Tools

Choose jdk and fill as given below

JDK installations

Add JDK

≡ JDK

Name

jdk17

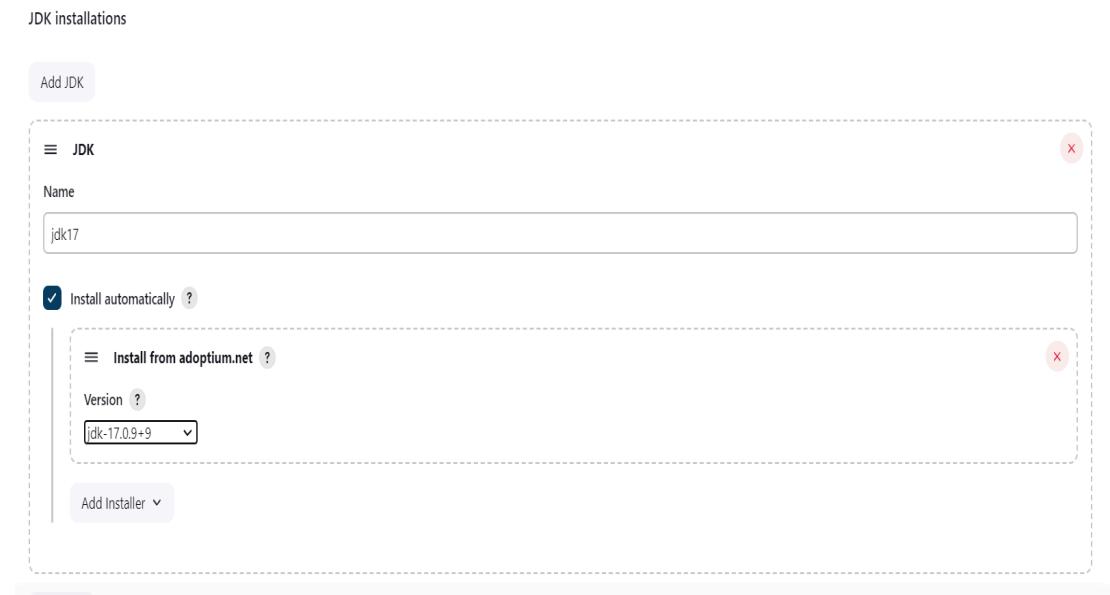
Install automatically ?

≡ Install from adoptium.net ?

Version ?

jdk-17.0.9+9

Add Installer ▾



choose sonarqube scanner and configure

SonarQube Scanner installations

Add SonarQube Scanner

≡ SonarQube Scanner

Name

sonar-scanner

Install automatically ?

≡ Install from Maven Central

Version

SonarQube Scanner 6.1.0.4477

Add Installer ▾



## choose maven and Configure

Maven installations

Add Maven

**Maven**

Name  
maven3

Install automatically ?

