

CI/CD

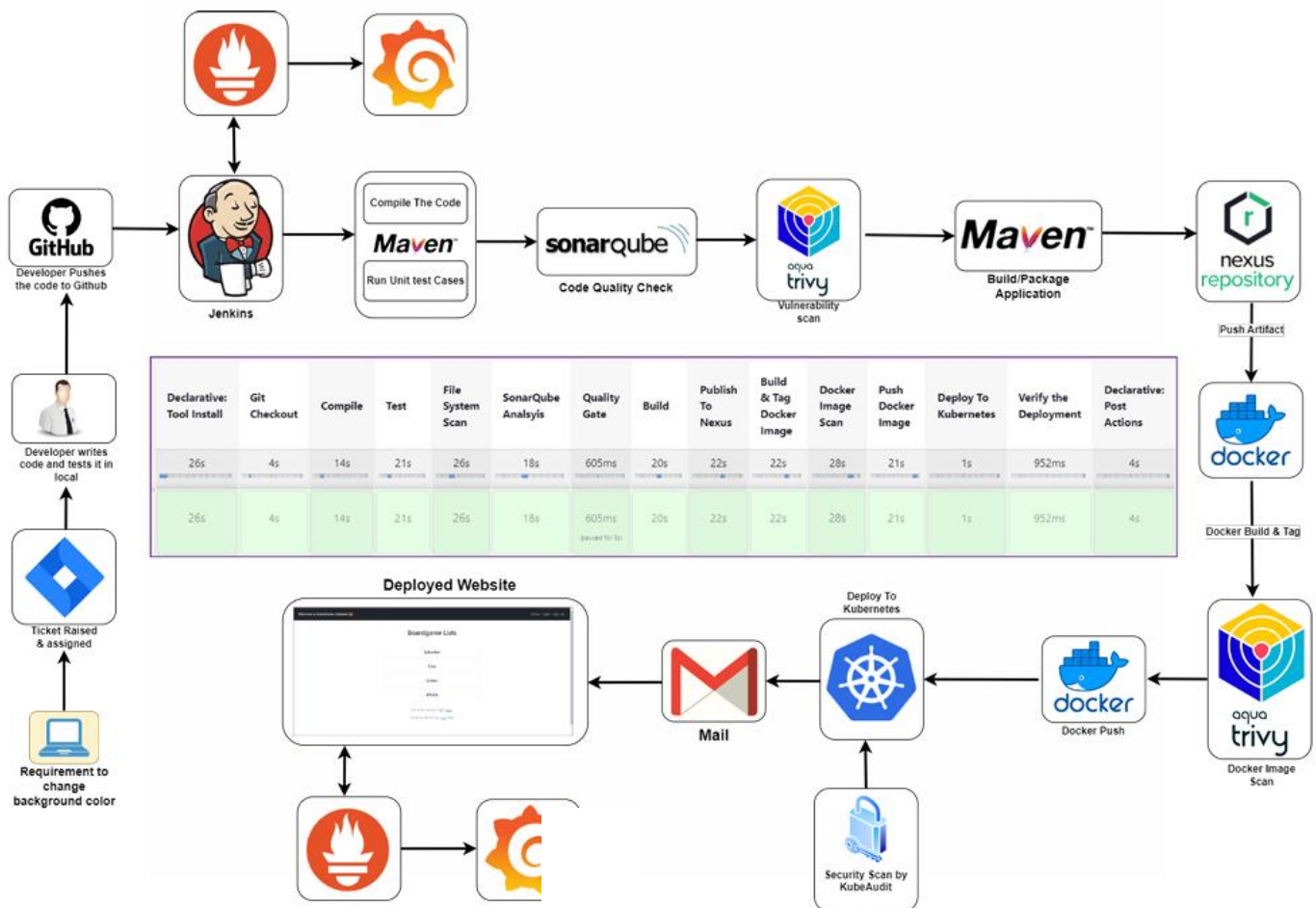
DEVOPS PIPELINE PROJECT

Deployment of Java Application on Kubernetes

Introduction

In the rapidly evolving landscape of software development, the adoption of DevOps practices has become imperative for organizations striving for agility, efficiency, and quality in their software delivery processes. The project at hand focuses on the implementation of a robust DevOps Continuous Integration/Continuous Deployment (CI/CD) pipeline, orchestrated by Jenkins, to streamline the development, testing, and deployment phases of a software product.

