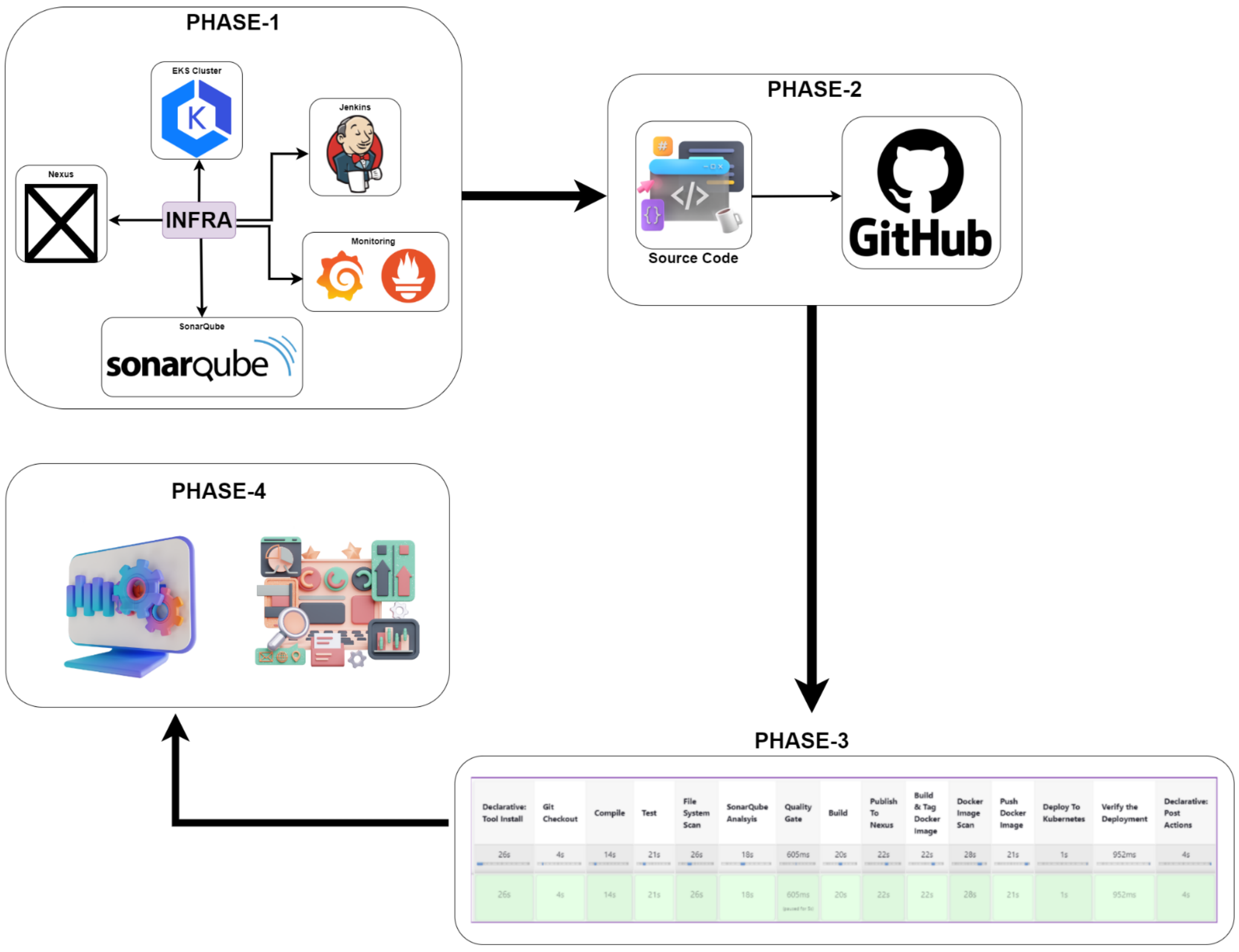
DevOps SHACK

DevOps Shack

ULTIMATE CICD PIPELINE PROJECT

[Click Here To Enrol To Batch-5 | DevOps & Cloud DevOps]



Declarative: Tool Install	Git Checkout	Compile	Test	File System Scan	SonarQube Analsyis	Quality Gate	Build	Publish To Nexus	Build & Tag Docker Image	Docker Image Scan	Push Docker Image	Deploy To Kubernetes	Verify the Deployment	Declarative: Post Actions
26s	4s	14s	21s	26s	18s	605ms	20s	22s	22s	28s	21s	1s	952ms	4s
26s	4s	14s	21s	26s	18s	605ms (paused for 5s)	20s	22s	22s	28s	21s	1s	952ms	4s

PHASE-1 | INFRA SETUP

#1 Creating 3 Ubuntu 24.04 VM Instance on AWS

1. Sign in to the AWS Management Console:

- Go to AWS Management Console.
- Sign in with your AWS account credentials.