**Install from Apache**

Version  
3.6.1

Add Installer ▾



## choose Docker and Configure

Docker installations

Add Docker

**Docker**

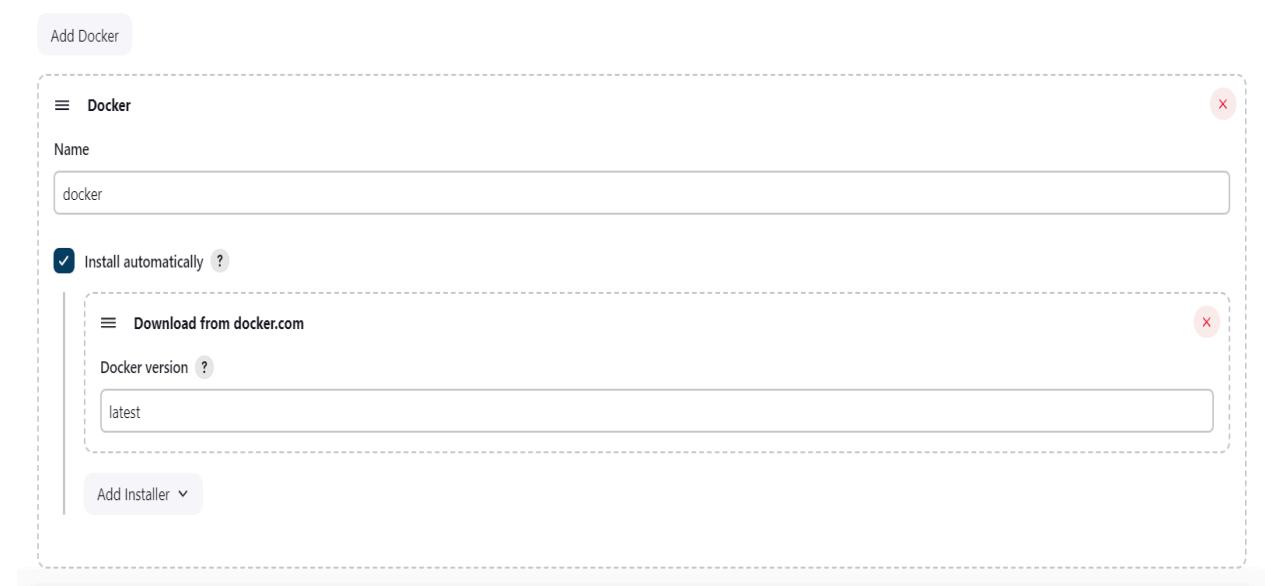
Name  
docker

Install automatically ?

**Download from docker.com**

Docker version ?  
latest

Add Installer ▾



## Create the new credentials for sonarqube

Dashboard > Manage Jenkins > Credentials > System > Global credentials (unrestricted) >

### New credentials

Kind

Secret text

Scope ?  
Global (Jenkins, nodes, items, all child items, etc)

Secret

ID ?

Description ?

**Create**

Get the token from sonarcube administration→ security→generate token  
(squ\_db0be0fe8e1896ad726617530d37cf233a82993d)

sonarqube Projects Issues Rules Quality Profiles Quality Gates Administration

Administration

### Tokens of Administrator

Generate Tokens

Name	Expires in
Enter Token Name	30 days

**!** New token "sonar-token" has been created. Make sure you copy it now, you won't be able to see it again!

**Copy** squ\_db0be0fe8e1896ad726617530d37cf233a82993d

Name	Type	Project	Last use	Created	Expiration
sonar-token	User		Never	August 15, 2024	September 14, 2024

SonarQube™ technology is powered by SonarSource SA  
Community Edition - v9.9.6 (build 92038) - [LGPL v3](#) - [Community](#) - [Documentation](#) - [Plugins](#) - [Web API](#)

91°

Now need to configure in the sonarqube server  
Go to manage jenkins >system

SonarQube installations

List of SonarQube installations

Name	X
sonar	
Server URL	
Default is http://localhost:9000	
http://43.204.97.53:9000/	
Server authentication token	
SonarQube authentication token. Mandatory when anonymous access is disabled.	
sonar-token	v
+ Add ▾	
Advanced ▾	

```
pipeline {
    agent any

    tools {
        jdk 'jdk17'
        maven 'maven3'
    }

    environment {
        SCANNER_HOME= tool 'sonar-scanner'
    }

    stages {
        stage('Git Checkout') {
            steps {
                git branch: 'main', credentialsId: 'git-cred', url: 'https://github.com/ganeshperumal007/Boardgame.git'
            }
        }

        stage('Compile') {
            steps {
                sh "mvn compile"
            }
        }

        stage('Test') {
            steps {
                sh "mvn test"
            }
        }
    }
}
```

```

}

stage('File System Scan'){
    steps {
        sh "trivy fs --format table -o trivy-fs-report.html ."
    }
}

stage('SonarQube Analysys'){
    steps {
        withSonarQubeEnv('sonar'){
            sh ''' $SCANNER_HOME/bin/sonar-scanner -
Dsonar.projectName=BoardGame -Dsonar.projectKey=BoardGame \
-Dsonar.java.binaries=. '''
        }
    }
}

stage('Quality Gate'){
    steps {
        script {
            waitForQualityGate abortPipeline: false, credentialsId: 'sonar-token'
        }
    }
}

stage('Build'){
    steps {
        sh "mvn package"
    }
}

stage('Publish To Nexus'){
    steps {
        withMaven(globalMavenSettingsConfig: 'global-settings', jdk: 'jdk17',
maven: 'maven3', mavenSettingsConfig: '', traceability: true){
            sh "mvn deploy"
        }
    }
}

stage('Build & Tag Docker Image'){
    steps {
        script {
            withDockerRegistry(credentialsId: 'docker-cred', toolName: 'docker'){
                sh "docker build -t ganeshperumal007/boardshack:latest ."
            }
        }
    }
}

stage('Docker Image Scan'){
    steps {
        sh "trivy image --format table -o trivy-image-report.html
ganeshperumal007/boardshack:latest "
    }
}