Architecture



Purpose and Objectives

The primary purpose of this project is to automate the software delivery lifecycle, from code compilation to deployment, thereby accelerating time-to-market, enhancing product quality, and reducing manual errors. The key objectives include:

- Establishing a seamless CI/CD pipeline using Jenkins to automate various stages of the software delivery process.
- Integrating essential DevOps tools such as Maven, SonarQube, Trivy, Nexus Repository, Docker, Kubernetes, Prometheus, and Grafana to ensure comprehensive automation and monitoring.
- Improving code quality through static code analysis and vulnerability scanning.
- Ensuring reliable and consistent deployments on a Kubernetes cluster with proper load balancing.
- Facilitating timely notifications and alerts via email integration for efficient communication and incident management.
- Implementing robust monitoring and alerting mechanisms to track system health and performance.

Tools Used

1. **Jenkins:** Automation orchestration for CI/CD pipeline.
2. **Maven:** Build automation and dependency management.
3. **SonarQube:** Static code analysis for quality assurance.
4. **Trivy:** Vulnerability scanning for Docker images.
5. **Nexus Repository:** Artifact management and version control.
6. **Docker:** Containerization for consistency and portability.
7. **Kubernetes:** Container orchestration for deployment.
8. **Gmail Integration:** Email notifications for pipeline status.
9. **Prometheus and Grafana:** Monitoring and visualization of system metrics.
10. **AWS :** Creating virtual machines .

SEGMENT 1 :

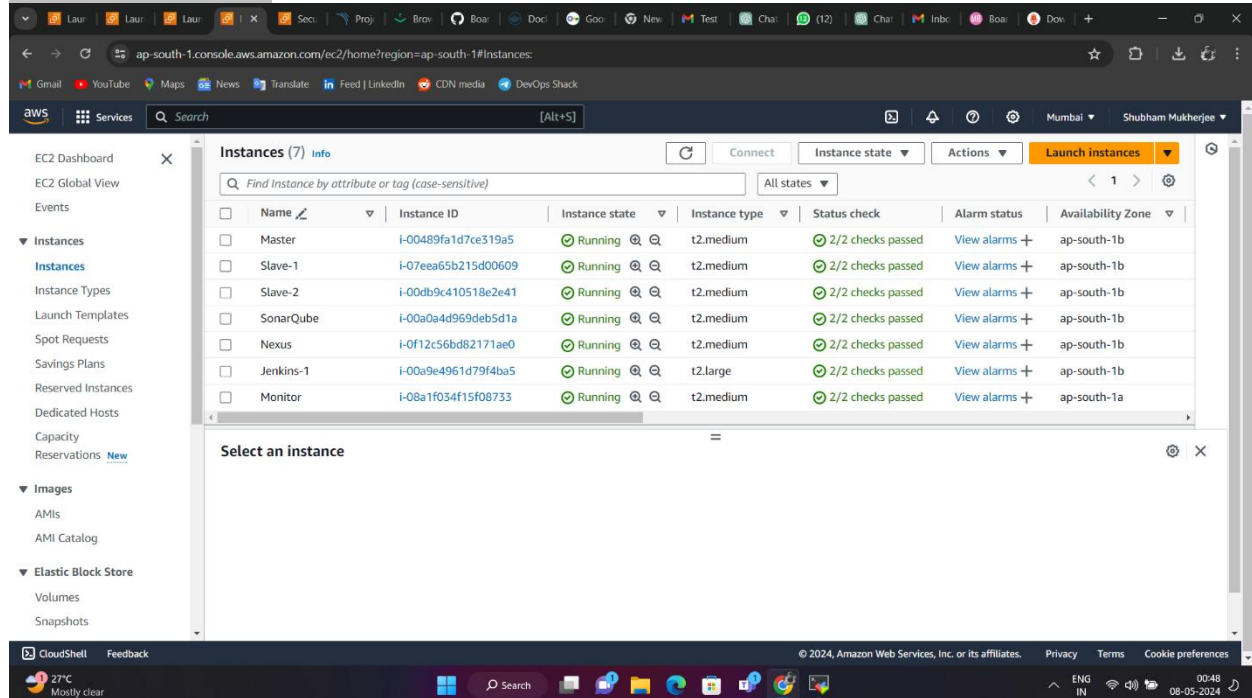
1.Setting up Virtual Machines on AWS

To establish the infrastructure required for the DevOps tools setup, virtual machines were provisioned on the Amazon Web Services (AWS) platform. Each virtual machine served a specific purpose in the CI/CD pipeline. Here's an overview of the virtual machines created for different tools:

