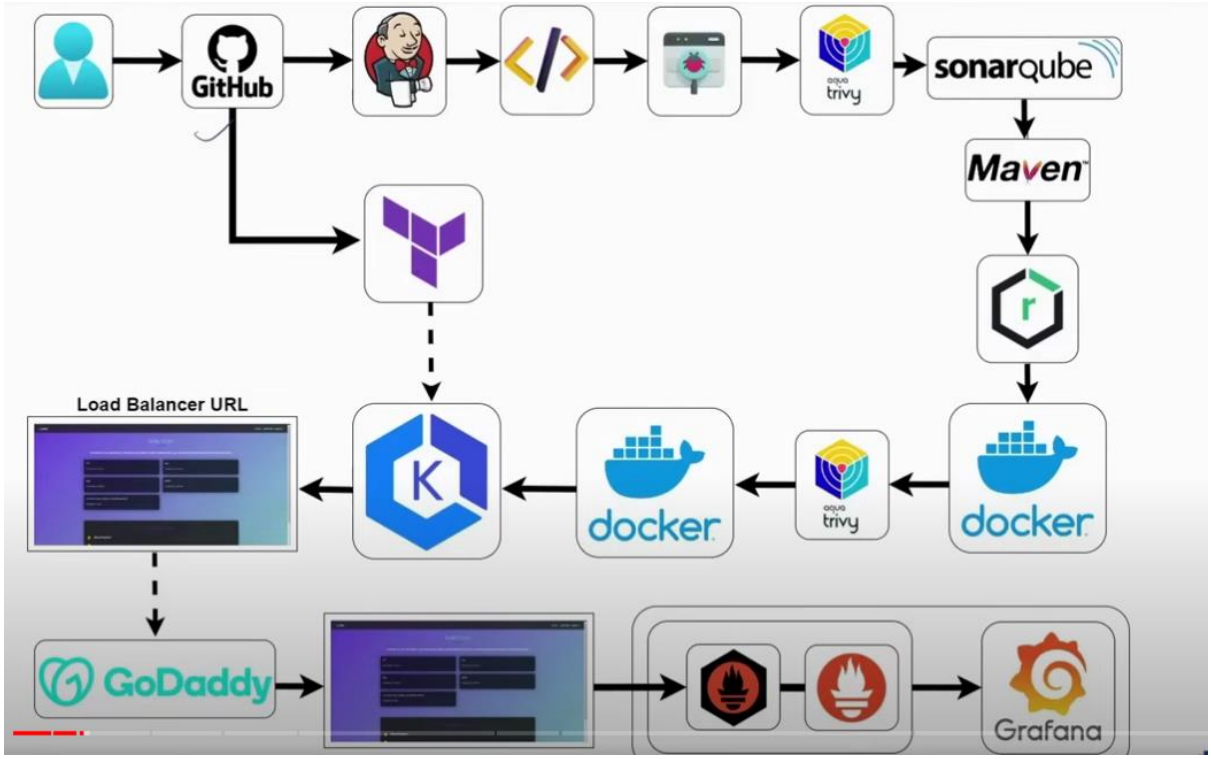
Production Level CICD Pipeline Project | CICD DevOps Project



Step 1: Launching an EC2 Instance on AWS

1. **Log in to AWS Console:** Go to the AWS Management Console and sign in.
2. **Launch an EC2 Instance:**
   - Navigate to **EC2 Dashboard** and click **Launch Instance**.
   - Choose an **Amazon Machine Image (AMI)**: Select **Ubuntu 22.04** (or the latest stable version).
   - Choose an **Instance Type**: Select a type like **t2.micro** for testing or small workloads (free tier eligible).
   - **Configure Security Group**: Allow the following inbound rules:
     - SSH (port 22) for remote access.
     - HTTP (port 80) or HTTPS (port 443) if Jenkins will be publicly accessible.
     - Custom TCP (port 8080) for Jenkins.
     - SMTP (port 25) (used for sending email).
     - SMTP (port 465)
     - Custom TCP (port 2000-11000) for Docker if you want to use Docker remotely.
   - **Key Pair**: Create a new key pair or use an existing one to SSH into the instance.

Name	Security group rule ID	Port range	Protocol	Source	Security groups
–	sgr-0970bf9e558295fa2	6443	TCP	0.0.0.0/0	jenkins
–	sgr-05bfb39fd7d8932a1	22	TCP	0.0.0.0/0	jenkins
–	sgr-0669632c14d96c052	465	TCP	0.0.0.0/0	jenkins
–	sgr-0377b430cc4484b64	80	TCP	0.0.0.0/0	jenkins
–	sgr-0c0625d7b2d0bc021	2000 - 11000	TCP	0.0.0.0/0	jenkins
–	sgr-009f0420991eabd8d	443	TCP	0.0.0.0/0	jenkins

3. **Launch the Instance:**
   - Review the settings and click **Launch**.
4. **Connect to the EC2 Instance:** Once your instance is running, connect to it via SSH using the key pair:

Step 2: Installing Java and Jenkins on the EC2 Instance

```
!#/bin/bash
#Step 1: Update your system
sudo apt update
sudo apt upgrade

#Step 2: Install Java
sudo apt install openjdk-11-jdk -y

#Step 3: Add Jenkins repository
To install the latest stable version of Jenkins, add the Jenkins repository and import the GPG key.

wget -q -O - https://pkg.jenkins.io/debian/jenkins.io.key | sudo apt-key add -
sudo sh -c 'echo deb http://pkg.jenkins.io/debian-stable binary/ > /etc/apt/sources.list.d/jenkins.list'

#Step 4: Install Jenkins
sudo apt update
sudo apt install jenkins -y

#Step 5:Start Jenkins
sudo systemctl enable Jenkins
sudo systemctl start Jenkins
sudo systemctl status Jenkins
```

Save the script to a file, e.g., **install_jenkins.sh.**

- Make it executable:

chmod +x install_jenkins.sh

- Run the script:

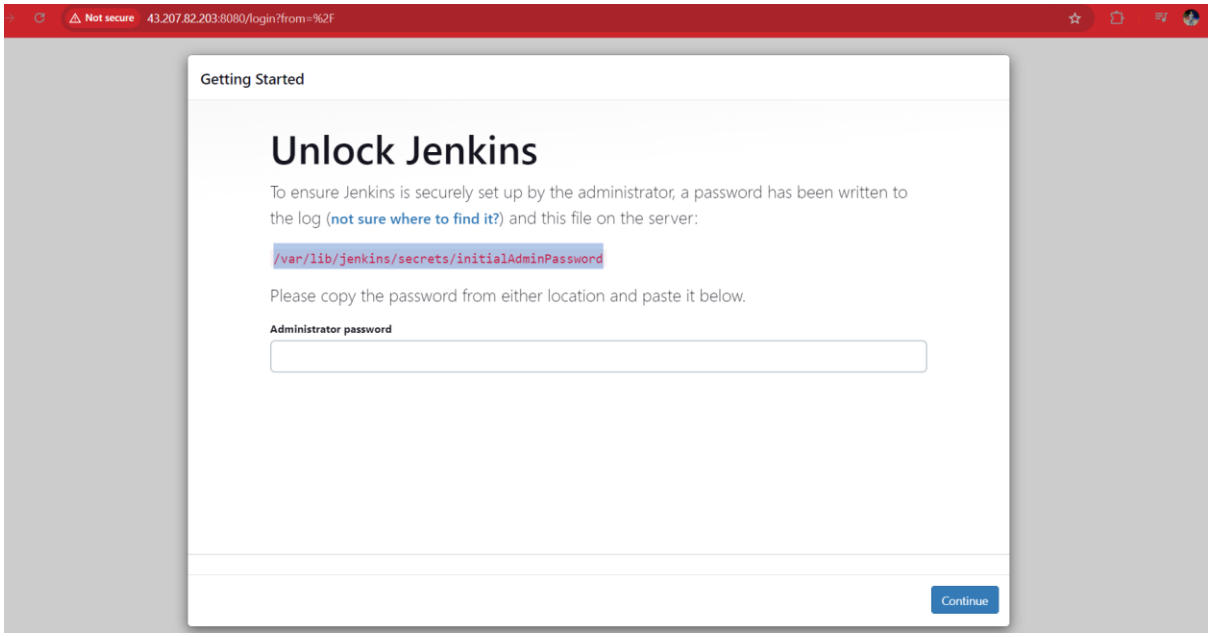
sudo ./install_jenkins.sh

```
root@ip-172-31-37-169:~# systemctl status jenkins.service
● jenkins.service - Jenkins Continuous Integration Server
     Loaded: loaded (/usr/lib/systemd/system/jenkins.service; enabled; preset: enabled)
     Active: active (running) since Sat 2024-09-28 18:10:16 UTC; 35min ago
   Main PID: 5857 (java)
      Tasks: 47 (limit: 9367)
     Memory: 881.8M (peak: 883.4M)
        CPU: 31.290s
     CGroup: /system.slice/jenkins.service
             └─5857 /usr/bin/java -Djava.awt.headless=true -jar /usr/share/java/jenkins.war --webroot=/var/cache/jenkins/w

Sep 28 18:10:10 ip-172-31-37-169 jenkins[5857]: 75d8e13ffaae4a4daa36738e1c9b0e01
Sep 28 18:10:10 ip-172-31-37-169 jenkins[5857]: This may also be found at: /var/lib/jenkins/secrets/initialAdminPassword
Sep 28 18:10:10 ip-172-31-37-169 jenkins[5857]: *************************************************************
Sep 28 18:10:10 ip-172-31-37-169 jenkins[5857]: *************************************************************
Sep 28 18:10:10 ip-172-31-37-169 jenkins[5857]: *************************************************************
```