```

```

        }
    }

stage('Push Docker Image'){
    steps {
        script {
            withDockerRegistry(credentialsId: 'docker-cred', toolName: 'docker') {
                sh "docker push ganeshperumal007/boardshack:latest"
            }
        }
    }
}

stage('Deploy To Kubernetes'){
    steps {
        withKubeConfig(caCertificate: ' ', clusterName: 'kubernetes', contextName: '',
        credentialsId: 'k8-cred', namespace: 'webapps', restrictKubeConfigAccess: false,
        serverUrl: 'https://172.31.8.146:6443'){
            sh "kubectl apply -f deployment-service.yaml"
        }
    }
}

stage('Verify the Deployment'){
    steps {
        withKubeConfig(caCertificate: ' ', clusterName: 'kubernetes', contextName: '',
        credentialsId: 'k8-cred', namespace: 'webapps', restrictKubeConfigAccess: false,
        serverUrl: 'https://172.31.8.146:6443'){
            sh "kubectl get pods -n webapps"
            sh "kubectl get svc -n webapps"
        }
    }
}

post {
    always {
        script {
            def jobName = env.JOB_NAME
            def buildNumber = env.BUILD_NUMBER
            def pipelineStatus = currentBuild.result ?: 'UNKNOWN'
            def bannerColor = pipelineStatus.toUpperCase() == 'SUCCESS' ? 'green' : 'red'

            def body = """
                <html>
                <body>
                    <div style="border: 4px solid ${bannerColor}; padding:
10px;">
                        <h2>${jobName} - Build
${buildNumber}</h2>
                        <div style="background-color:
${bannerColor}; padding:
10px;">
                            <h3 style="color: white;">Pipeline
Status:</h3>
                            ${pipelineStatus.toUpperCase()}</h3>
                        </div>
                    <p>Check the <a href="${BUILD_URL}">console

```

```

        output</a>.</p> </div>
        </body>
        </html>

        """

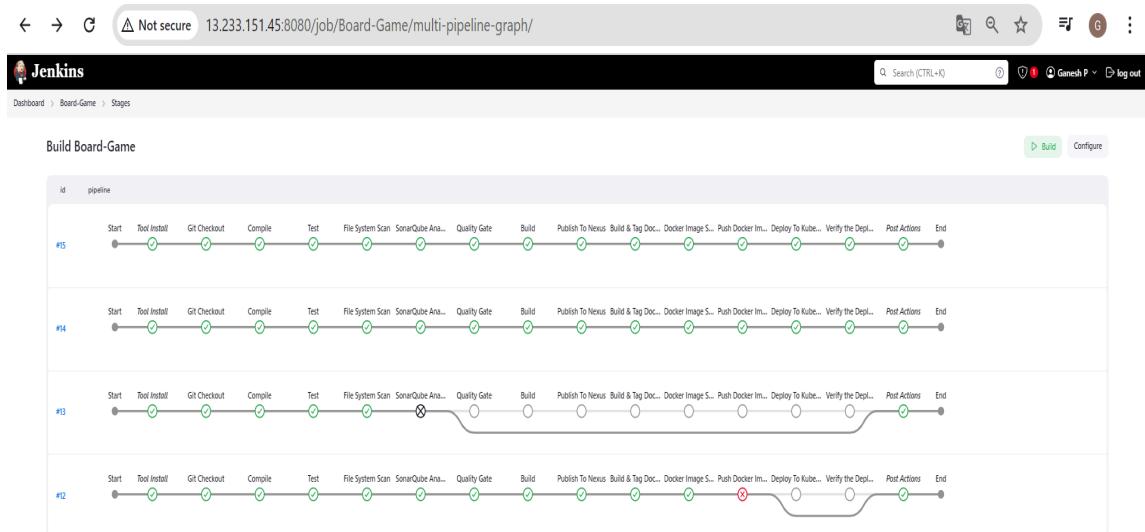
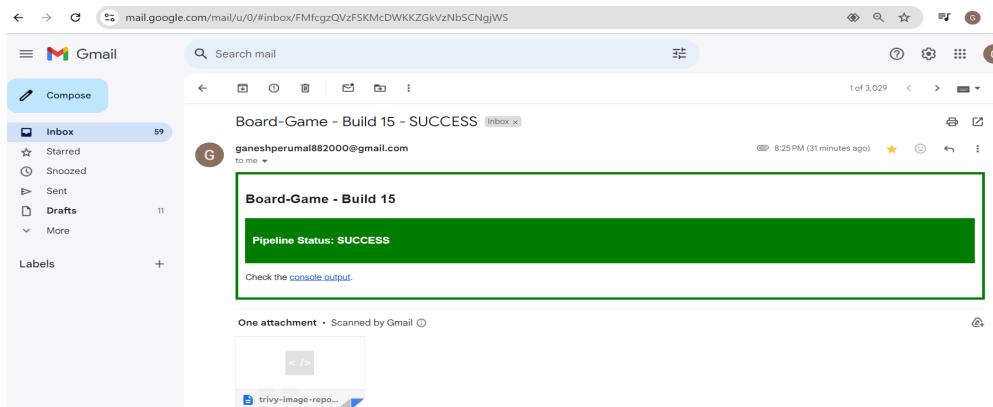
emailext (
    subject: "${jobName} - Build ${buildNumber} - 
${pipelineStatus.toUpperCase()}",
    body: body,
    to:
        'ganeshperumal882000@gmail.co
m', from: 'jenkins@example.com',
    replyTo: 'jenkins@example.com',
    mimeType: 'text/html',
    attachmentsPattern: 'trivy-image-report.html'
)
}
}

}

}