1. **Kubernetes Master Node:** This virtual machine served as the master node in the Kubernetes cluster. It was responsible for managing the cluster's state, scheduling applications, and coordinating communication between cluster nodes.
2. **Kubernetes Worker Node 1 and Node 2:** These virtual machines acted as worker nodes in the Kubernetes cluster, hosting and running containerized applications. They executed tasks assigned by the master node and provided resources for application deployment and scaling.
3. **SonarQube Server:** A dedicated virtual machine hosted the SonarQube server, which performed static code analysis to ensure code quality and identify potential issues such as bugs, code smells, and security vulnerabilities.
4. **Nexus Repository Manager:** Another virtual machine hosted the Nexus Repository Manager, serving as a centralized repository for storing and managing build artifacts, Docker images, and other dependencies used in the CI/CD pipeline.
5. **Jenkins Server:** A virtual machine was allocated for the Jenkins server, which served as the central hub for orchestrating the CI/CD pipeline. Jenkins coordinated the execution of pipeline stages, triggered builds, and integrated with other DevOps tools for seamless automation.
6. **Monitoring Server (Prometheus and Grafana):** A single virtual machine hosted both Prometheus and Grafana for monitoring and visualization of system metrics. Prometheus collected metrics from various components of the CI/CD pipeline, while Grafana provided interactive dashboards for real-time monitoring and analysis.

Each virtual machine was configured with the necessary resources, including CPU, memory, and storage, to support the respective tool's functionalities and accommodate the workload demands of the CI/CD pipeline. Additionally, security measures such as access controls, network configurations, and encryption were implemented to safeguard the virtualized infrastructure and data integrity.

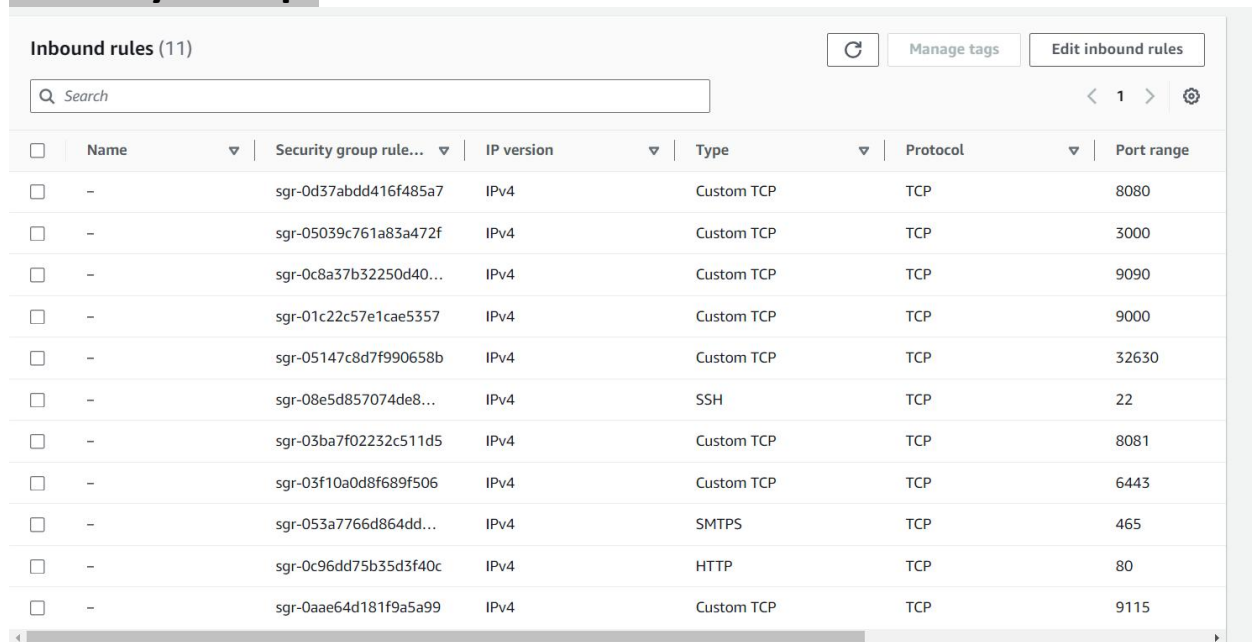
EC2 Instances :



The screenshot displays the AWS Management Console for the 'ap-south-1' region. The 'Instances' page shows a list of seven EC2 instances, all in a 'Running' state. The instances are: Master, Slave-1, Slave-2, SonarQube, Nexus, Jenkins-1, and Monitor. Each instance has a unique ID, is of type 't2.medium' or 't2.large', and has passed all status checks. The 'Availability Zone' for all instances is 'ap-south-1a'. The console also shows a sidebar with navigation options like 'EC2 Dashboard', 'Events', 'Instances', 'Instance Types', 'Launch Templates', 'Spot Requests', 'Savings Plans', 'Reserved Instances', 'Dedicated Hosts', 'Capacity', 'Reservations', 'Images', 'AMIs', 'AMI Catalog', 'Elastic Block Store', 'Volumes', and 'Snapshots'.

Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone
Master	i-00489fa1d7ce519a5	Running	t2.medium	2/2 checks passed	View alarms +	ap-south-1b
Slave-1	i-07eea65b215d00609	Running	t2.medium	2/2 checks passed	View alarms +	ap-south-1b
Slave-2	i-00db9c410518e2e41	Running	t2.medium	2/2 checks passed	View alarms +	ap-south-1b
SonarQube	i-00a0a4d969deb5d1a	Running	t2.medium	2/2 checks passed	View alarms +	ap-south-1b
Nexus	i-0f12c56bd82171ae0	Running	t2.medium	2/2 checks passed	View alarms +	ap-south-1b
Jenkins-1	i-00a9e4961d79f4ba5	Running	t2.large	2/2 checks passed	View alarms +	ap-south-1b
Monitor	i-08a1f034f15f08733	Running	t2.medium	2/2 checks passed	View alarms +	ap-south-1a

Security Group:



The screenshot displays the 'Inbound rules' page for a security group. It shows a list of 11 inbound rules, all of which are 'Custom TCP' rules. The rules are defined by their 'Security group rule ID', 'IP version' (all IPv4), 'Type' (all Custom TCP), 'Protocol' (all TCP), and 'Port range'. The rules are: sgr-0d37abdd416f485a7 (8080), sgr-05039c761a83a472f (3000), sgr-0c8a37b32250d40... (9090), sgr-01c22c57e1cae5357 (9000), sgr-05147c8d7f990658b (32630), sgr-08e5d857074de8... (22), sgr-03ba7f02232c511d5 (8081), sgr-03f10a0d8f689f506 (6443), sgr-053a7766d864dd... (465), sgr-0c96dd75b35d3f40c (80), and sgr-0aae64d181f9a5a99 (9115).

Name	Security group rule...	IP version	Type	Protocol	Port range
-	sgr-0d37abdd416f485a7	IPv4	Custom TCP	TCP	8080
-	sgr-05039c761a83a472f	IPv4	Custom TCP	TCP	3000
-	sgr-0c8a37b32250d40...	IPv4	Custom TCP	TCP	9090
-	sgr-01c22c57e1cae5357	IPv4	Custom TCP	TCP	9000
-	sgr-05147c8d7f990658b	IPv4	Custom TCP	TCP	32630
-	sgr-08e5d857074de8...	IPv4	SSH	TCP	22
-	sgr-03ba7f02232c511d5	IPv4	Custom TCP	TCP	8081
-	sgr-03f10a0d8f689f506	IPv4	Custom TCP	TCP	6443
-	sgr-053a7766d864dd...	IPv4	SMTPS	TCP	465
-	sgr-0c96dd75b35d3f40c	IPv4	HTTP	TCP	80
-	sgr-0aae64d181f9a5a99	IPv4	Custom TCP	TCP	9115

2. Setup K8-Cluster using Kubeadm

This guide outlines the steps needed to set up a Kubernetes cluster using kubeadm.

Pre-requisites

- Ubuntu OS (Xenial or later)
- sudo privileges
- Internet access
- t2.medium instance type or higher

AWS Setup

- Make sure your all instance are in same **Security group**.
- Expose port **6443** in the **Security group**, so that worker nodes can join the cluster.

Execute on Both "Master" & "Worker Node"