2. Navigate to EC2:

- Type "EC2" in the search bar or select "Services" > "EC2" under the "Compute" section.

3. Launch Instance:

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

4. Choose an Amazon Machine Image (AMI):

- Select "Ubuntu" from the list of available AMIs.
- Choose "Ubuntu Server 24.04 LTS".
- Click "Select".

5. Choose an Instance Type:

- Select an instance type (e.g., t2.micro for testing).
- Click "Next: Configure Instance Details".

6. Configure Instance Details:

- Configure optional settings or leave them as default.
- Click "Next: Add Storage".

7. Add Storage:

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

8. Add Tags:

- Optionally, add tags for better organization.
- Click "Next: Configure Security Group".

9. Configure Security Group:

- Allow SSH access (port 22) from your IP address.
- Optionally, allow other ports (e.g., HTTP port 80, HTTPS port 443).
- Click "Review and Launch".

10. Review and Launch:

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

11. Select Key Pair:

- Select an existing key pair or create a new one.
- Check the acknowledgment box.
- Click "Launch Instances".

12. Access Your Instance:

- Use an SSH client like MobaXterm:
  - Open MobaXterm and click "Session" > "SSH".
  - Enter the public IP address of your instance.
  - Select "Specify username" and enter "ubuntu".
  - Under "Advanced SSH settings", select "Use private key" and browse to your key pair file (.pem).
  - Click "OK" to connect.