```

## OUTPUT



```
Dashboard > Board-Game > #17
[Pipeline] withKubeConfig
[Pipeline] {
[Pipeline] sh
+ kubectl get pods -n webapps
NAME                   READY   STATUS    RESTARTS   AGE
boardgame-deployment-54bcc9875c-7mh4c  1/1     Running   0          11h
boardgame-deployment-54bcc9875c-zrlml  1/1     Running   0          11h
[Pipeline] sh
+ kubectl get svc -n webapps
NAME           TYPE      CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
boardgame-svc  LoadBalancer  10.108.214.70  <pending>  80:31598/TCP  12h
[Pipeline]
[Pipeline] kubectl configuration cleaned up
[Pipeline] // withKubeConfig
[Pipeline] {
[Pipeline] // withEnv
[Pipeline] {
[Pipeline] // stage
[Pipeline] stage
[Pipeline] {
  [Pipeline] { Declarative: Post Actions
[Pipeline] script
[Pipeline] {
[Pipeline] {
  [Pipeline] emulet
  Sending email to: ganesperumal882000@gmail.com
[Pipeline]
[Pipeline] // script
[Pipeline] {
[Pipeline] // stage
[Pipeline]
[Pipeline] // withEnv
[Pipeline] {
[Pipeline] // withEnv
[Pipeline] {
[Pipeline] // node
[Pipeline] End of Pipeline
Finished: SUCCESS
```

The screenshot shows the SonarQube dashboard for the 'BoardGame' project. At the top, there's a navigation bar with links for 'Projects', 'Issues', 'Rules', 'Quality Profiles', 'Quality Gates', and 'Administration'. A search bar is also present. The main content area includes a 'QUALITY GATE STATUS' box indicating 'Passed' with a green background and 'All conditions passed.' message. Below this are several 'MEASURES' sections: 'New Code' (since August 15, 2024, started 23 hours ago), 'New Bugs' (0), 'Reliability' (A), 'New Vulnerabilities' (0), 'Security' (A), 'New Security Hotspots' (0), 'Reviewed' (status bar), 'Security Review' (A), 'Added Debt' (0), 'New Code Smells' (0), and 'Maintainability' (A). The bottom of the screen shows a taskbar with various icons and system status indicators.

The screenshot shows the Sonatype Nexus Repository interface. The top navigation bar includes links for 'Browse', 'Search', 'Upload', and 'Sign out'. The main content area is titled 'Browse / maven-releases'. It displays a tree view of a Maven project structure under 'javaproject':

- javaproject/  
  database\_service\_project/  
    0.0.4/  
      database\_service\_project-0.0.4.jar  
      database\_service\_project-0.0.4.jar.md5  
      database\_service\_project-0.0.4.jar.sha1  
      database\_service\_project-0.0.4.pom  
      database\_service\_project-0.0.4.pom.md5  
      database\_service\_project-0.0.4.pom.sha1  
      maven-metadata.xml  
      maven-metadata.xml.md5  
      maven-metadata.xml.sha1

Now we can able to access the see our application is running in slave machine 1&2

Check the ip address of both slave 1 & 2 <http://yourip:31508/>

The screenshot shows a web application titled 'Boardgame Lists'. The header includes a welcome message 'Welcome to BoardGame Database!', a navigation bar with 'Home', 'Login', and 'Sign-up' links, and a status bar indicating 'Not secure' and the URL '13.201.188.1:31508'.