Run the following commands on both the master and worker nodes to prepare them for kubeadm.

```
# disable swap
sudo swapoff -a
```

```
# Create the .conf file to load the modules at bootup
cat <<EOF | sudo tee /etc/modules-load.d/k8s.conf
overlay
br_netfilter
EOF
```

```
sudo modprobe overlay
sudo modprobe br_netfilter
```

```
# sysctl params required by setup, params persist across reboots
cat <<EOF | sudo tee /etc/sysctl.d/k8s.conf
net.bridge.bridge-nf-call-iptables = 1
net.bridge.bridge-nf-call-ip6tables = 1
net.ipv4.ip_forward = 1
EOF
```

```
# Apply sysctl params without reboot
```

```
sudo systemctl --system
```

```
## Install CRIO Runtime
```

```
sudo apt-get update -y  
sudo apt-get install -y software-properties-common curl apt-transport-https ca-  
certificates gpg
```

```
sudo curl -fsSL https://pkgs.k8s.io/addons:/cri-  
o:/prerelease:/main/deb/Release.key | sudo gpg --dearmor -o  
/etc/apt/keyrings/cri-o-apt-keyring.gpg  
echo "deb [signed-by=/etc/apt/keyrings/cri-o-apt-keyring.gpg]  
https://pkgs.k8s.io/addons:/cri-o:/prerelease:/main/deb/ /" | sudo tee  
/etc/apt/sources.list.d/cri-o.list
```

```
sudo apt-get update -y  
sudo apt-get install -y cri-o
```

```
sudo systemctl daemon-reload  
sudo systemctl enable crio --now  
sudo systemctl start crio.service
```

```
echo "CRI runtime installed successfully"
```

```
# Add Kubernetes APT repository and install required packages  
curl -fsSL https://pkgs.k8s.io/core:/stable:/v1.29/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:/v1.29/deb/ /' | sudo tee  
/etc/apt/sources.list.d/kubernetes.list
```

```
sudo apt-get update -y  
sudo apt-get install -y kubelet="1.29.0-*" kubectcl="1.29.0-*" kubeadm="1.29.0-*"  
sudo apt-get update -y  
sudo apt-get install -y jq
```

```
sudo systemctl enable --now kubelet  
sudo systemctl start kubelet
```

Execute ONLY on "Master Node"

```
sudo kubeadm config images pull
```

```
sudo kubeadm init
```

```
mkdir -p "$HOME"/.kube
```

```
sudo cp -i /etc/kubernetes/admin.conf "$HOME"/.kube/config
```

```
sudo chown "$(id -u)": "$(id -g)" "$HOME"/.kube/config
```

```
# Network Plugin = calico
```

```
kubectl apply -f
```

```
https://raw.githubusercontent.com/projectcalico/calico/v3.26.0/manifests/calico.yaml
```

```
kubeadm token create --print-join-command
```

- You will get kubeadm token, **Copy it.**

```

MINGW64/c/Users/Faizan/Desktop/New folder
customresourcedefinition.apiextensions.k8s.io/kubecontrollersconfigurations.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/networkpolicies.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/networksets.crd.projectcalico.org created
clusterrole.rbac.authorization.k8s.io/calico-kube-controllers created
clusterrole.rbac.authorization.k8s.io/calico-node created
clusterrole.rbac.authorization.k8s.io/calico-cni-plugin created
clusterrolebinding.rbac.authorization.k8s.io/calico-kube-controllers created
clusterrolebinding.rbac.authorization.k8s.io/calico-node created
clusterrolebinding.rbac.authorization.k8s.io/calico-cni-plugin created
daemonset.apps/calico-node created
deployment.apps/calico-kube-controllers created
kubeadm join 172.31.62.216:6443 --token br7fe5.hq28adbmn1mu17ky --discovery-token-ca-cert-hash sha256:2bc469a8d14fbeb0f879328d2b416fad32b29a8505d3f448b98703fff3b014d9
ubuntu@ip-172-31-62-216:~/kubernetes_cluster_with_kubeadm/aws-ec2$ kubectl get nodes
NAME                                STATUS    ROLES    AGE    VERSION
ip-172-31-62-216                    Ready    control-plane   21s    v1.29.0
ubuntu@ip-172-31-62-216:~/kubernetes_cluster_with_kubeadm/aws-ec2$ kubectl get nodes
NAME                                STATUS    ROLES    AGE    VERSION
ip-172-31-20-103                    Ready    <none>      4s    v1.29.0
ip-172-31-62-216                    Ready    control-plane   105s   v1.29.0
ubuntu@ip-172-31-62-216:~/kubernetes_cluster_with_kubeadm/aws-ec2$ git clone https://github.com/faizan35/scripts_For_Help.git^C
ubuntu@ip-172-31-62-216:~/kubernetes_cluster_with_kubeadm/aws-ec2$ cd ~
ubuntu@ip-172-31-62-216:~$ git clone https://github.com/faizan35/scripts_For_Help.git
Cloning into 'scripts_For_Help'...
remote: Enumerating objects: 437, done.
remote: Counting objects: 100% (130/130), done.
remote: Compressing objects: 100% (94/94), done.
remote: Total 437 (delta 56), reused 99 (delta 30), pack-reused 307

```


Execute on ALL of your Worker Node's

1. Perform pre-flight checks
`sudo kubeadm reset pre-flight checks`
2. Paste the join command you got from the master node and append `--v=5` at the end.

```
sudo your-token --v=5
```

Use sudo before the token.

