CI/CD Pipeline for Boardgame

Phase 1 - Where we will Set up whole insfractructure.

Network - { Private , Isolated , Secure }
K8s - Deploy Application [Scan for vulnerabilities and issues].
VMs - { Sonar, Nexus , Jenkins and Monitoring}

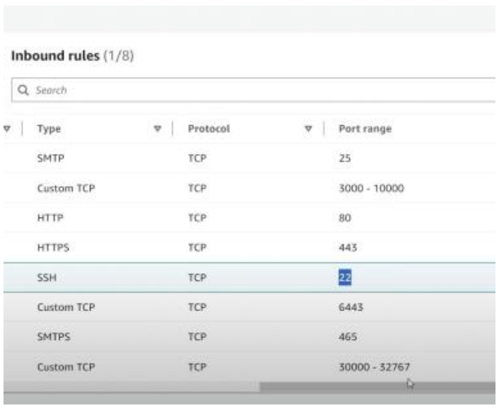
Phase 2 -
Create git repository [Private]
Push the Source Code
Visible in Repo

Phase 3 -
Working with CI/CD Pipeline
Best Practices
Security Measures
Mail Notification [To check status of our Pipeline Failed or Pass.]

Phase 4-

Setup monitor tools for monitoring applications.
System Level [CPU , RAM]
Website Level [Website Level (Traffic)]

So we gonna user default private VPC of AWS Cloud.
Ports which needs to be opened for our resources through security groups.



Inbound rules (1/8)

Type	Protocol	Port range
SMTP	TCP	25
Custom TCP	TCP	3000 - 10000
HTTP	TCP	80
HTTPS	TCP	443
SSH	TCP	22
Custom TCP	TCP	6443
SMTPS	TCP	465
Custom TCP	TCP	30000 - 32767

To open these ports in Ubuntu linux terminal.

```
#!/bin/bash

# First, ensure UFW is installed
sudo apt update
sudo apt install ufw -y

# Disable UFW to prevent any connection issues while configuring
sudo ufw disable

# Set default policies
sudo ufw default deny incoming
sudo ufw default allow outgoing

# Allow SSH (port 22) first to prevent lockout
sudo ufw allow 22/tcp

# Allow standard web ports
sudo ufw allow 80/tcp       # HTTP
sudo ufw allow 443/tcp      # HTTPS

# Allow Kubernetes API Server
sudo ufw allow 6443/tcp     # Kubernetes API server port

# Allow NodePort Services range
sudo ufw allow 30000:32767/tcp  # Kubernetes NodePort Services

# Allow email ports
sudo ufw allow 25/tcp       # SMTP
sudo ufw allow 465/tcp      # SMTPS

# Allow custom port range
sudo ufw allow 3000:10000/tcp   # Custom application ports

# Enable UFW
sudo ufw enable

# Display status
sudo ufw status numbered
```

30000 -32767 - For kubernetes cluster and deployment of applications.
465 - To send mails notification from jenkins pipeline to mails.
6443 - To setup kubernetes cluster.
22 - To access virtual machines.
443 - HTTPS
80 - HTTP
3000 - 10000 - To deploy the applications easily.
25 - For SMTP Server to send gmail notifications some companies prefer port 25.

First we will Setup three VM's for our Kubernetes Cluster.

So we created 3 VM's of ubuntu with 25 GB EBS.

Master
Slave -1
Slave -2

Now we gonna access these using Mobaxterm.

Make sure to set settings configuration ssh keepalive on.

Now after making connection with all the three servers and naming them master slave-1 slave-2 switch into root user in all three servers.

Now we will Install Kubernetes Cluster on all the Servers.

With the help of groups makings scripts we will run these commands.

Setup K8-Cluster using kubeadm [K8 Version-->1.28.1]

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

```
sudo apt-get update
```

2. Install Docker[On Master & Worker Node]

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

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

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

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

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

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

```
sudo apt update
```

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

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

Till here on all three servers Now only on Master Nodes.

7. Initialize Kubernetes Master Node [On MasterNode]

This will make master node create a command by which we can connect to the slaves machines and make them as worker nodes on which our deployments will be done.

So only for Master Server I switched to my desktop ubuntu as it requires higher configuration machine like 4gb ram and 2 core cpu's.