The main content area displays three boardgame titles in boxes:

- Splendor
- Clue
- Linkee

Below the boxes, there is a message: 'For more services, login [Here](#)' and 'To join to the service [Click here](#)'.

Welcome to BoardGame Database! 🎲

Home Login Sign-up

## Boardgame Lists

Splendor

Clue

Linkee

For more services, login [Here](#)

To login to the service [Click here](#)

If You face issue while rebuilding the artifacts is not uploaded , Allow re-deploy in the settings

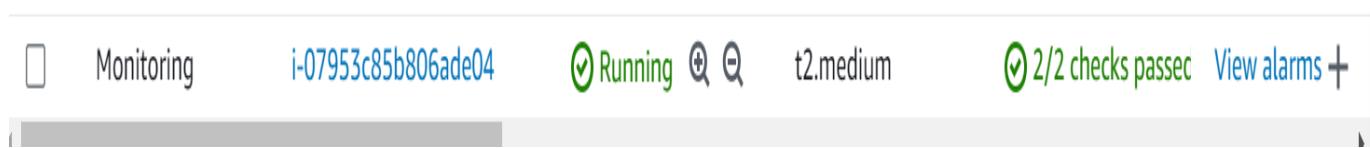
choose settings → administration → repositories → maven-releases

there you can enable the re-deploy option as shown below in the picture

The screenshot shows the Sonatype Nexus Repository interface. The top navigation bar includes the Sonatype Nexus Repository logo, OSS 3.71.0-06, a search bar, and user account information for 'admin'. The left sidebar menu is titled 'Administration' and lists 'Repository' (selected), 'Blob Stores', 'Proprietary Repositories', 'Content Selectors', 'Cleanup Policies', 'Routing Rules', 'Security', 'Privileges', 'Roles', and 'Users'. The main content area is titled 'Repositories / maven-releases'. It features a 'Delete repository' button and a 'Rebuild index' button. Below these are sections for 'Settings', 'Blob store:' (set to 'default'), 'Strict Content Type Validation:' (unchecked), 'Hosted', 'Deployment policy:' (set to 'Allow redeploy'), 'Proprietary Components:' (unchecked), and 'Cleanup'.

# PHASE-4 | Monitoring

Launch an EC2 instances with the t2.medium and start setup installation prometheus and Grafana



Before Starting Installation Update the package

```
sudo apt update
```

1.Links to download Prometheus, Node\_Exporter & black Box exporter <https://prometheus.io/download/>

## Download the latest prometheus

Copy the link from the above official document website and download in your local machine

```
wget  
https://github.com/prometheus/prometheus/releases/download/v2.54.0/prometheus-2.54.0.linux-amd64.tar.gz
```

## Extract the downloaded archive

```
tar -xvf prometheus-2.54.0.linux-amd64.tar.gz
```

```
cd prometheus-2.54.0.linux-amd64/
```

Now give ls we can able to see the executable file in the name of prometheus ,To start prometheus run the executable script in the name prometheus

```
./prometheus.sh &
```

Now we can able to access the prometheus ,[http://<your\\_server\\_IP>:9090/](http://<your_server_IP>:9090/).

The screenshot shows the Prometheus web interface. At the top, there's a navigation bar with icons for back, forward, and search, followed by a status message "Not secure" and the URL "3.110.151.216:9090/graph?g0.expr=&g0.tab=1&g0.display\_mode=lines&g0.show\_exemplars=0&g0.range\_input=1h". Below the header is a dark navigation bar with the Prometheus logo and links for Alerts, Graph, Status, and Help. Underneath are several configuration checkboxes: "Use local time" (unchecked), "Enable query history" (unchecked), "Enable autocomplete" (checked), "Enable highlighting" (checked), and "Enable linter" (checked). A search bar with a magnifying glass icon contains the placeholder text "Expression (press Shift+Enter for newlines)". Below the search bar is a tab switcher between "Table" and "Graph", currently set to "Graph". Underneath the tabs are two large buttons with arrows pointing left and right, labeled "Evaluation time". A message "No data queried yet" is displayed. At the bottom left is a blue button labeled "Add Panel".