Access Jenkins:

- Jenkins runs on port 8080. Access it in your browser:

- http://your-ec2-public-ip:8080



- To get the initial administrator password:
- **sudo cat /var/lib/jenkins/secrets/initialAdminPassword**

```
root@ip-172-31-37-169:~#
root@ip-172-31-37-169:~# cat  /var/lib/jenkins/secrets/initialAdminPassword
75d8e13ffaae4a4daa36738e1c9b0e01
root@ip-172-31-37-169:~#
```

- Copy the password, paste it into the Jenkins web interface, and follow the prompts to install suggested plugins and create an admin user.

Step 3: Installing Docker on the Same Server

```
!#/bin/bash
#docker Installation
# Add Docker's official GPG key:
sudo apt-get update
sudo apt-get install ca-certificates curl -y
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

# Add the 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
sudo apt-get update

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

~
```

Save the script to a file, e.g., **install_docker.sh.**

- Make it executable:

chmod +x install_docker.sh

- Run the script:

sudo./install_docker.sh

```
root@ip-172-31-37-169:~# systemctl status docker.service
● docker.service - Docker Application Container Engine
     Loaded: loaded (/usr/lib/systemd/system/docker.service; enabled; preset: enabled)
     Active: active (running) since Sat 2024-09-28 18:15:58 UTC; 43min ago
TriggeredBy: ● docker.socket
       Docs: https://docs.docker.com
   Main PID: 7270 (dockerd)
      Tasks: 9
     Memory: 20.3M (peak: 21.1M)
        CPU: 478ms
     CGroup: /system.slice/docker.service
             └─7270 /usr/bin/dockerd -H fd:// --containerd=/run/containerd/containerd.sock

Sep 28 18:15:57 ip-172-31-37-169 dockerd[7270]: time="2024-09-28T18:15:57.467276935Z" level=info msg="Starting up"
Sep 28 18:15:57 ip-172-31-37-169 dockerd[7270]: time="2024-09-28T18:15:57.469994727Z" level=info msg="detected 127.0.0.53 n
Sep 28 18:15:57 ip-172-31-37-169 dockerd[7270]: time="2024-09-28T18:15:57.745850328Z" level=info msg="Loading containers: s
Sep 28 18:15:58 ip-172-31-37-169 dockerd[7270]: time="2024-09-28T18:15:58.035670474Z" level=info msg="Loading containers: d
```

Install Trivy

To install **Trivy**, a security scanning tool for vulnerabilities in container images and file systems

```
!#/bin/bash
# install trivy
sudo apt-get install wget apt-transport-https gnupg lsb-release
wget -qO - https://aquasecurity.github.io/trivy-repo/deb/public.key | sudo apt-key add -
echo deb 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 -y
sudo apt-get install trivy -y
```

Save the script to a file, e.g., **install_Trivy.sh.**

- Make it executable:

chmod +x install_ Trivy.sh

- Run the script:

sudo./install_ Trivy.sh

Verify Installation

After installation, verify that Trivy is installed and working:

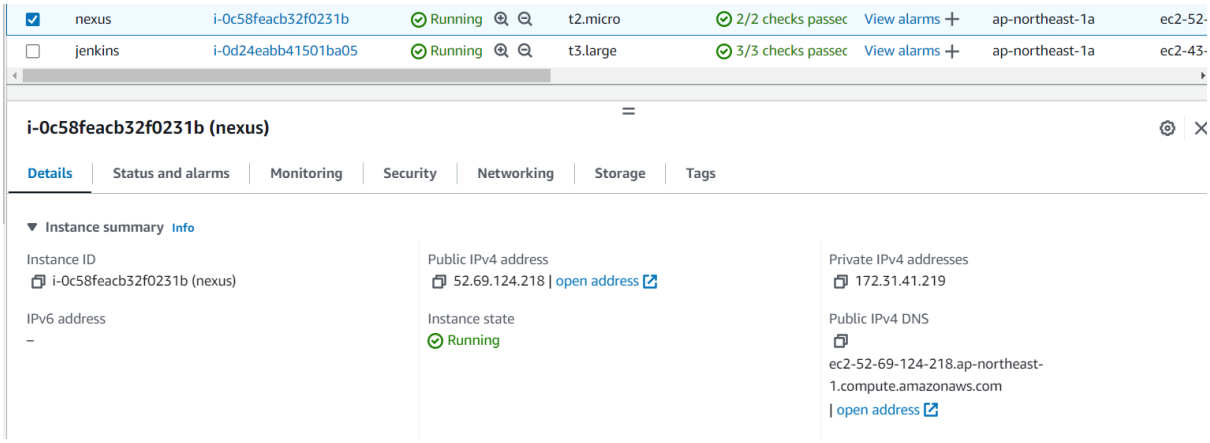
trivy –version

```
root@jenkins:~#
root@jenkins:~# trivy --version
Version: 0.55.2
root@jenkins:~#
root@jenkins:~#
```

To setup Nexus on an Ubuntu server, here's a detailed step-by-step guide

Launch an Nexus EC2 Instance:

- Navigate to **EC2 Dashboard** and click **Launch Instance**.
- Choose an **Amazon Machine Image (AMI)**: Select **Ubuntu 22.04** (or the latest stable version).
- Choose an **Instance Type**: Select a type like **t2.large** for testing or small workloads (free tier eligible).
- **Configure Security Group**: same security group used
- **Key Pair**: used key pair existing one to SSH into the instance.