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

8. Configure Kubernetes Cluster [On MasterNode]

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

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

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

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

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

From <https://github.com/jaiswaladi246/DevOps_Shack_Ultimate_Pipeline_12_march/blob/main/PHASE-1/2.%20K8-Setup.md>

Command to run on Slave Nodes to connect them with the master node as worker nodes -

```
kubeadm join 172.31.15.190:6443 --token xey5cd.9s9n62dooa1c9yjv \
        --discovery-token-ca-cert-hash
sha256:f557212917ad12c10efb1d3d4d7fa4f5d2be6417339beb28fa914cd8b36710c1

kubeadm join 172.31.15.190:6443 --token ztgide.scwp4knxd6mf0ypf --discovery-token-ca-cert-hash
sha256:f557212917ad12c10efb1d3d4d7fa4f5d2be6417339beb28fa914cd8b36710c1
kubeadm token create --print-join-command
```

Now run this on both the nodes so that they can join the cluster as worker nodes.

Troubleshooting -
The error indicates that your Container Runtime Interface (CRI) is not properly configured. Let's resolve this by setting up containerd:

1. Remove existing configurations:

Bash
Copy

```
sudo kubeadm reset -f
sudo apt-mark unhold kubelet kubeadm kubectl
sudo apt-get remove -y kubelet kubeadm kubectl
sudo rm -rf /etc/kubernetes/*
sudo rm -rf ~/.kube
```

1. Install containerd:

bash
Copy

```
sudo apt-get update
sudo apt-get install -y containerd
sudo mkdir -p /etc/containerd
sudo containerd config default | sudo tee /etc/containerd/config.toml
sudo sed -i 's/SystemdCgroup = false/SystemdCgroup = true/' /etc/containerd/config.toml
```

1. Restart and enable containerd:

bash

```
sudo systemctl restart containerd
sudo systemctl enable containerd
```

1. Reinstall Kubernetes components:

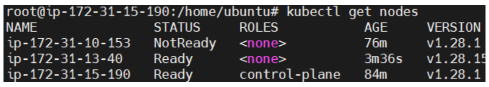
bash

```
sudo apt-get update
sudo apt-get install -y apt-transport-https ca-certificates curl
curl -fsSL https://pkgs.k8s.io/core:/stable:/v1.28/deb/Release.key | sudo gpg --dearmor -o
/etc/apt/keyrings/kubernetes-apt-keyring.gpg
echo 'deb [signed-by=/etc/apt/keyrings/kubernetes-apt-keyring.gpg]
https://pkgs.k8s.io/core:/stable:/v1.28/deb/ /' | sudo tee /etc/apt/sources.list.d/kubernetes.list
sudo apt-get update
sudo apt-get install -y kubelet kubeadm kubectl
sudo apt-mark hold kubelet kubeadm kubectl
```

1. Initialize the cluster:

bash

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



```
root@ip-172-31-15-190:/home/ubuntu# kubectl get nodes
NAME               STATUS     ROLES           AGE     VERSION
ip-172-31-10-153   NotReady   <none>          76m     v1.28.1
ip-172-31-13-40    Ready      <none>          3m36s   v1.28.15
ip-172-31-15-190   Ready      control-plane   84m     v1.28.1
```

So now both the worker nodes are connected to the cluster master node.

Now as if we have setup the kubernetes cluster we can scan it for any kind of issues.

Kubeaudit is a tool that can be used for scanning the kubernetes cluster.

To download the kubeedit on master node.

```
wget https://github.com/Shopify/kubeaudit/releases/download/v0.22.2/kubeaudit_0.22.2_linux_amd64.tar.gz
```

Extract the file.

```
tar -xvzf kubeaudit_0.22.2_linux_amd64.tar.gz
mv kubeaudit /usr/local/bin/
```

To Scan the whole Cluster.

```
kubeaudit all
```

Now we will create some virtual machines where we can confiugure sonar , nexus , jenkins and monitoring.

2 Instances for SonarQube Server and Nexus Server.
ubuntu linux
t2.medium
20 gb gpg2

Now will create vm for Jenkins.
t2.large
30gb

If we want to use resources within the VM from its repositories we need to update so they are available for the usage.

To resolve the SSH problem of again and again with the mobaxterm I have created elastic ip's for each servers.

Now, we will setup SonarQube and Nginx.

Install docker on both the Servers.

Using Script -

```
#!/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
```

Command to give the root users permissions also.

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

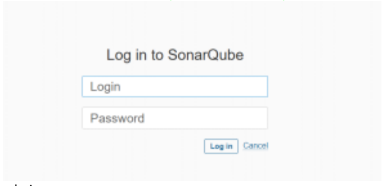
Just to check if docker is working or not.

```
docker pull hello-world
```

Now docker is installed on both servers now we will make docker conatiner on both the servers Nexus and SonarQube.

To create the conatiner in detached mode on vm port 9000 and conatiner port 9000 using sonarqube image by downloading it.