## 2. Links to download

Grafana <https://grafana.com/grafana/download>

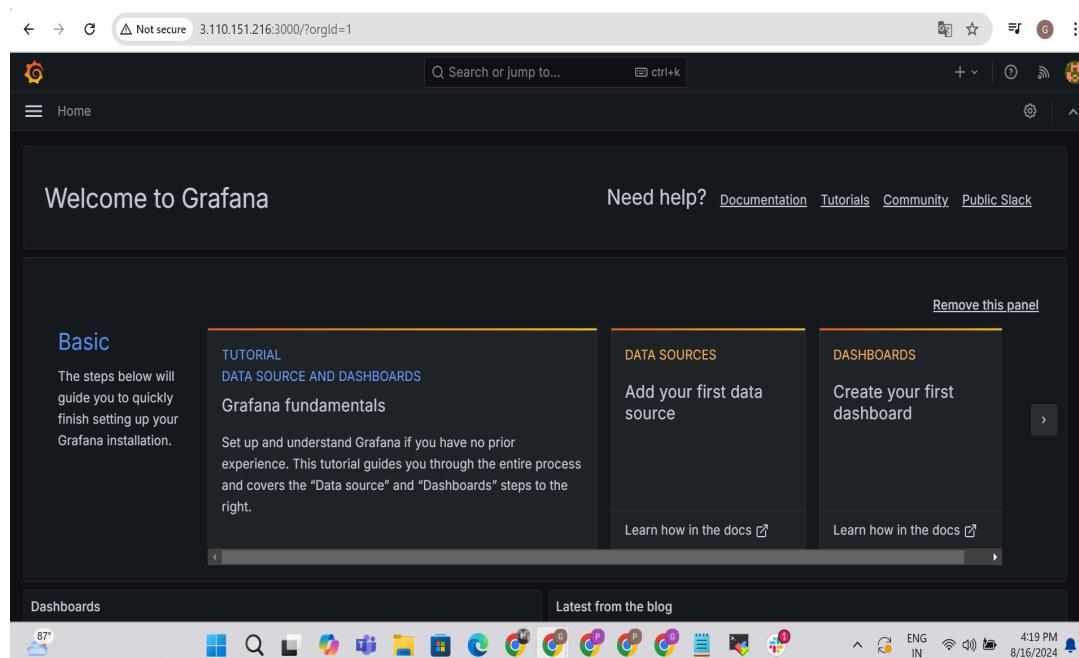
Refer the official documentation and install the latest version

```
sudo apt-get install -y adduser libfontconfig1 musl  
wget https://dl.grafana.com/enterprise/release/grafana-enterprise_11.1.4_amd64.deb  
sudo dpkg -i grafana-enterprise_11.1.4_amd64.deb
```

### You can start grafana-server by executing

```
sudo /bin/systemctl start grafana-server
```

Grafana is running now, and we can connect to it at <http://your.server.ip:3000>. The default user & password is admin / admin.



### [3.https://github.com/prometheus/blackbox\\_exporter](https://github.com/prometheus/blackbox_exporter)

**Download blackbox exporter from the official website of prometheus,**

`wget`

`https://github.com/prometheus/blackbox_exporter/releases/download/v0.25.0/blackbox_exporter-0.25.0.linux-amd64.tar.gz`

**Now Extract the tar file of blackbox exporter**

```
tar -xvf blackbox_exporter-0.25.0.linux-amd64.tar.gz
```

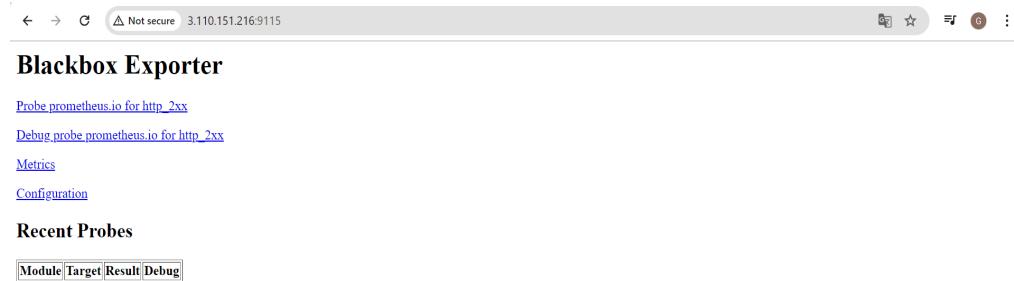
**To start the black box exporter**

```
cd blackbox_exporter-0.25.0.linux-amd64/
```

Now give ls we can able to see the executable file in the name of blackbox ,To start blackbox run the executable script in the name blackbox

```
./blackbox_exporter &
```

Now we can able to access the prometheus ,[http://<your\\_server\\_IP>:9115/](http://<your_server_IP>:9115/).