Step 3: Installing Docker on the Same Server

```
!#/bin/bash
#docker Installation
# Add Docker's official GPG key:
sudo apt-get update
sudo apt-get install ca-certificates curl -y
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

# Add the 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
sudo apt-get update

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

~
```

Save the script to a file, e.g., **install_docker.sh.**

- Make it executable:

  chmod +x install_docker.sh

- Run the script:

  sudo./install_docker.sh

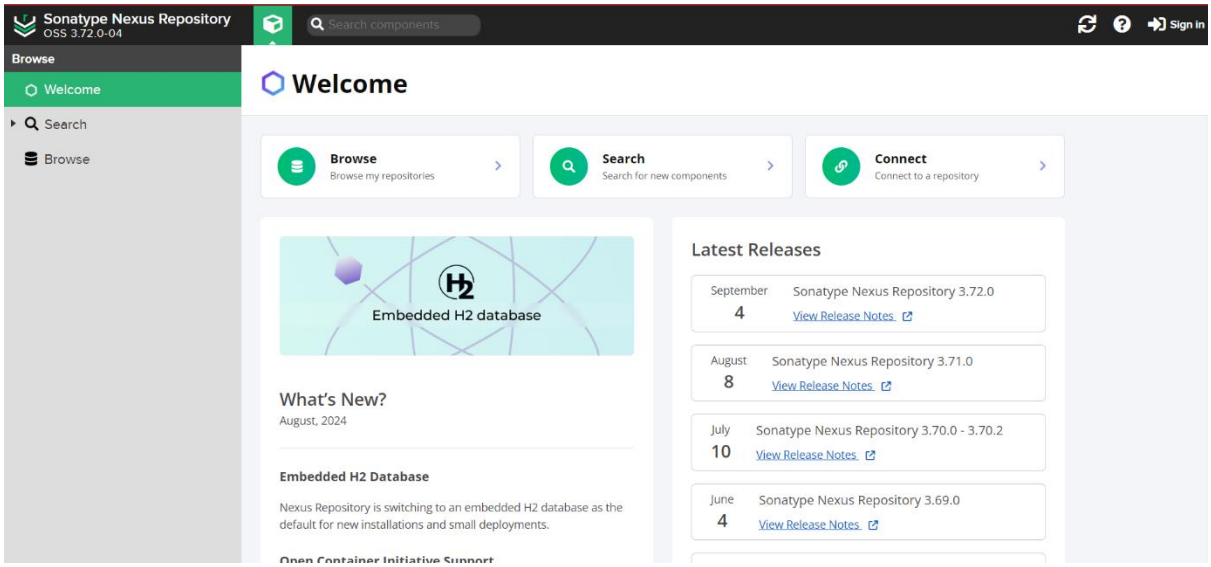
```
root@ip-172-31-37-169:~# systemctl status docker.service
● docker.service - Docker Application Container Engine
     Loaded: loaded (/usr/lib/systemd/system/docker.service; enabled; preset: enabled)
     Active: active (running) since Sat 2024-09-28 18:15:58 UTC; 43min ago
TriggeredBy: ● docker.socket
       Docs: https://docs.docker.com
   Main PID: 7270 (dockerd)
      Tasks: 9
     Memory: 20.3M (peak: 21.1M)
        CPU: 478ms
     CGroup: /system.slice/docker.service
             └─7270 /usr/bin/dockerd -H fd:// --containerd=/run/containerd/containerd.sock

Sep 28 18:15:57 ip-172-31-37-169 dockerd[7270]: time="2024-09-28T18:15:57.467276935Z" level=info msg="Starting up"
Sep 28 18:15:57 ip-172-31-37-169 dockerd[7270]: time="2024-09-28T18:15:57.469994727Z" level=info msg="detected 127.0.0.53 n
Sep 28 18:15:57 ip-172-31-37-169 dockerd[7270]: time="2024-09-28T18:15:57.745850328Z" level=info msg="Loading containers: s
Sep 28 18:15:58 ip-172-31-37-169 dockerd[7270]: time="2024-09-28T18:15:58.035670474Z" level=info msg="Loading containers: d
```

To run Nexus using Docker, you can use the following command:

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

Access Nexus-server**:**

- Nexus runs on port 8081 Access it in your browser:
- http://your-ec2-public-ip:8081



Steps to Set or Change the Nexus Username and Password

Log in using the default credentials:

- **Username:** `admin`
- **Password:** (Retrieve this from the `admin.password` file).

Retrieve the Admin Password:

To access the file, you can either:

- Use `docker exec` to get a shell in the running Nexus container:

```
docker exec -it <nexus-container-name-id> /bin/sh
cat sonatype-work/nexus3/admin.password
```

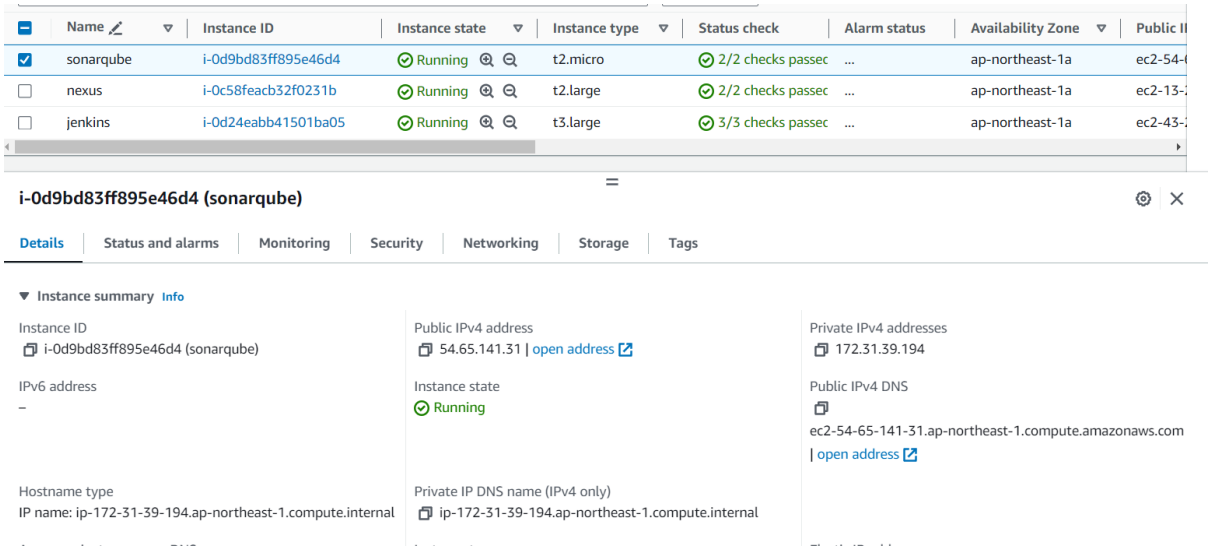
```
bash-4.4$
bash-4.4$ ls
nexus  sonatype-work  start-nexus-repository-manager.sh
bash-4.4$
bash-4.4$ cd sonatype-work/nexus3/
bash-4.4$
bash-4.4$ ls
admin.password  blobs  cache  db  elasticsearch  etc  generated-bundles  instances  javaprefs  karaf.pid  keystores  lock  log  port  restore-from-backup
bash-4.4$
bash-4.4$ cat admin.password
813d06e1-759e-4d8f-9b6f-ef1a3101a39fbash-4.4$
```

- Change the Admin Password: Update the password and save the changes.

To setup Nexus on an Ubuntu server, here's a detailed step-by-step guide

Launch an Snarqube EC2 Instance:

- Navigate to **EC2 Dashboard** and click **Launch Instance**.
- Choose an **Amazon Machine Image (AMI)**: Select **Ubuntu 22.04** (or the latest stable version).
- Choose an **Instance Type**: Select a type like **t2.micro** for testing or small workloads (free tier eligible).
- **Configure Security Group**: same security group used
- **Key Pair**: used key pair existing one to SSH into the instance.