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


Log in to SonarQube
Login
Password
Log in Cancel

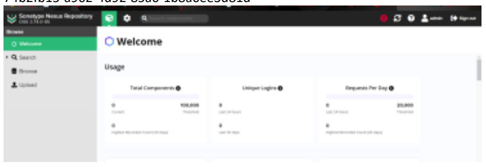
admin
admin

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

Now to login in Nexus we need to get the password by entering inside of the container bash shell.
Get the password.

```
docker exec -it 12d47291d678 /bin/bash
```

74b2fb15-a962-4d92-85a6-1b8a6ee3d81d



Now we will setup Jenkins -

As we know prerequisties for jenkins is java so we will install the java first.

```
apt install openjdk-17-jre-headless

sudo wget -O /usr/share/keyrings/jenkins-keyring.asc \
  https://pkg.jenkins.io/debian-stable/jenkins.io-2023.key
echo "deb [signed-by=/usr/share/keyrings/jenkins-keyring.asc]" \
  https://pkg.jenkins.io/debian-stable binary/ | sudo tee \
  /etc/apt/sources.list.d/jenkins.list > /dev/null
```

```
sudo apt-get update
sudo apt-get install jenkins
```

Also install dockers using that docker script from nexus or sonar server and run the below command to give the permission to other users other than root.

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

Now as Jenkins has been setup too we have completed the phase 1 now we have to configure phase 2.

In this we have to create private git repo and push the code in it and make it visible.

We have to upload files from pc to git repository using git bash.

As now we pushed all the files to the git repository we have done the phase 2.

Now we need to install some plugins before configureing CI/CD.

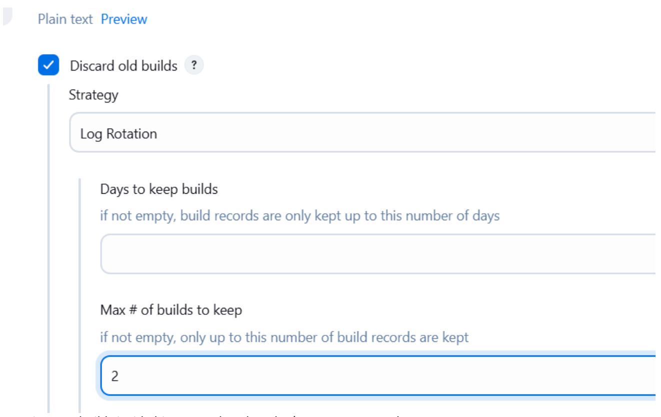
Install Plugins in Jenkins

1. **Eclipse Temurin Installer**:
   - This plugin enables Jenkins to automatically install and configure the Eclipse Temurin JDK (formerly known as AdoptOpenJDK).
   - To install, go to Jenkins dashboard -> Manage Jenkins -> Manage Plugins -> Available tab.
   - Search for "Eclipse Temurin Installer" and select it.
   - Click on the "Install without restart" button.
2. **Pipeline Maven Integration**:
   - This plugin provides Maven support for Jenkins Pipeline.
   - It allows you to use Maven commands directly within your Jenkins Pipeline scripts.
   - To install, follow the same steps as above, but search for "Pipeline Maven Integration" instead.
3. **Config File Provider**:
   - This plugin allows you to define configuration files (e.g., properties, XML, JSON) centrally in Jenkins.
   - These configurations can then be referenced and used by your Jenkins jobs.
   - Install it using the same procedure as mentioned earlier.
4. **SonarQube Scanner**:
   - SonarQube is a code quality and security analysis tool.
   - This plugin integrates Jenkins with SonarQube by providing a scanner that analyzes code during builds.
   - You can install it from the Jenkins plugin manager as described above.
5. **Kubernetes CLI**:
   - This plugin allows Jenkins to interact with Kubernetes clusters using the Kubernetes command-line tool (kubectl).
   - It's useful for tasks like deploying applications to Kubernetes from Jenkins jobs.
   - Install it through the plugin manager.
6. **Kubernetes**:
   - This plugin integrates Jenkins with Kubernetes by allowing Jenkins agents to run as pods within a Kubernetes cluster.
   - It provides dynamic scaling and resource optimization capabilities for Jenkins builds.
   - Install it from the Jenkins plugin manager.
7. **Docker**:
   - This plugin allows Jenkins to interact with Docker, enabling Docker builds and integration with Docker registries.
   - You can use it to build Docker images, run Docker containers, and push/pull images from Docker registries.
   - Install it from the plugin manager.
8. **Docker Pipeline Step**:
   - This plugin extends Jenkins Pipeline with steps to build, publish, and run Docker containers as part of your Pipeline scripts.
   - It provides a convenient way to manage Docker containers directly from Jenkins Pipelines.
   - Install it through the plugin manager like the others.