## Prometheus yaml Configuration

```
cd prometheus-2.54.0.linux-amd64/
ls
vi prometheus.yml

# Load rules once and periodically evaluate them according to the global
# 'evaluation_interval'.

rule_files:
  # - "first_rules.yml"
  # - "second_rules.yml"

# A scrape configuration containing exactly one endpoint to scrape:
# Here it's Prometheus itself.

scrape_configs:
  # The job name is added as a label `job=<job_name>` to any timeseries scraped from this
  config.
  - job_name: "prometheus"

    - job_name: "prometheus"
```

```

# metrics_path defaults to '/metrics'
# scheme defaults to 'http'.

static_configs:
  - targets: ["localhost:9090"]

  - job_name: 'blackbox'
    metrics_path: /probe
    params:
      module: [http_2xx] # Look for a HTTP 200 response.

static_configs:
  - targets:
    - http://prometheus.io      # Target to probe with http
    - http://13.201.188.1:31508/ # Target to probe with http on port 31508 our application.

relabel_configs:
  - source_labels: [__address__]
    target_label: __param_target
  - source_labels: [__param_target]
    target_label: instance
  - target_label: __address__
    replacement: 3.110.151.216:9115 # The blackbox exporter's real hostname:127.0.0.1:

```

## Now need to restart the prometheus

`pgrep prometheus`

`it will shows the id`  
`kill the id`  
`kill 1992`

`Then restart`  
`./prometheus &`

← → G △ Not secure 3.110.151.216:9090/targets?search=

Prometheus Alerts Graph Status Help

## Targets

All scrape pools ▾ All Unhealthy Collapse All Filter by endpoint or labels  Unknown  Unhealthy  Healthy

**blackbox (2/2 up)** show less

Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
http://3.110.151.216:9115/probe module="http_2xx" target="http://prometheus.io"	UP	instance="http://prometheus.io" job="blackbox"	-3.000ms ago	169.086ms	
http://3.110.151.216:9115/probe module="http_2xx" target="http://13.201.188.1:31508/"	UP	instance="http://13.201.188.1:31508/" job="blackbox"	11.570s ago	9.513s	

**prometheus (1/1 up)** show less

Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
http://localhost:9090/metrics	UP	instance="localhost:9090" job="prometheus"	731.000ms ago	6.037ms	



NOW We Need to add prometheus as our datasource in Grafana

Then Import dashboard give ID 7587 to create black box exporter dashboard



let us monitor the jenkins machine metrics by node exporter  
install prometheus metrics plugins in jenkins

```
let us download the node exporter in jenkins machine
```

```
wget
```

```
https://github.com/prometheus/node\_exporter/releases/download/v1.8.2/node\_exporter-1.8.2.linux-amd64.tar.gz
```