Verify Cluster Connection

On Master Node:

```
kubectll get nodes
```

```
ubuntu@ip-172-31-17-101:~$ kubectll get nodes
NAME                STATUS    ROLES    AGE   VERSION
ip-172-31-17-101    Ready    control-plane  12m   v1.29.0
ip-172-31-17-255    Ready    <none>      8m27s v1.29.0
ubuntu@ip-172-31-17-101:~$
```


3. Installing Jenkins on Ubuntu

Execute these commands on Jenkins Server

```
#!/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.

KUBECTL

```
curl -o kubectl https://amazon-eks.s3.us-west-2.amazonaws.com/1.19.6/2021-01-05/bin/linux/amd64/kubectl
chmod +x ./kubectl
sudo mv ./kubectl /usr/local/bin
kubectl version --short --client
```

4. Install docker for future use

Execute these commands on Jenkins, SonarQube and Nexus Servers

```
#!/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
```

5. SetUp Nexus

Execute these commands on Nexues VM

```
#!/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.

6. SetUp SonarQube

Execute these commands on SonarQube VM

```
#!/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:its-community
```

This command will download the sonarqube:its-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>.

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.

SEGMENT-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:

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

2. Generate a Personal Access Token:

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

3. Clone the Repository Locally:

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

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

5. Add Your Source Code Files:

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

6. Stage and Commit Changes:

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

`git add .`

- o 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:

- o Use the git push command to push your committed changes to the remote repository:

`git push`

- o 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:

o 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.

GIT REPOSITORY - <https://github.com/Shubham-Stunner/BoardGame.git>

SEGMENT-3 | CICD

Install below Plugins in Jenkins

1. Eclipse Temurin Installer:

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

2. Pipeline Maven Integration:

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

3. Config File Provider:

- o This plugin allows you to define configuration files (e.g., properties, XML, JSON) centrally in Jenkins.
- o These configurations can then be referenced and used by your Jenkins jobs.
- o Install it using the same procedure as mentioned earlier.

4. SonarQube Scanner:

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

5. Kubernetes CLI:

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

- o Install it through the plugin manager.

6. Kubernetes:

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

7. Docker:

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

8. Docker Pipeline Step:

- o This plugin extends Jenkins Pipeline with steps to build, publish, and run Docker containers as part of your Pipeline scripts.
- o It provides a convenient way to manage Docker containers directly from Jenkins Pipelines.
- o 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.

Jenkins Pipeline

Create a new Pipeline job .

```
pipeline {
  agent any

  environment {
    SCANNER_HOME = tool 'sonar-scanner'
  }

  tools {
    jdk 'jdk17'
    maven 'maven3'
  }

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

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

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

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

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

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

stage('Docker Image Scan') {
    steps {
        script {
            withDockerRegistry(credentialsId: 'docker-cred', toolName: 'docker') {
                sh "trivy image --format table -o trivy-image-report.html
stunnershubham/boardgame:latest"
            }
        }
    }
}
```

```

    }
  }
  stage('Push Docker Image') {
    steps {
      script {
        withDockerRegistry(credentialsId: 'docker-cred', toolName: 'docker') {
          sh "docker push stunnershubham/boardgame:latest"
        }
      }
    }
  }
  stage('Deploy to Kubernetes') {
    steps {
      withKubeConfig(caCertificate: '', clusterName: 'kubernetes', contextName: '', credentialsId: 'k8-cred', namespace: 'webapps', restrictKubeConfigAccess: false, serverUrl: 'https://172.31.8.22:6443') {
        sh "kubectl apply -f deployment-service.yaml"
        sh "kubectl get pods -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>
      """

      emailx(
        subject: "${jobName} - Build ${buildNumber} - ${pipelineStatus.toUpperCase()}",

```

DevOps Pipeline

```
body: body,  
  to: 'shubhammukherji654@gmail.com',  
  from: 'jenkins@example.com',  
  replyTo: 'jenkins@example.com',  
  mimeType: 'text/html',  
  attachmentsPattern: 'trivy-image-report.html'  
)  
}  
}  
}
```


SEGMENT-4 | Monitoring

Prometheus

Links to download Prometheus, Node_Exporter & black Box exporter

<https://prometheus.io/download/>

Extract and Run Prometheus

After downloading Promethous extract the .tar file

Now Cd into the extracted file and and run

```
./prometheus &
```

By default Prometheus runs on Port 9090 and access it using your instance

```
<IP address>:9090
```