As we have installed the plugins now we have to configure these plugins.

JDK
Name
jdk17
Install automatically
Install from adoptium.net
Version
jdk-17.0.9+9
Add Installer

Now we will create the pipeline itself.


Plain text Preview

☑ Discard old builds ?

Strategy

Log Rotation

Days to keep builds

if not empty, build records are only kept up to this number of days

Max # of builds to keep

if not empty, only up to this number of build records are kept

2

Keeping two builds inside history so that they don't occupy too much space.

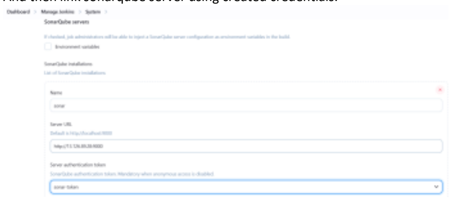
Now we writing jenkins file and also configuring credentials.

Installing trivy on jenkins to check if there is any kind of sensitive data stored.

```
sudo apt-get update
wget https://github.com/aquasecurity/trivy/releases/download/v0.48.3/trivy_0.48.3_Linux-64bit.deb
sudo dpkg -i trivy_0.48.3_Linux-64bit.deb
```

In pipeline we are configuing sonarqube tool we have to configure sonarqube server also.

First configure soarqube server credentials using secret text.
And then link sonarqube server using created credentials.


Dashboard > Manage Jenkins > System >

SonarQube servers

[illegible]

☐ Environment variables

SonarQube installations

List of SonarQube installations

Name

sonar

Server URL

Default is http://localhost:9000

[illegible]

Server authentication token

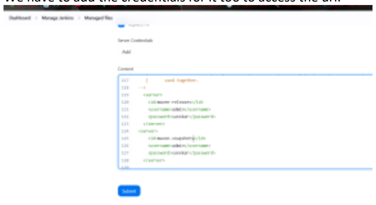
SonarQube authentication token. Mandatory when anonymous access is disabled.

sonar-token

Also creating quality gates which check on basis of conditions by making it from sonar qube administration.
Then build the image in pipeline than publishing artifacts to nexus.

http://43.204.7.145:8081/repository/maven-releases/

We have to add the credentials for it too to access the url.


Dashboard > Manage Jenkins > Managed files

Server Credentials

Add

Content

```
    |      used together.
  -->
   <server>
     <id>maven-releases</id>
     [illegible]
     [illegible]
   </server>
  <server>
     <id>maven-snapshots</id>
     [illegible]
     [illegible]
   </server>
```

Submit

Now to deploy the application in a proper way after pushing the image to the repository we can create a service account or we can use RBAC.

So first we will create an Service account (Jenkins) on Master Node in Yaml file.

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

Creating namespace for this service account as webapps.

```
kubectl create ns webapps
```

Now execute service account yaml to create service account.

```
kubectl apply -f svc.yaml
```

Now will create role which will have all the necessary complete access for deployment.
Creating role.yaml file for this.

```
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"]
```

Execute

```
kubectl apply -f role.yaml
```

Now we assign this role to the service account.
Creating bind.yaml file for this.

```
kubectl apply -f bind.yaml
```

Now we want the jenkins able to connect the kubernetes cluster for this we will create authentication token.

```
apiVersion: v1
kind: Secret
type: kubernetes.io/service-account-token
metadata:
  name: mysecretname
  annotations:
    kubernetes.io/service-account.name: jenkins
```

```
kubectl apply -f sec.yaml -n webapps
```

```
kubectl describe secret mysecretname -n webapps
```

Now we will use this token to authenticate the jenkins to the kubernetes by configuring it into the jenkins.

Now we will create pipeline syntax for kubernetes for this we need to get kubernetes end point.

```
cd ~/.kube
```

```
cat config
```

Endpoint

https://172.31.15.190:6443

Manifest file will also require to put in the Deployment to kubernetes part of jenkins and change the own docker image into the manifest file of kubernetes.

Here we are using load balancer concept for the communication of Pods.
Now we need to install kubectl on jenkins too.

Run this on Jenkins Server to install it.

```
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
```

Now to check if deployments is done.(In Jenkins Pipeline)

Now we need to configure the mail notification for which 465 port needs to be opened.

For this go to your google account 2 step verification - app password.

App name - Jenkins
and we get the password - lztc kzfe nrpd epqh

Now to set it go to manage jenkins - system - extended email notification.