Step 3: Installing Docker on the Same Server

```
!#/bin/bash
#docker Installation
# Add Docker's official GPG key:
sudo apt-get update
sudo apt-get install ca-certificates curl -y
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

# Add the 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
sudo apt-get update

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

~
```

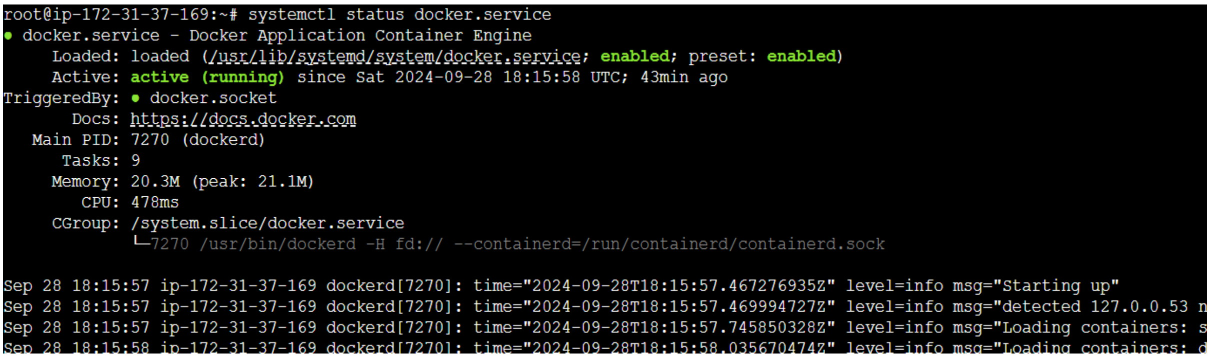
Save the script to a file, e.g., **install_docker.sh.**

- Make it executable:

chmod +x install_docker.sh

- Run the script:

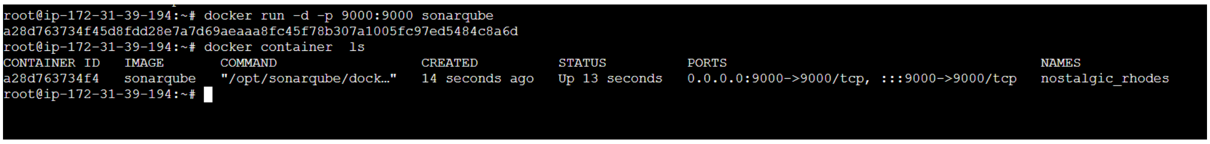
sudo./install_docker.sh



```
root@ip-172-31-37-169:~# systemctl status docker.service
● docker.service - Docker Application Container Engine
     Loaded: loaded (/usr/lib/systemd/system/docker.service; enabled; preset: enabled)
     Active: active (running) since Sat 2024-09-28 18:15:58 UTC; 43min ago
TriggeredBy: ● docker.socket
       Docs: https://docs.docker.com
   Main PID: 7270 (dockerd)
      Tasks: 9
     Memory: 20.3M (peak: 21.1M)
        CPU: 478ms
     CGroup: /system.slice/docker.service
             └─7270 /usr/bin/dockerd -H fd:// --containerd=/run/containerd/containerd.sock

Sep 28 18:15:57 ip-172-31-37-169 dockerd[7270]: time="2024-09-28T18:15:57.467276935Z" level=info msg="Starting up"
Sep 28 18:15:57 ip-172-31-37-169 dockerd[7270]: time="2024-09-28T18:15:57.469994727Z" level=info msg="detected 127.0.0.53 n
Sep 28 18:15:57 ip-172-31-37-169 dockerd[7270]: time="2024-09-28T18:15:57.745850328Z" level=info msg="Loading containers: s
Sep 28 18:15:58 ip-172-31-37-169 dockerd[7270]: time="2024-09-28T18:15:58.035670474Z" level=info msg="Loading containers: d
```

To run Nexus using Docker, you can use the following command:

docker run -d -p 9000:9000   sonarqube

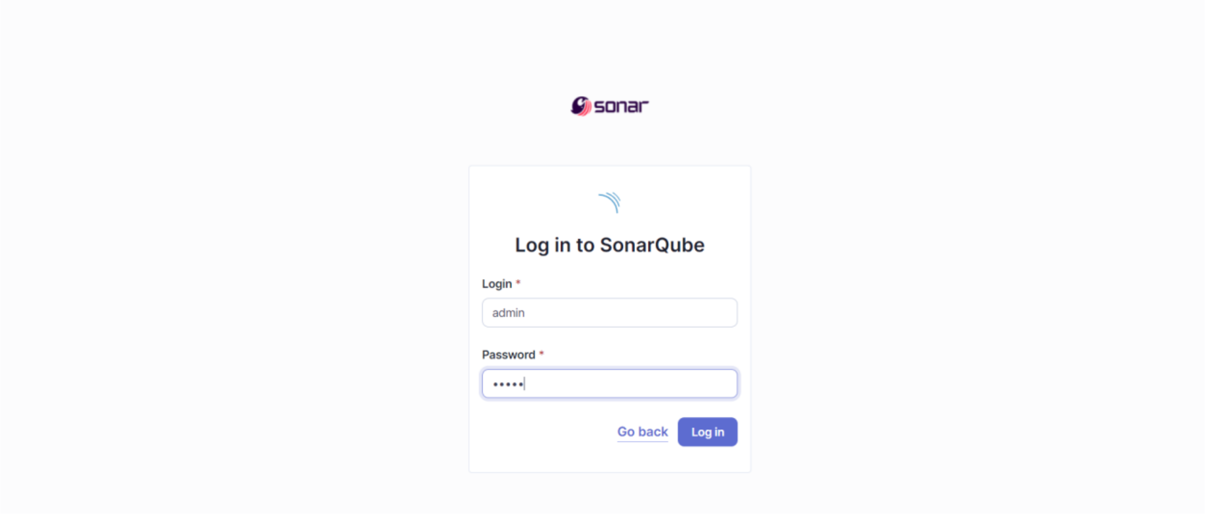


```
root@ip-172-31-39-194:~# docker run -d -p 9000:9000 sonarqube
a28d763734f45d8fdd28e7a7d69aeaaa8fc45f78b307a1005fc97ed5484c8a6d
root@ip-172-31-39-194:~# docker container  ls
CONTAINER ID   IMAGE       COMMAND                  CREATED          STATUS          PORTS                                       NAMES
a28d763734f4   sonarqube   "/opt/sonarqube/dock…"   14 seconds ago   Up 13 seconds   0.0.0.0:9000->9000/tcp, :::9000->9000/tcp   nostalgic_rhodes
root@ip-172-31-39-194:~# █
```

Access Sonarqube-server**:**

- Nexus runs on port 9000 Access it in your browser:

http://your-ec2-public-ip:9000



Log in to SonarQube

Login *

admin

Password *

•••••

//

Search and Install Plugins Jenkins server:

- Dashboard--Manage Jenkins--Plugins-- Available plugins

  Here's a list of plugins to install:

  - SonarQube Scanner
  - Maven Integration
  - Config File Provider
  - Pipeline: REST API
  - Pipeline: Stage View
  - Docker
  - Kubernetes Client API
  - Kubernetes Credentials
  - Kubernetes CLI
  - Eclipse Temurin installer

To manage tools in Jenkins, such as JDK, Maven, Gradle, Docker, and others, follow these steps to access the **Tools** configuration in the **Manage Jenkins** section:

Manage Jenkins:

- From the Jenkins dashboard, click on **Manage Jenkins**-- Tools.

configure Jenkins tools like JDK, Git, Maven, SonarQube Scanner, and Docker from the **Manage Jenkins** > **Global Tool Configuration**:

1. JDK Installation

- **Name:** `jdk17`
- **Install Automatically:** Yes (check this)
- **Install from adoptium.net**
- **Version:** Select `jdk-17.0.11+9`
- **Add Installer:** This will automatically download and install JDK version 17 from Adoptium.

2. Git Installation

- **Name:** `Default`
- **Path to Git executable:** Leave this as `git` (if Git is installed locally, Jenkins will use it).
- **Install Automatically:** Yes (optional, if you want Jenkins to install it automatically).
- If selected, choose to install Git automatically from the package manager.

3. Maven Installation

- **Name:** `maven3`
- **Install Automatically:** Yes (check this)
- **Install from Apache**
- **Version:** Select `3.9.9`
- **Add Installer:** This will install Maven version 3.9.9 automatically from the Apache repository.

4. SonarQube Scanner Installation

- **Name:** `sonar-scan`
- **Install Automatically:** Yes (check this)
- **Install from Maven Central**
- **Version:** Select `SonarQube Scanner 6.2.0.4584`
- **Add Installer:** This installs the selected version of SonarQube Scanner.

5. Docker Installation

- **Name:** `docker`
- **Install Automatically:** Yes (check this)
- **Download from docker.com**
- **Version:** Select `latest`
- **Add Installer:** This will install the latest version of Docker.

Steps to Create a Pipeline in Jenkins

Create a New Item

- Click on **New Item** in the left sidebar.