13. Make sure to Install Docker on All 3 VMs

Step-by-Step Installation

1. **Install prerequisite packages:**

   ```
   sudo apt-get update
   sudo apt-get install ca-certificates curl
   ```

2. **Download and add Docker's official GPG key:**

   ```
   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
   ```

3. **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
   ```

4. **Update package index:**

   ```
   sudo apt-get update
   ```

5. **Install Docker packages:**

   ```
   sudo apt-get install docker-ce docker-ce-cli containerd.io -y
   ```

6. **Grant permission to Docker socket (optional, for convenience):**

   ```
   sudo chmod 666 /var/run/docker.sock
   ```

By following these steps, you should have successfully installed Docker on your Ubuntu system. You can now start using Docker to containerize and manage your applications.

Setting Up Jenkins on Ubuntu

Step-by-Step Installation

1. **Update the system:**

   ```
   sudo apt-get update
   sudo apt-get upgrade -y
   ```

2. **Install Java (Jenkins requires Java):**

   ```
   sudo apt install -y fontconfig openjdk-17-jre
   ```

3. **Add Jenkins repository key:**

   ```
   sudo wget -O /usr/share/keyrings/jenkins-keyring.asc
   https://pkg.jenkins.io/debian-stable/jenkins.io-2023.key
   ```

4. **Add Jenkins repository:**

   ```
   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
   ```

5. **Update the package index:**

   ```
   sudo apt-get update
   ```

6. **Install Jenkins:**

   ```
   sudo apt-get install -y jenkins
   ```

7. **Start and enable Jenkins:**

   ```
   sudo systemctl start jenkins
   sudo systemctl enable jenkins
   ```

8. **Access Jenkins:**

   - Open a web browser and go to `http://your_server_ip_or_domain:8080`.
   - You will see a page asking for the initial admin password. Retrieve it using:

     ```
     sudo cat /var/lib/jenkins/secrets/initialAdminPassword
     ```
   - Enter the password, install suggested plugins, and create your first admin user.

Installing Trivy on Jenkins Server

Step-by-Step Installation

1. **Install prerequisite packages:**

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

2. **Add Trivy repository key:**

   ```
   wget -qO - https://aquasecurity.github.io/trivy-repo/deb/public.key |
   sudo apt-key add -
   ```

3. **Add Trivy repository to sources:**

   ```
   echo deb https://aquasecurity.github.io/trivy-repo/deb $(lsb_release
   -sc) main | sudo tee -a /etc/apt/sources.list.d/trivy.list
   ```

4. **Update package index:**

   ```
   sudo apt-get update
   ```

5. **Install Trivy:**

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

Setting Up Nexus Repository Manager Using Docker

Step-by-Step Installation

1. **Pull the Nexus Docker image:**

   ```
   sudo docker pull sonatype/nexus3
   ```

2. **Run the Nexus container:**

   ```
   sudo docker run -d -p 8081:8081 --name nexus -v nexus-data:/nexus-
   data sonatype/nexus3
   ```

3. **Access Nexus:**

   - Open a web browser and go to `http://your_server_ip_or_domain:8081`.
   - The default username is `admin`. Retrieve the initial admin password from the log:

     ```
     sudo docker logs nexus 2>&1 | grep -i password
     ```

   - Complete the setup wizard.

Setting Up SonarQube Using Docker

Step-by-Step Installation

1. **Create a network for SonarQube and PostgreSQL:**

   ```
   sudo docker network create sonarnet
   ```

2. **Run PostgreSQL container:**

   ```
   sudo docker run -d --name sonarqube_db --network sonarnet -e
   POSTGRES_USER=sonar -e POSTGRES_PASSWORD=sonar -e
   POSTGRES_DB=sonarqube -v postgresql:/var/lib/postgresql -v
   postgresql_data:/var/lib/postgresql/data postgres:latest
   ```

3. **Run SonarQube container:**

   ```
   sudo docker run -d --name sonarqube --network sonarnet -p 9000:9000 -
   e sonar.jdbc.url=jdbc:postgresql://sonarqube_db:5432/sonarqube -e
   sonar.jdbc.username=sonar -e sonar.jdbc.password=sonar -v
   sonarqube_data:/opt/sonarqube/data -v
   sonarqube_extensions:/opt/sonarqube/extensions -v
   sonarqube_logs:/opt/sonarqube/logs sonarqube:latest
   ```

4. **Access SonarQube:**

   - Open a web browser and go to `http://your_server_ip_or_domain:9000`.
   - The default username and password are both `admin`.

#2 SetUp EKS Cluster

https://github.com/jaiswaladi246/EKS-Complete/blob/main/Steps-eks.md

PHASE-2 | Source Code SetUp

Project Repo: https://github.com/jaiswaladi246/Mission.git

Creating a Private Repository on GitHub and Pushing Source Code Using Git Bash

Part 1: Create a Private Repository on GitHub

1. **Sign in to GitHub:**
   - Go to GitHub.
   - Sign in with your GitHub account credentials.
2. **Create a New Repository:**
   - Click the "+" icon in the upper-right corner of the GitHub interface.
   - Select "New repository".
3. **Repository Details:**
   - **Repository name:** Enter a name for your repository.
   - **Description:** Optionally, add a description.
   - **Privacy:** Select "Private".
   - **Initialize repository:** Optionally, check "Add a README file".
   - Click "Create repository".

Part 2: Push Source Code from Local Using Git Bash

1. **Install Git Bash:**
   - Download and install Git Bash from Git for Windows.
2. **Open Git Bash:**
   - Navigate to the directory containing your source code.
   - Right-click in the folder and select "Git Bash Here".
3. **Initialize Git Repository (if not already a Git repository):**

```
git init
```

4. **Add Remote Repository:**
    - Copy the repository URL from GitHub (e.g., `https://github.com/username/repository.git`).
    - In Git Bash, add the remote repository:

```
git remote add origin https://github.com/username/repository.git
```

5. **Add Files to Git:**
    - Stage all files for the first commit:

```
git add .
```

6. **Commit Files:**
    - Commit the staged files with a commit message:

```
git commit -m "Initial commit"
```

7. **Push to GitHub:**
    - Push the local repository to GitHub:

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

PHASE-3 | CICD Pipeline

Detailed Documentation for Jenkins Pipeline

Overview

This Jenkins pipeline automates the build, test, security scan, deployment, and verification process of a Java project using Maven. The pipeline includes the following stages:

1. **Git Checkout**
2. **Compile**
3. **Test**
4. **Trivy Scan File System**
5. **SonarQube Analysis**
6. **Build**
7. **Deploy Artifacts To Nexus**
8. **Build & Tag Docker Image**
9. **Trivy Scan Image**
10. **Publish Docker Image**
11. **Deploy To Kubernetes (K8s)**
12. **Verify Deployment**

Additionally, the pipeline sends an email notification upon completion with the status and relevant reports.

Prerequisites

1. **Jenkins Setup:**
    - Jenkins installed and configured.
    - Required plugins installed: Pipeline, Git, Maven Integration, Docker, SonarQube Scanner, Trivy, Email Extension, Kubernetes CLI, and Configuration as Code Plugin.
2. **Tools and Credentials:**
    - JDK 17 (`jdk17`).
    - Maven 3 (`maven3`).

  - SonarQube Scanner (`sonar-scanner`).
  - Docker Registry credentials (`docker-cred`).
  - Git credentials (`git-cred`).
  - Kubernetes token (`k8-token`).

3. **Environment Configurations:**
  - SonarQube server configured in Jenkins.
  - Nexus repository configured in Jenkins.
  - Docker tool and registry configured.
  - Kubernetes cluster and namespace configured.

Jenkins Pipeline Script

```
pipeline {
    agent any

    tools {
        jdk 'jdk17'
        maven 'maven3'
    }

    environment {
        SCANNER_HOME = tool 'sonar-scanner'
    }

    stages {
        stage('Git Checkout') {
            steps {
                git branch: 'main', changelog: false, credentialsId: 'git-
cred', poll: false, url: 'https://github.com/jaiswaladi246/Mission.git'
            }
        }

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

        stage('Test') {
            steps {
                sh "mvn package -DskipTests=true"
            }
        }

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

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

        stage('Build') {
            steps {
                sh "mvn package -DskipTests=true"
            }
        }

        stage('Deploy Artifacts To Nexus') {
            steps {
                withMaven(globalMavenSettingsConfig: 'maven-setting', jdk:
'jdk17', maven: 'maven3', mavenSettingsConfig: '', traceability: true) {
                    sh "mvn deploy -DskipTests=true"
                }
            }
        }

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

        stage('Trivy Scan Image') {
            steps {
                sh "trivy image --format table -o trivy-image-report.html
adijaiswal/mission:latest"
            }
        }

        stage('Publish Docker Image') {
            steps {
                script {
                    withDockerRegistry(credentialsId: 'docker-cred',
toolName: 'docker') {
                        sh "docker push adijaiswal/mission:latest"
                    }
                }
            }
        }

        stage('Deploy To K8s') {
            steps {
                withKubeConfig(caCertificate: '', clusterName: 'DS-EKS',
contextName: '', credentialsId: 'k8-token', namespace: 'webapps',
restrictKubeConfigAccess: false, serverUrl:
'https://EA12CBD2F14726DD103E88821D89490F.gr7.ap-south-
1.eks.amazonaws.com') {
                    sh "kubectl apply -f ds.yml -n webapps"
                    sleep 60
                }
            }
        }

        stage('Verify Deployment') {
            steps {
                withKubeConfig(caCertificate: '', clusterName: 'DS-EKS',
contextName: '', credentialsId: 'k8-token', namespace: 'webapps',
restrictKubeConfigAccess: false, serverUrl:
'https://EA12CBD2F14726DD103E88821D89490F.gr7.ap-south-
1.eks.amazonaws.com') {
                    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:
${pipelineStatus.toUpperCase()}</h3>
                    </div>
                    <p>Check the <a href="${BUILD_URL}">console
output</a>.</p>
                    </div>
                    </body>
                    </html>
                """

                emailext (
                    subject: "${jobName} - Build ${buildNumber} -
${pipelineStatus.toUpperCase()}",
                    body: body,
                    to: 'jaiswaladi246@gmail.com',
                    from: 'jenkins@example.com',
                    replyTo: 'jenkins@example.com',
                    mimeType: 'text/html',
                    attachmentsPattern: 'trivy-image-report.html'
                )
            }
        }
    }
}
```

Detailed Breakdown of Pipeline Stages

1. Git Checkout

- **Purpose:** Checkout the source code from the GitHub repository.
- **Steps:**

```
git branch: 'main', changelog: false, credentialsId: 'git-cred',
poll: false, url: 'https://github.com/jaiswaladi246/Mission.git'
```

2. Compile

- **Purpose:** Compile the source code using Maven.
- **Steps:**