Extended Email Notification -

SMTP SERVER - smtp.gmail.com

SMTP Port - 465

Credentials - Jenkins

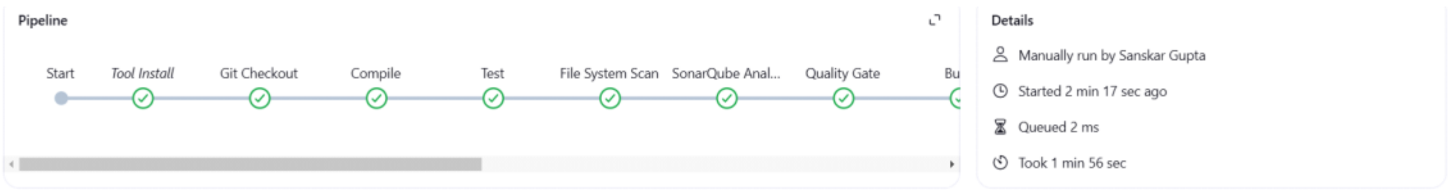
Use SSL

mail-cred

Email Notification -

Use SMTP Authentication

Now Run the Pipeline after adding the notification part into the jenkins file.



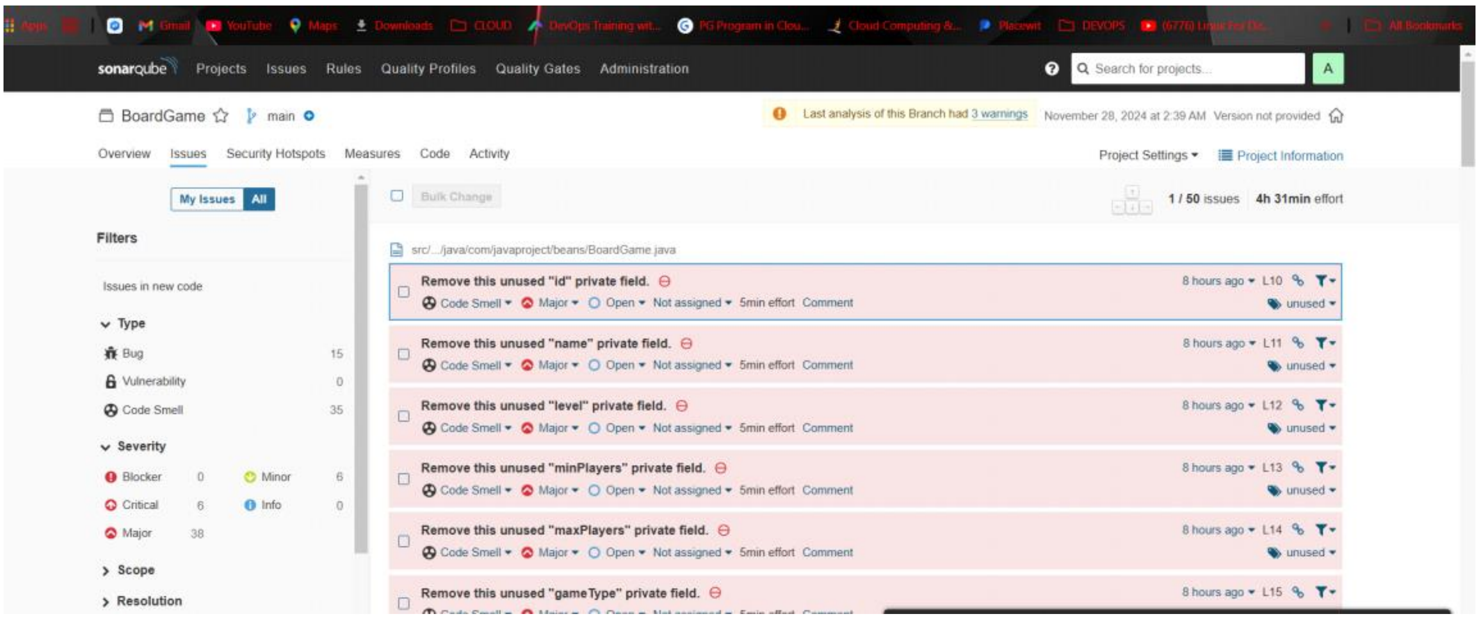
Details

Manually run by Sanskar Gupta

Started 2 min 17 sec ago

Queued 2 ms

Took 1 min 56 sec



Last analysis of this Branch had 3 warnings   November 28, 2024 at 2:39 AM   Version not provided

1 / 50 issues   4h 31min effort

Issues in new code

Type

Bug	15
Vulnerability	0
Code Smell	35

Severity

Blocker	0	Minor	6
Critical	6	Info	0
Major	38		

Scope

Resolution

src/.../java/com/javaproject/beans/BoardGame.java

Remove this unused "id" private field. — Code Smell · Major · Open · Not assigned · 5min effort · Comment — 8 hours ago · L10 · unused