Enter Pipeline Name

- Enter a name for your pipeline (e.g., `MyFirstPipeline`).

Select Pipeline Type

- Choose **Pipeline** from the list of options.
- Click **OK**.

```
pipeline {

  agent any

  tools {

    jdk "jdk17"

    maven "maven3"

  }



  stages {
```

```
        stage('git checkout') {
            steps {
                git branch: 'main', credentialsId: 'github-jenkins', url:
'https://github.com/vaibhav0342/django-notes-app.git'
            }
        }

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

        stage('test') {
            steps {
                sh 'mvn test'
            }
        }

        stage('trivy fs scan') {
            steps {
                echo 'trivy fs --format table -o fs.html .'
            }
        }
    }
}
```

- Dashboard--- pipeline---Pipeline Syntax

Steps

Sample Step

git: Git

git?

Repository URL?

https://github

Branch?

main

Credentials?

Vaibhav0342/****** (github and jenkins integreation)

Credentials Provider: Jenkins

Add Credentials

Domain

Global credentials (unrestricted)

Kind

Username with password

Scope?

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

Username?

vaibhav0342

☐ Treat username as secret?

Password?

****

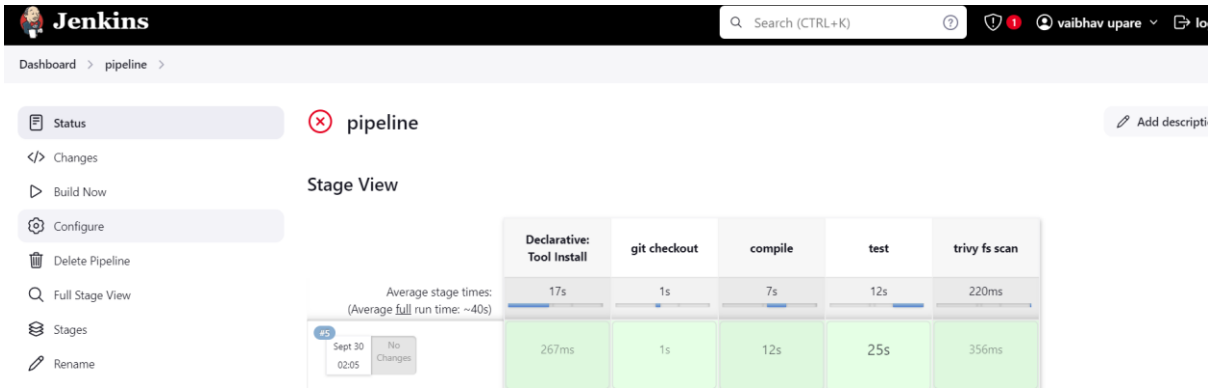
Token : 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

ID?

git and jenki

Description?

git and jenki



Step : Log in to SonarQube

Select Administration

- Go to the **Security** tab
- Under the **Generate Tokens** section

token is generated, **copy it** and store it safely

Step : Log in to jenkins server

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

New credentials

Kind

Secret text

Scope?

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

Secret

****

squ_b32a603c7455c8fbf3c604992bd6cf3f76828abc

ID?

sonar-toker

Description?

sonar-token

- Dashboard-----Manage Jenkins---- System

SonarQube installations

List of SonarQube installations
Name
sonar
Server URL
Default is http://localhost:9000
http://52.194
Server authentication token
SonarQube authentication token. Mandatory when anonymous access is disabled.
sonar-token

- Dashboard---pipeline---Pipeline Syntax

Steps

Sample Step

withSonarQubeEnv: Prepare SonarQube Scanner environment

withSonarQubeEnv?

Server authentication token

sonar-token

Generate Pipeline Script

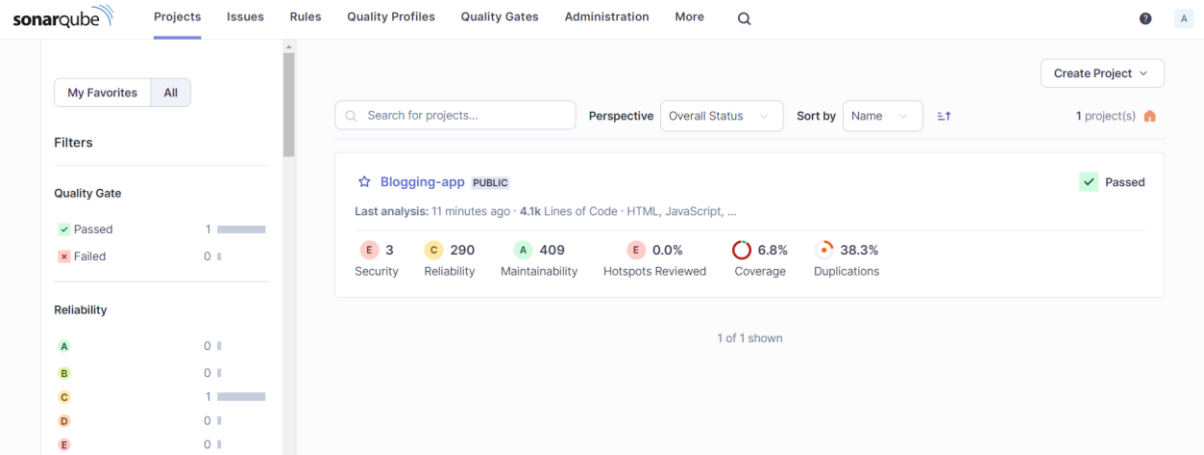
```
withSonarQubeEnv(credentialsId: 'sonar-token') {
    // some block
}
```

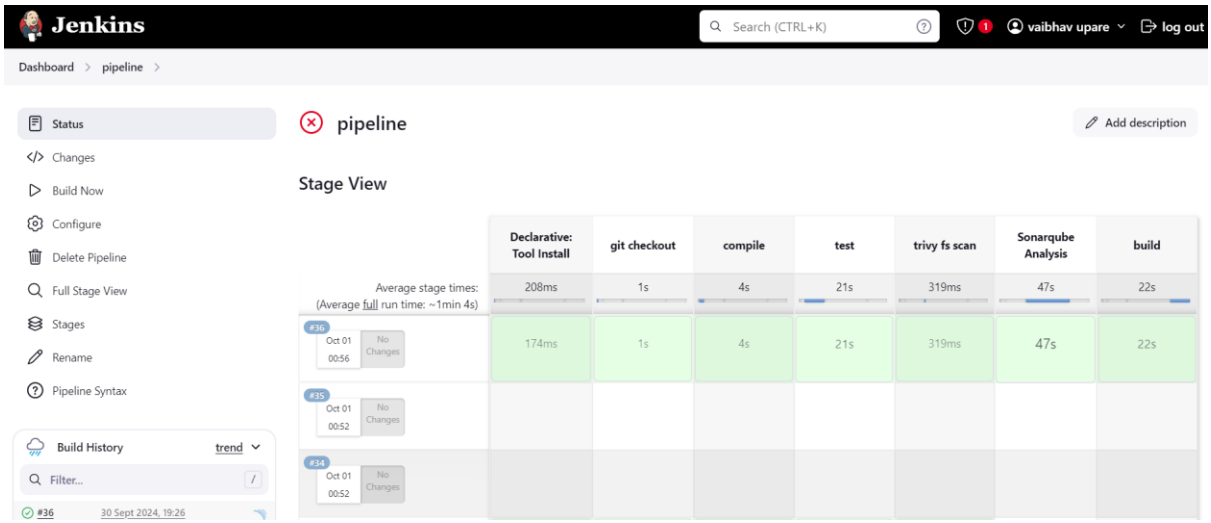
==========

```
pipeline {
    agent any
    tools {
        jdk "jdk17"
        maven "maven3"
```

```
    }

    environment {  // Corrected spelling

        SCANNER_HOME= tool 'sonar-scanner'  // SonarQube server configured in
Jenkins

    }





    stages {

        stage('git checkout') {

            steps {

                git branch: 'main', credentialsId: 'github-jenkins', url:
'https://github.com/vaibhav0342/Blogging-App.git'

            }

        }



        stage('compile') {

            steps {

                sh 'mvn compile'

            }

        }



        stage('test') {

            steps {

                sh 'mvn test'

            }

        }
```

```
        stage('trivy fs scan') {

            steps {

                echo 'trivy fs --format table -o fs.html .'

            }

        }



        stage('Sonarqube Analysis') {

            steps {

                withSonarQubeEnv('sonar-server') {  // 'sonar-server' should match the
SonarQube installation name configured in Jenkins

                    sh '''$SCANNER_HOME/bin/sonar-scanner  -
Dsonar.projectName=Blogging-app -Dsonar.projectKey=Blogging-app -
Dsonar.java.binaries=target'''

                }

            }

        }

         stage('build') {

            steps {

                sh 'mvn package'

            }

        }

    }
```