```
sh "mvn compile"
```

3. Test

- **Purpose:** Package the code and skip tests to speed up the process.
- **Steps:**

```
sh "mvn package -DskipTests=true"
```

4. Trivy Scan File System

- **Purpose:** Perform a security scan on the file system.
- **Steps:**

```
sh "trivy fs --format table -o trivy-fs-report.html ."
```

5. SonarQube Analysis

- **Purpose:** Analyze the code quality using SonarQube.
- **Steps:**

```
withSonarQubeEnv('sonar') {
    sh ''' $SCANNER_HOME/bin/sonar-scanner -Dsonar.projectKey=Mission
-Dsonar.projectName=Mission \
            -Dsonar.java.binaries=. '''
}
```

6. Build

- **Purpose:** Build the project and skip tests.
- **Steps:**

```
sh "mvn package -DskipTests=true"
```

7. Deploy Artifacts To Nexus

- **Purpose:** Deploy the built artifacts to Nexus repository.
- **Steps:**

```
withMaven(globalMavenSettingsConfig: 'maven-setting', jdk: 'jdk17',
maven: 'maven3', mavenSettingsConfig: '', traceability: true) {
    sh "mvn deploy -DskipTests=true"
}
```

8. Build & Tag Docker Image

- **Purpose:** Build and tag a Docker image.
- **Steps:**

```
withDockerRegistry(credentialsId: 'docker-cred', toolName: 'docker')
{
    sh "docker build -t adijaiswal/mission:latest ."
}
```

9. Trivy Scan Image

- **Purpose:** Perform a security scan on the Docker image.
- **Steps:**

```
sh "trivy image --format table -o trivy-image-report.html
adijaiswal/mission:latest"
```

10. Publish Docker Image

- **Purpose:** Push the Docker image to the Docker registry.
- **Steps:**

```
withDockerRegistry(credentialsId: 'docker-cred', toolName: 'docker')
{
    sh "docker push adijaiswal/mission:latest"
}
```

11. Deploy To Kubernetes (K8s)

- **Purpose:** Deploy the application to a Kubernetes cluster.
- **Steps:**

```
withKubeConfig(caCertificate: '', clusterName: 'DS-EKS', contextName: '',
credentialsId: 'k8-token', namespace: 'webapps', restrictKubeConfigAccess:
false, serverUrl: 'https://EA12CBD2F14726DD103E88821D89490F.gr7.ap-south-
1.eks.amazonaws.com') {
    sh "kubectl apply -f ds.yml -n webapps"
    sleep 60
}
```

12. Verify Deployment

- **Purpose:** Verify the deployment by checking the pods and services.
- **Steps:**

  ```
  withKubeConfig(caCertificate: '', clusterName: 'DS-EKS', contextName:
  '', credentialsId: 'k8-token', namespace: 'webapps',
  restrictKubeConfigAccess: false, serverUrl:
  'https://EA12CBD2F14726DD103E88821D89490F.gr7.ap-south-
  1.eks.amazonaws.com') {
      sh "kubectl get pods -n webapps"
      sh "kubectl get svc -n webapps"
  }
  ```

Post-Build Actions

Always

- **Purpose:** Send an email notification with the build status and attach the Trivy image scan report.
- **Steps:**

  ```
  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:
${pipelineStatus.toUpperCase()}</h3>
        </div>
       <p>Check the <a href="${BUILD_URL}">console output</a>.</p>
        </div>
        </body>
        </html>
    """