Similarly download and run Blackbox exporter.

```
./ backbox_exporter &
```

Grafana

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

OR

Run This code on Monitoring VM to Install Grafana

```
sudo apt-get install -y adduser libfontconfig1 musl
wget https://dl.grafana.com/enterprise/release/grafana-enterprise_10.4.2_amd64.deb
sudo dpkg -i grafana-enterprise_10.4.2_amd64.deb
```

once Installed run

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

by default Grafana runs on port 3000 so access it using instance <IPAddress>:3000

Configure Prometheus

Go inside the `Prometheus.yaml` file and edit it

```
scrape_configs:
- 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.
      - https://prometheus.io # Target to probe with https.
      - http://example.com:8080 # Target to probe with http on port 8080.
  relabel_configs:
    - source_labels: [__address__]
      target_label: __param_target
    - source_labels: [__param_target]
      target_label: instance
    - target_label: __address__
      replacement:<IP address>:9115
```

Replace the IP address with your instance IP address.

After this Restart Prometheus using this command

```
pgrep Prometheus
```

Once you run the above command you will get the Id of Prometheus then use the id and kill it

```
kill <ID>
```

Add Prometheus as Data sources inside Grafana

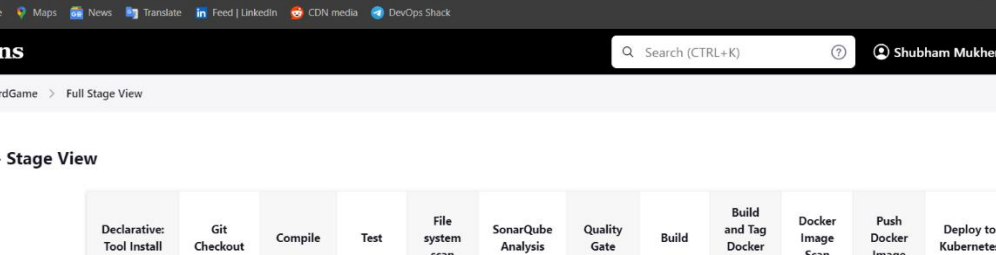
Go to Prometheus server > Data Sources

> Prometheus add IPaddress of Prometheus

> Import Dashboard form web .

Results :

JENKINS PIPELINE



The screenshot displays the Jenkins web interface for a pipeline named 'BoardGame'. The top navigation bar includes the Jenkins logo, a search bar, and the user 'Shubham Mukherjee'. The breadcrumb trail shows 'Dashboard > BoardGame > Full Stage View'.

BoardGame - Stage View

Average stage times:
(Average full run time: ~3min 0s)

	Declarative: Tool Install	Git Checkout	Compile	Test	File system scan	SonarQube Analysis	Quality Gate	Build	Build and Tag Docker Image	Docker Image Scan	Push Docker Image	Deploy to Kubernetes	Declarative: Post Actions
71 May 07 23:47 No Changes	160ms	1s	3s	16s	2s	14s	334ms	17s	18s	1min 9s	19s	2s	10s
66 May 07 23:46 No Changes	160ms	1s	3s	16s	2s	14s	334ms (skipped for 1s)	17s	18s	1min 9s	19s	2s	10s

The bottom of the screenshot shows the system tray with weather information (26°C Mostly clear), taskbar icons, and system clock (08:04, 08-05-2024).

PROMETHEUS

Prometheus

AlertsGraphStatusHelp

Targets

All scrape poolsUnhealthyCollapse All

Q

filter by endpoint or labels

UnknownUnhealthyHealthy

blackbox (2/2 up)