Step : Log in to Nexus server

To integrate SonarQube with your Maven project and configure it for both release and snapshot builds copy

http://13.230.89.139:8081/repository/maven-releases/

http://13.230.89.139:8081/repository/maven-snapshots/

edit the `pom.xml` file in your GitHub project

```
<repository>
        <id>maven-releases</id>
        <url>http://13.115.59.141:8081/repository/maven-releases/</url>
    </repository>
    <snapshotRepository>
        <id>maven-snapshots</id>
```

```
        <url>http://13.115.59.141:8081/repository/maven-snapshots/</url>

    </snapshotRepository>
```

- Dashboard---- Manage Jenkins---Managed files

New configuration

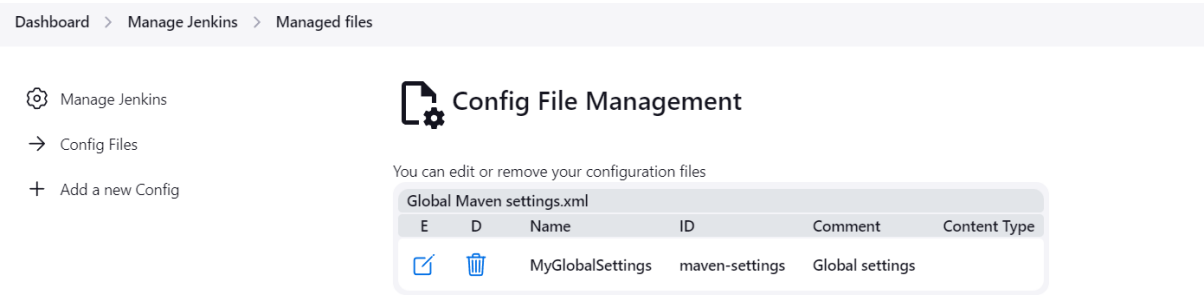
Type

Global Maven settings.xml

ID of the config file---add name {maven-seeting] –done

Edit file

```
<servers>

  <server>
    <id>maven-releases</id>
    <username>admin</username>
    <password>vaibhav</password>
  </server>

   <server>
    <id>maven-snapshots</id>
    <username>admin</username>
    <password>vaibhav</password>
  </server>
```



Steps

Sample Step

```
withMaven: Provide Maven environment
```

withMaven?

Maven?

maven3

JDK?

jdk17

Temporary Binary Directory?

Maven Settings Config?

--- Use system default settings or file path ---

Maven Settings File Path?

Global Maven Settings Config?

MyGlobalSettings

Global Maven Settings File Path?

Maven JVM Opts?

☑ Maven Traceability?

Maven Local Repository?

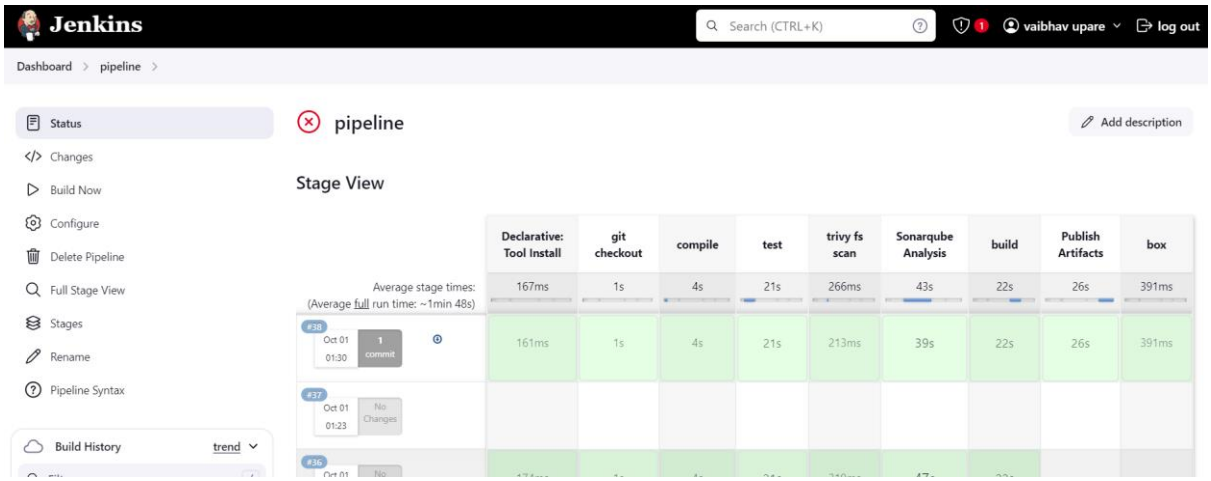
Publisher Strategy?

Implicit

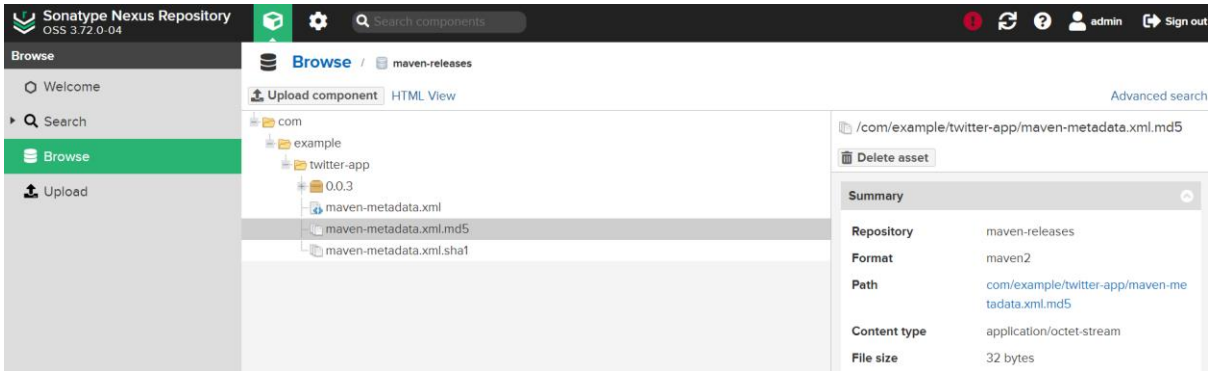
```
withMaven(globalMavenSettingsConfig: 'maven-settings', jdk: 'jdk17', maven: 'maven3',
mavenSettingsConfig: '', traceability: true) {
    // some block
}
```