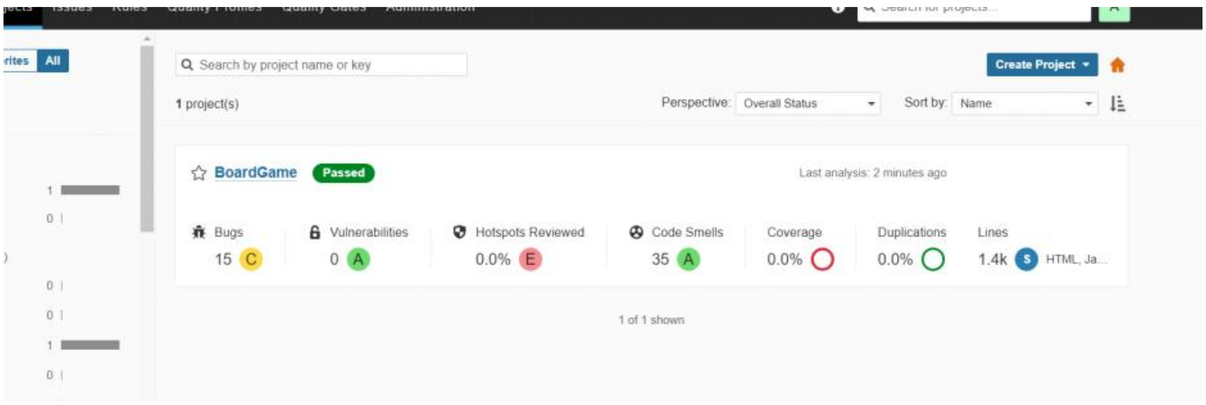
Remove this unused "name" private field. — Code Smell · Major · Open · Not assigned · 5min effort · Comment — 8 hours ago · L11 · unused

Remove this unused "level" private field. — Code Smell · Major · Open · Not assigned · 5min effort · Comment — 8 hours ago · L12 · unused

Remove this unused "minPlayers" private field. — Code Smell · Major · Open · Not assigned · 5min effort · Comment — 8 hours ago · L13 · unused

Remove this unused "maxPlayers" private field. — Code Smell · Major · Open · Not assigned · 5min effort · Comment — 8 hours ago · L14 · unused