show all

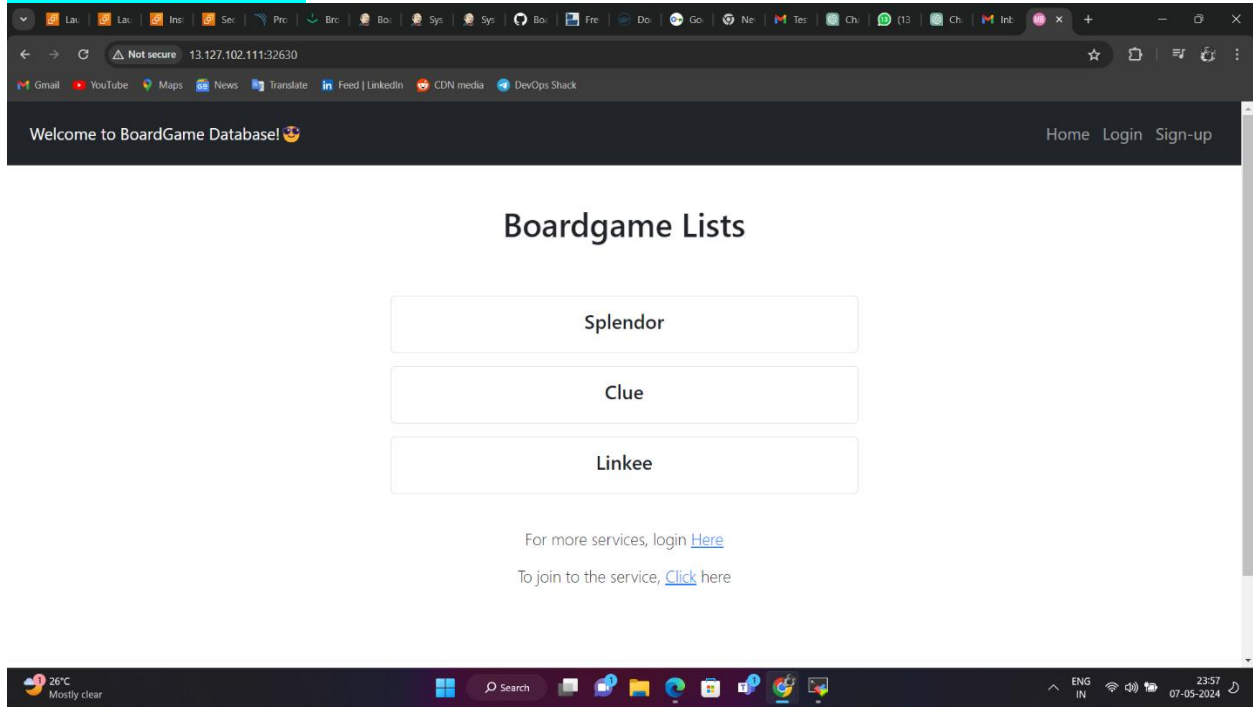
Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
<div>http://13.201.115.179:9115/probe</div> <div>instance="http://prometheus.jp"job="blackbox"</div>	UP	instance="http://prometheus.jp"job="blackbox"	2.591s ago	336.470ms	
<div>http://13.201.115.179:9115/probe</div> <div>instance="http.jp"target="http://13.127.102.111:32638"</div>	UP	instance="http://13.127.102.111:32638"job="blackbox"	12.480s ago	9.502s	

prometheus (1/1 up)

show all

Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
<div>http://localhost:9090/metrics</div>	UP	instance="localhost:9090"job="prometheus"	10.808s ago	4.795ms	

APPLICATION



Conclusion

In conclusion, the successful implementation of the DevOps CI/CD pipeline project marks a significant milestone in enhancing the efficiency, reliability, and quality of software delivery processes. By automating key aspects of the software development lifecycle, including compilation, testing, deployment, and monitoring, the project has enabled rapid and consistent delivery of software releases, contributing to improved time-to-market and customer satisfaction.

Acknowledgment of Contributions:

I would like to extend my gratitude to DevOps shack for helping me achieving my goals and objectives.

Final Thoughts

Looking ahead, the project's impact extends beyond its immediate benefits, paving the way for continuous improvement and innovation in software development practices. By embracing DevOps principles and leveraging cutting-edge tools and technologies, we have laid a solid foundation for future projects to build upon. The scalability, flexibility, and resilience of the CI/CD pipeline ensure its adaptability to evolving requirements and technological advancements, positioning our organization for long-term success in a competitive market landscape.

References

1. Jenkins Documentation:
<https://www.jenkins.io/doc/>
2. Maven Documentation:
<https://maven.apache.org/guides/index.html>
3. SonarQube Documentation:
<https://docs.sonarqube.org/latest/>
4. Trivy Documentation:
<https://github.com/aquasecurity/trivy>
5. Nexus Repository Manager Documentation:
<https://help.sonatype.com/repomanager3>
6. Docker Documentation: <https://docs.docker.com/>
7. Kubernetes Documentation:
<https://kubernetes.io/docs/>
8. Prometheus Documentation:
<https://prometheus.io/docs/>
9. Grafana Documentation:
<https://grafana.com/docs/>

These resources provided valuable insights, guidance, and support throughout the project lifecycle, enabling us to achieve our goals effectively.