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

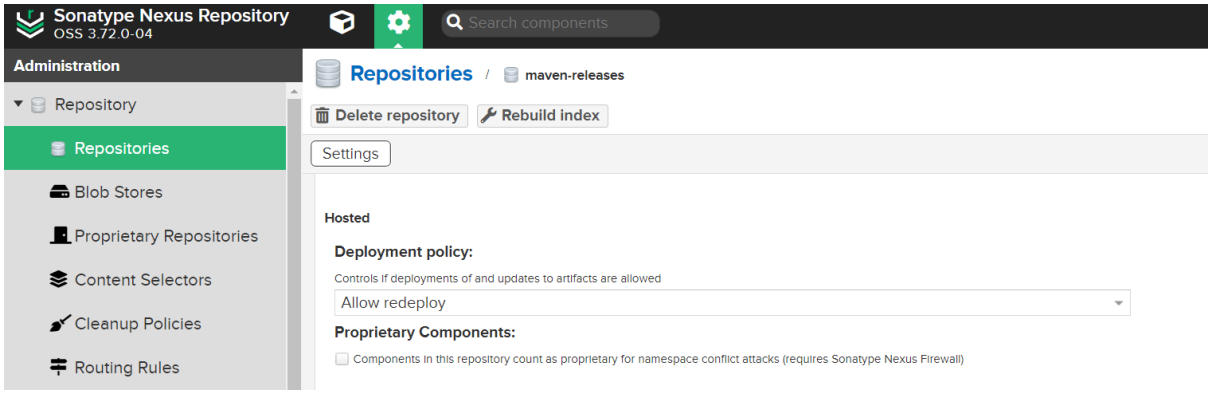
```
    }
  }
```



`mvn deploy`: This command is part of the Maven lifecycle and is used to deploy the built artifacts (like JAR or WAR files) to a remote repository (e.g., Nexus



- **Select the Repository**:
  - Navigate to **Repositories** under the **Administration** menu.
  - Find the repository you are working with (e.g., `maven-releases`).
- **Edit Repository Settings**:
  - Click on the repository name to open its configuration settings.
  - Look for the **Deployment Policy** setting.
  - Set it to **Allow Redeploy** or **Redeployable** (the exact wording may vary).
- **Save Changes**: Click **Save** to apply the changes.



1. Log in to Docker Hub

log in to Docker Hub and create a private repository

- Dashboard-----pipeline-------Pipeline Syntax

Steps

Sample Step

withDockerRegistry: Sets up Docker registry endpoint

withDockerRegistry?

Docker registry URL?

Registry credentials

vaibhav0342/****** (docker-hub)

Add

-

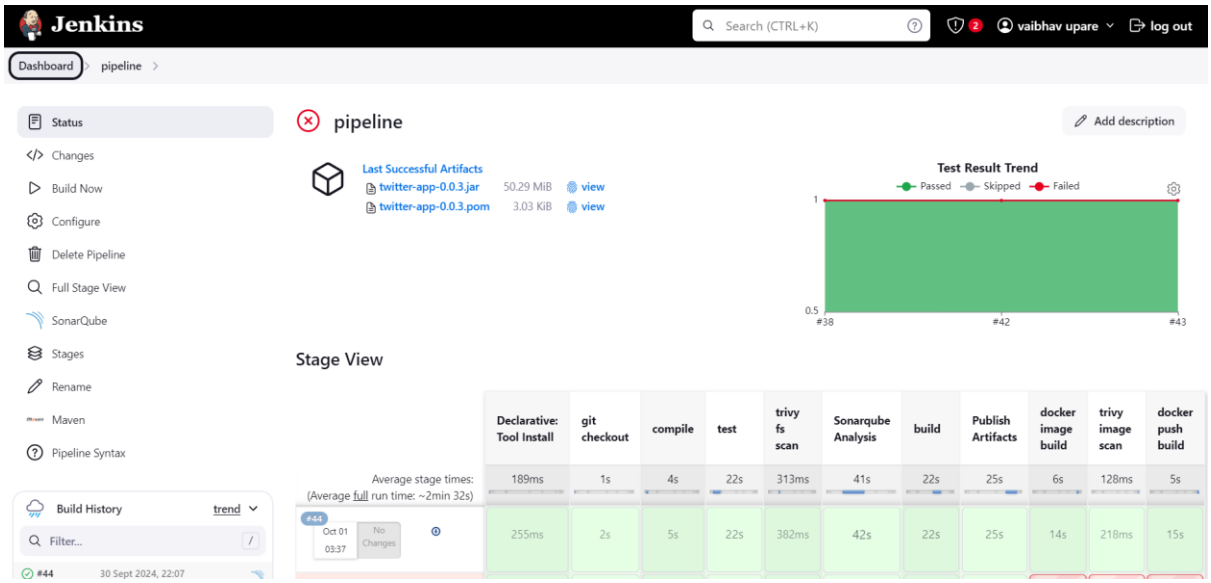
Docker installation

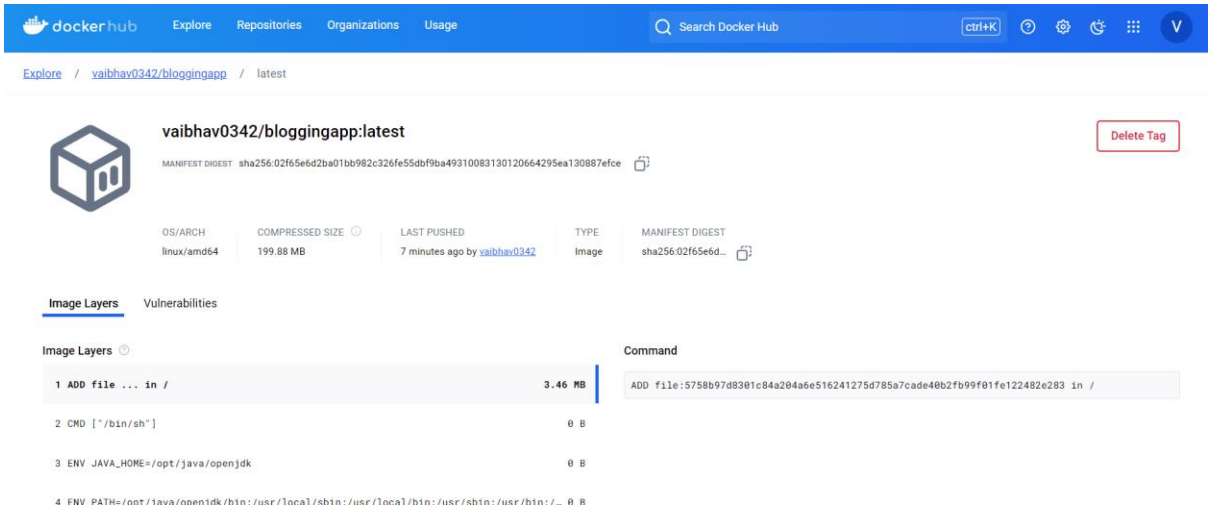
docker

```
// This step should not normally be used in your script. Consult the inline help for details.
withDockerRegistry(credentialsId: 'docker-hub', toolName: 'docker') {
    // some block
}
```

Steps to Resolve Docker Permissions Issue
Run the following command in your terminal: **sudo usermod -aG docker Jenkins**
Restart Jenkins Service : **sudo systemctl restart Jenkins**
Check Docker Service Status**: sudo systemctl status docker**

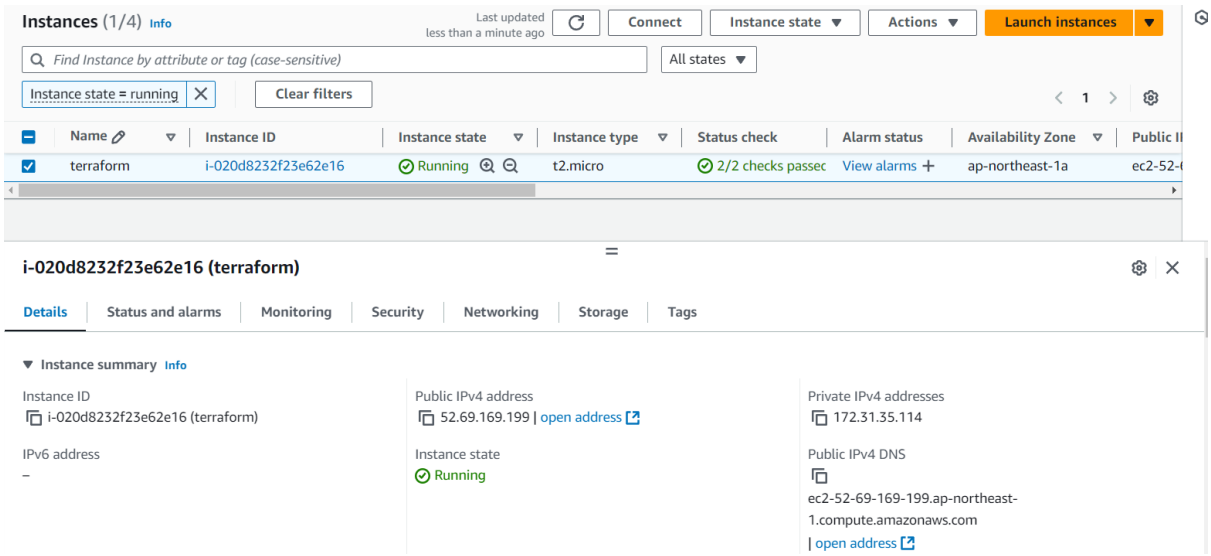
```
stage('docker image build') {
        steps {
           script {
withDockerRegistry(credentialsId: 'docker-hub', toolName: 'docker') {
     sh 'docker build -t vaibhav0342/bloggingapp:latest .'
                }
            }
        }
     }
       stage('trivy image scan') {
        steps {
           echo 'trivy image --format table -o image.html vaibhav0342/bloggingapp:latest'
        }
     }
      stage('docker push build') {
        steps {
           script {
withDockerRegistry(credentialsId: 'docker-hub', toolName: 'docker') {
     sh 'docker push vaibhav0342/bloggingapp:latest'
                }
            }
        }
     }
   }
}
```