Remove this unused "gameType" private field. — 8 hours ago · L15

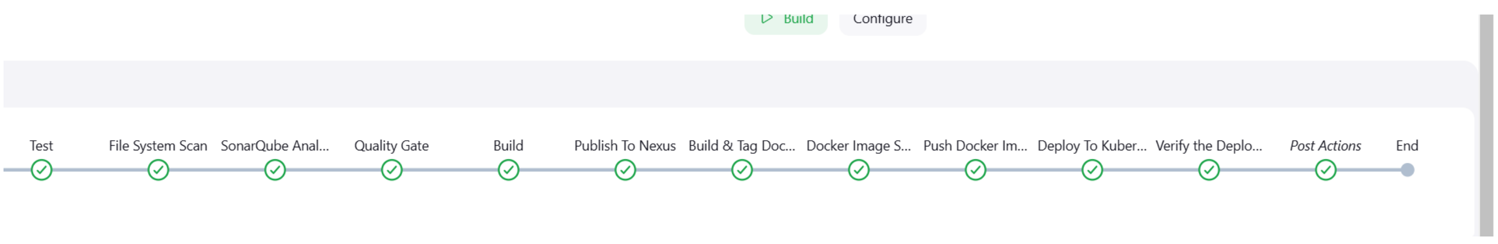


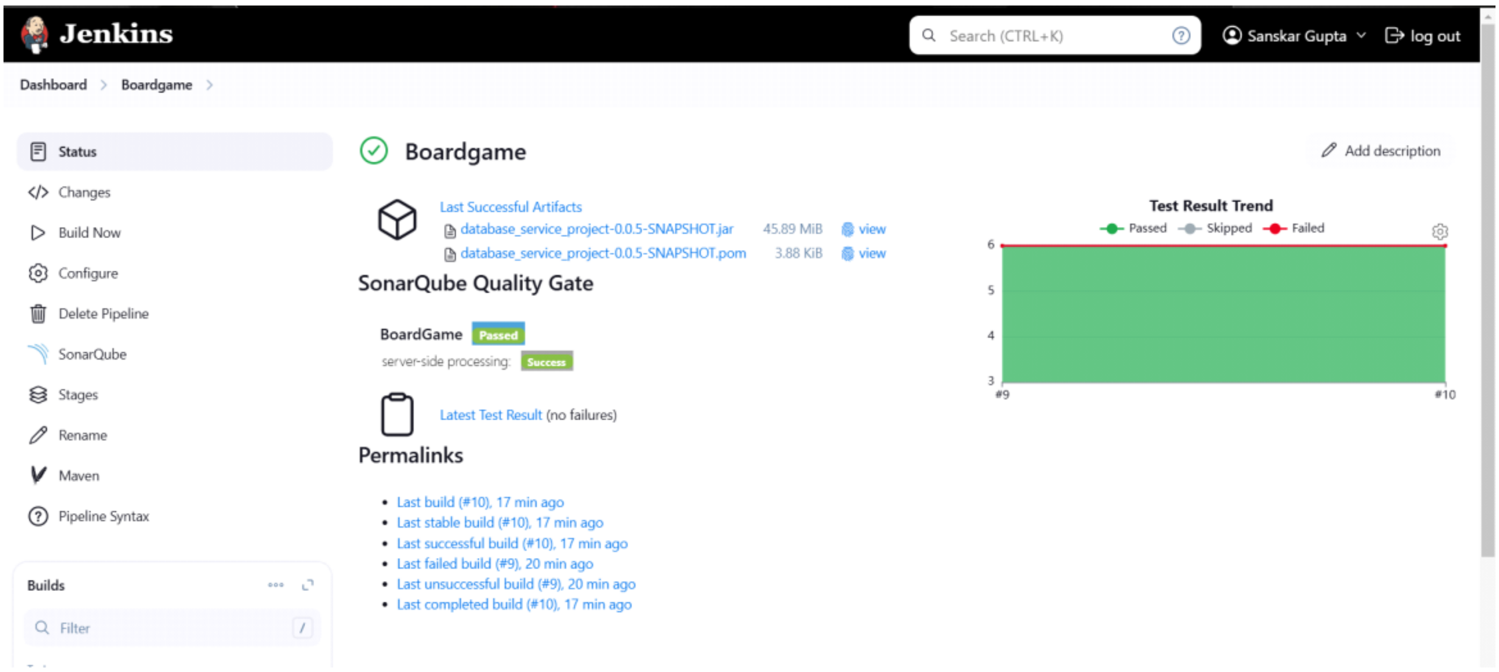
1 project(s)

BoardGame Passed   Last analysis: 2 minutes ago

Bugs	Vulnerabilities	Hotspots Reviewed	Code Smells	Coverage	Duplications	Lines
15 C	0 A	0.0% E	35 A	0.0%	0.0%	1.4k S HTML, Ja...

1 of 1 shown





Boardgame

Last Successful Artifacts
database_service_project-0.0.5-SNAPSHOT.jar   45.89 MiB   view
database_service_project-0.0.5-SNAPSHOT.pom   3.88 KiB   view

SonarQube Quality Gate

BoardGame Passed
server-side processing: Success

Latest Test Result (no failures)

Permalinks

- Last build (#10), 17 min ago
- Last stable build (#10), 17 min ago
- Last successful build (#10), 17 min ago
- Last failed build (#9), 20 min ago
- Last unsuccessful build (#9), 20 min ago
- Last completed build (#10), 17 min ago

As we have run the pipeline now successfully we have to do the configuration for the monitoring part.

Created Monitor server for monitoring as t2.medium.

Now we will install prometheus grafana in this.

→ Wget https://github.com/prometheus/prometheus/releases/download/v3.0.0/prometheus-3.0.0.linux-amd64.tar.gz

→ tar -xvf prometheus-3.0.0.linux-amd64.tar.gz

→ rm -rf prometheus-3.0.0.linux-amd64.tar.gz

→ cd prometheus-3.0.0.linux-amd64/

Now when I execute promethues file promethues will start to run immediately.

→ ./prometheus &

http://13.201.46.200:9090/

Now we will install Grafana.

→ `sudo apt-get install` -y adduser libfontconfig1 musl
→ `wget https://dl.grafana.com/enterprise/release/grafana-enterprise_11.3.1_amd64.deb`
→ `sudo dpkg` -i grafana-enterprise_11.3.1_amd64.deb

http://13.201.46.200:3000/login

Now we need to install blackbox exporter which will help us in monitor the application.