    emailext (
        subject: "${jobName} - Build ${buildNumber} -
${pipelineStatus.toUpperCase()}",
        body: body,
        to: 'jaiswaladi246@gmail.com',
        from: 'jenkins@example.com',
        replyTo: 'jenkins@example.com',
        mimeType: 'text/html',
        attachmentsPattern: 'trivy-image-report.html'
    )
}
```

PHASE-4 | Monitoring

Setup Prometheus,Grafana,node-exporter,blackbox-exporter

Prerequisites

- Linux-based system with `wget`, `tar`, and basic shell utilities installed.
- User with sudo privileges.

1. Install Prometheus

1. **Download Prometheus:**

   ```
   wget
   https://github.com/prometheus/prometheus/releases/download/v2.52.0/pr
   ometheus-2.52.0.linux-amd64.tar.gz
   ```

2. **Extract the Tarball:**

   ```
   tar -xzvf prometheus-2.52.0.linux-amd64.tar.gz
   ```

3. **Move to the Extracted Directory:**

   ```
   cd prometheus-2.52.0.linux-amd64
   ```

4. **Run Prometheus:**

   ```
   ./prometheus &
   ```

5. **Verify Prometheus is Running:**

   - Open a web browser and navigate to `http://localhost:9090`.

2. Install Node Exporter

1. **Download Node Exporter:**

   ```
   wget
   https://github.com/prometheus/node_exporter/releases/download/v1.8.1/
   node_exporter-1.8.1.linux-amd64.tar.gz
   ```

2. **Extract the Tarball:**

   ```
   tar -xzvf node_exporter-1.8.1.linux-amd64.tar.gz
   ```

3. **Move to the Extracted Directory:**

   ```
   cd node_exporter-1.8.1.linux-amd64
   ```

4. **Run Node Exporter:**

   ```
   ./node_exporter &
   ```

5. **Verify Node Exporter is Running:**

   - Open a web browser and navigate to `http://localhost:9100/metrics`.

3. Install Blackbox Exporter

1. **Download Blackbox Exporter:**

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

2. **Extract the Tarball:**

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

3. **Move to the Extracted Directory:**

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

4. **Run Blackbox Exporter:**

   ```
   ./blackbox_exporter &
   ```

5. **Verify Blackbox Exporter is Running:**

   - Open a web browser and navigate to `http://localhost:9115/metrics`.

Configuration

Prometheus Configuration

To scrape metrics from Node Exporter and Blackbox Exporter, you need to configure Prometheus.

1. **Edit the Prometheus Configuration File (`prometheus.yml`):**

```
global:
  scrape_interval: 15s
scrape_configs:
  - job_name: 'prometheus'
    static_configs:
      - targets: ['localhost:9090']

  - job_name: 'node_exporter'
    static_configs:
      - targets: ['localhost:9100']

  - job_name: 'blackbox_exporter'
    metrics_path: /probe
    params:
      module: [http_2xx]
    static_configs:
      - targets:
        - http://localhost:9115
    relabel_configs:
      - source_labels: [__address__]
        target_label: __param_target
      - source_labels: [__param_target]
        target_label: instance
      - target_label: __address__
        replacement: localhost:9115
```

2. **Restart Prometheus to Apply the Configuration:**

```
3. pkill prometheus
   ./prometheus &
```

Installation and Setup of Grafana

This guide will walk you through the steps to download, install, and set up Grafana on a Linux-based system.

Prerequisites

Ensure you have the following prerequisites installed on your system:

- `adduser`
- `libfontconfig1`
- `musl`

1. Install Prerequisites

1. **Update your package list:**

   ```
   sudo apt-get update
   ```

2. **Install necessary packages:**

   ```
   sudo apt-get install -y adduser libfontconfig1 musl
   ```

2. Download and Install Grafana

1. **Download the Grafana Enterprise package:**

   ```
   wget https://dl.grafana.com/enterprise/release/grafana-
   enterprise_11.0.0_amd64.deb
   ```
2. **Install Grafana using `dpkg`:**
   ```
   sudo dpkg -i grafana-enterprise_11.0.0_amd64.deb
   ```

3. Start and Enable Grafana

1. **Start the Grafana service:**

   ```
   sudo systemctl start grafana-server
   ```

2. **Enable the Grafana service to start on boot:**

   ```
   sudo systemctl enable grafana-server
   ```

4. Access Grafana

1. **Open a web browser and navigate to:**

```
2. http://localhost:3000
```

3. **Log in to Grafana:**
   - The default username is `admin`.
   - The default password is `admin`.
4. **Change the default password:**
   - Upon first login, you will be prompted to change the default password. Enter a new password and confirm it.

5. Configure Grafana

1. **Add a Data Source:**
   - Navigate to `Configuration` > `Data Sources`.
   - Click `Add data source`.
   - Choose your desired data source type (e.g., Prometheus).
   - Configure the data source with the appropriate URL (e.g., `http://localhost:9090` for Prometheus).
   - Click `Save & Test`.
2. **Create a Dashboard:**
   - Navigate to `Create` > `Dashboard`.
   - Add panels and configure queries to visualize your metrics.
   - Save the dashboard.