Launch an Terraform EC2 Instance:

- Navigate to **EC2 Dashboard** and click **Launch Instance**.
- Choose an **Amazon Machine Image (AMI)**: Select **Ubuntu 22.04** (or the latest stable version).
- Choose an **Instance Type**: Select a type like **t2.micro** for testing or small workloads (free tier eligible).
- **Configure Security Group**: same security group used
- **Key Pair**: used key pair existing one to SSH into the instance.



To install the AWS Command Line Interface (CLI) on Ubuntu To install the AWS Command Line Interface (CLI) on Ubuntu

sudo apt update
sudo apt install unzip curl –y
curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o "awscliv2.zip"
unzip awscliv2.zip
aws –version

To install the Terraform Command : **snap install terraform –classic**

{ To create an IAM user in AWS, attach a role (e.g., `AdministratorAccess` or `AmazonS3FullAccess`), and generate access keys for programmatic access }

After installation, you can configure the AWS CLI with your AWS credentials by running:
aws configure
AKIARJCTHA3ZMXFFYHHF
w16HajjS5GwonA0yGaxw1lQ2Ml+8SRPB2QUhZV/5

crerate folder **: mkdir terra**
change directory **: cd terra**
paste code and save **:vim main.tf**
:vm output.tf
:vim variables.tf
Terraform init
Terraform plan
Terraform validate
Terraform apply --auto-approve

Install kubectl command **: sudo snap install kubectl - - classic**

aws eks --region ap-south-1 update-kubeconfig --name devopsshack-cluster

kubectl get all

create service : vim svc.yaml

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: jenkins
  namespace: webapps
```

kubectl create namespace webapps

kubectl apply –f svc.yaml

create role: vim role.yaml

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: app-role
  namespace: webapps
rules:
  - apiGroups:
      - ""
      - apps
      - autoscaling
```

```
      - batch
      - extensions
      - policy
      - rbac.authorization.k8s.io
    resources:
      - pods
      - secrets
      - componentstatuses
      - configmaps
      - daemonsets
      - deployments
      - events
      - endpoints
      - horizontalpodautoscalers
      - ingress
      - jobs
      - limitranges
      - namespaces
      - nodes
      - pods
      - persistentvolumes
      - persistentvolumeclaims
      - resourcequotas
      - replicasets
      - replicationcontrollers
      - serviceaccounts
      - services
    verbs: ["get", "list", "watch", "create", "update", "patch", "delete"]
```

kubectl create –f role.yaml

create bind role for this service account: vim bind.yaml

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: app-rolebinding
  namespace: webapps
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: app-role
subjects:
- namespace: webapps
  kind: ServiceAccount
  name: Jenkins
```

create token **:** vim Jenkins-token.yaml

apiVersion: v1
kind: Secret
metadata:
** name: secret-sa-sample**
** annotations:**
**   kubernetes.io/service-account.name: "jenkins"**
type: kubernetes.io/service-account-token

kubecctl create –f Jenkins-token.yaml -n webapps

```
root@terraform:~/terra# aws eks --region ap-south-1 update-kubeconfig --name devopsshack-cluster
Added new context arn:aws:eks:ap-south-1:088221353714:cluster/devopsshack-cluster to /root/.kube/config
root@terraform:~/terra# kubectl get all
NAME                 TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
service/kubernetes   ClusterIP   172.20.0.1   <none>        443/TCP   11m
root@terraform:~/terra# cd
root@terraform:~# vim svc.yaml
root@terraform:~# kubectl create namespace webapps
namespace/webapps created
root@terraform:~# kubectl apply -f svc.yaml
.aws/          .bashrc        .profile       .terraform.d/  aws/           snap/          terra/
.bash_history  .kube/         .ssh/          .viminfo       awscliv2.zip   svc.yaml
root@terraform:~# kubectl apply -f svc.yaml
serviceaccount/jenkins created
root@terraform:~# vim role.yaml
root@terraform:~# kubectl apply -f role.yaml
role.rbac.authorization.k8s.io/app-role created
root@terraform:~# vim bind.yaml
root@terraform:~# kubectl apply -f  bind.yaml
rolebinding.rbac.authorization.k8s.io/app-rolebinding created
root@terraform:~# vim jenkins-token.yaml
root@terraform:~# kubectl apply -f  jenkins-token.yaml
secret/secret-sa-sample created
root@terraform:~# kubectl apply -f  jenkins-token.yaml -n webapps
secret/secret-sa-sample created
```

secret/regcred created:

**kubectl create secret docker-registry regcred **
** --docker-server=https://index.docker.io/v1/ **
** --docker-username=vaibhav0342 **
** --docker-password=vaibhav0342 **
** --namespace=webapps**

Kubectl get secrets -n webapps
kubectl describe secret secret-sa-sample -n webapps

Copy the data :
eyJhbGciOiJSUzI1NiIsImtpZCI6ImdteEVIbjl1SEEtanMyMVFFenJvWVVCSjlWOElkVU5aS2hmeWlnWVZvOHcifQ.

Login Jenkins server : **add creadantial**

Install in jenkin server **: snap install kubectl --classic**

Steps

Sample Step

withKubeConfig: Configure Kubernetes CLI (kubectl)

withKubeConfig?

Credentials

k8-cred

Kubernetes server endpoint?

https://A895

Cluster name?

devopssha

Context name?

Namespace?

webapps

```
withKubeConfig(caCertificate: '', clusterName: ' devopsshack-cluster', contextName: '',
credentialsId: 'k8-cred', namespace: 'webapps', restrictKubeConfigAccess: false, serverUrl:
'https://A8956E433BAEE5E7C6B1D589C3FC8A28.gr7.ap-south-1.eks.amazonaws.com') {
   // some block
}
```

==========

```
stage('Deploy to K8s') {
        steps {
           withKubeConfig(caCertificate: '', clusterName: 'devopsshack-cluster',
contextName: '', credentialsId: 'k8-cred', namespace: 'webapps', restrictKubeConfigAccess:
false, serverUrl: 'https://A8956E433BAEE5E7C6B1D589C3FC8A28.gr7.ap-south-
1.eks.amazonaws.com') {
                sh 'kubectl create -f deployment-service.yml'
                sleep 20
           }
        }
     }

     stage('Check K8s Resources') {
```

```
        steps {
            withKubeConfig(caCertificate: '', clusterName: 'devopsshack-cluster',
contextName: '', credentialsId: 'k8-cred', namespace: 'webapps', restrictKubeConfigAccess:
false, serverUrl: 'https://A8956E433BAEE5E7C6B1D589C3FC8A28.gr7.ap-south-
1.eks.amazonaws.com') {
                sh 'kubectl get pods'
                sh 'kubectl get svc'
            }
        }
    }
  }
}
```