Now extract the file

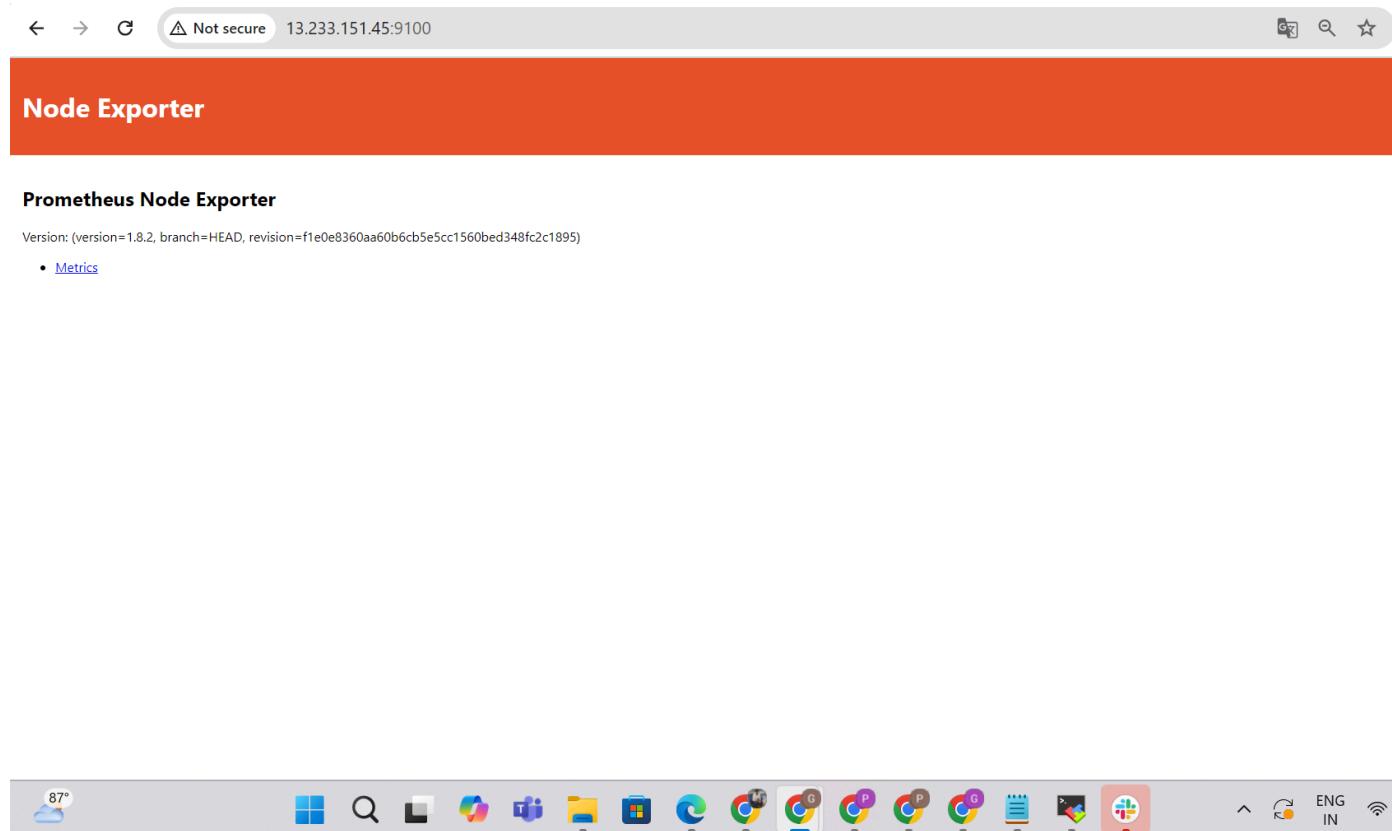
```
tar -xvf node_exporter-1.8.2.linux-amd64.tar.gz
```

Now run the nodeexporter script

```
cd node_exporter-1.8.2.linux-amd64/  
ls  
.node_exporter &
```

Now nodeExporter can be accessible through the browser

```
http://<your\_server\_IP>:9100
```



**Make sure the below one was configured in manage jenkins→system**

The screenshot shows the Jenkins configuration interface for the Prometheus plugin. It includes fields for Path (prometheus), Default Namespace (default), and a checkbox for Enable authentication for prometheus end-point. A dropdown for Collecting metrics period in seconds is set to 120. A list of build-related metrics is shown with checkboxes, all of which are checked: Count duration of successful builds, Count duration of unstable builds, Count duration of failed builds, Count duration of not-built builds, Count duration of aborted builds, Fetch the test results of builds, Add build parameter label to metrics, Add build status label to metrics, and Process disabled jobs. At the bottom are Save and Apply buttons.

**now need to edit the yaml file of prometheus**

**vi prometheus.yaml**

```
#update the yaml file

- job_name: 'node_exporter'

  static_configs:
    - targets: ['13.233.151.45:9100']

- job_name: 'Jenkins'

  metrics_path: '/prometheus'

  static_configs:
    - targets: ['13.233.151.45:8080']
```

After update the yaml file make sure restart the prometheus

```
pgrep prometheus
kill id
./prometheus &
```

**Now in Grafana Create another dashboard for node exporter  
import dashboard id - 1860**

```
{
  "title": "Example - Repeating Dictionary variables",
  "uid": "_OHnEoN4z",
  "panels": [...]
}
```