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

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

→ rm -rf blackbox_exporter-0.25.0.linux-amd64.tar.gz

→ cd blackbox_exporter-0.25.0.linux-amd64/

→ ./blackbox_exporter &

http://13.201.46.200:9115/

Now we need to change some configuration of promethus by adding config in prometheus.yml

→ vim promethues.yml
Edit the ip of blackbox running into the replacement and give the url of application which needs to be monitored.

```
 - job_name: 'blackbox'
   metrics_path: /probe
   params:
     module: [http_2xx]  # Look for a HTTP 200 response.
   static_configs:
     - targets:
       - http://prometheus.io    # Target to probe with http.
       -http://13.232.185.254:32028

   relabel_configs:
     - source_labels: [__address__]
       target_label: __param_target
     - source_labels: [__param_target]
       target_label: instance
     - target_label: __address__
       replacement: 13.201.46.200:9115  # The blackbox exporter's real hostname:port.
```

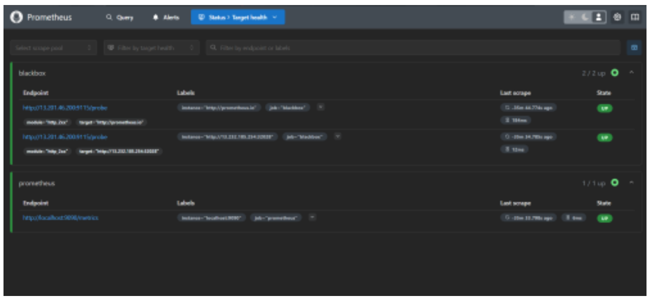
Now restart prometheus.

To get the id of process running.

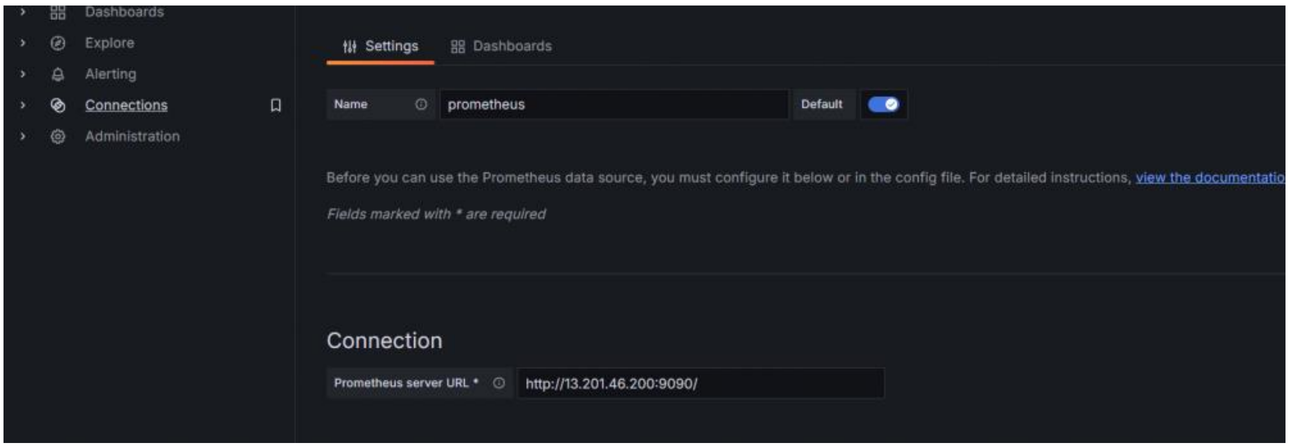
→ pgrep prometheus

→ kill processid

→ ./promethues &



Now we want to add prometheus as data source inside our grafana.



Name: prometheus
Default: on

Before you can use the Prometheus data source, you must configure it below or in the config file. For detailed instructions, view the documentatio

*Fields marked with * are required*

Connection

Prometheus server URL *: http://13.201.46.200:9090/

Copy the dashboard id from here and paste it in the import dashboard for prometheus.

https://grafana.com/grafana/dashboards/7587-prometheus-blackbox-exporter/

So, it was the application level monitoring now we will see how we can do system level monitoring.

To monitor Jenkins itself.

Install Prometheus Metrics Plugin.
Restart Jenkins.
Now Installing Node Exporter into the Jenkins Server.

```
wget https://github.com/prometheus/node_exporter/releases/download/v1.8.2/node_exporter-1.8.2.linux-amd64.tar.gz
```

Unzip delete the zip and cd into the node exporter and install node exporter and run it using jenkins ip and node exporter port.

http://3.108.170.189:9100/

Now we need to add some certain things inside prometheus server.

We need to add the new job of node exporter in prometheus.yml file.

Include Jenkins ip in this.

```
- job_name: 'node_exporter'
    static_configs:
      - targets: ['13.201.167.137:9100']

  - job_name: 'jenkins'
    metrics_path: '/prometheus'
    static_configs:
      - targets: ['13.201.167.137:8080']
```

Now Restart Prometheus.

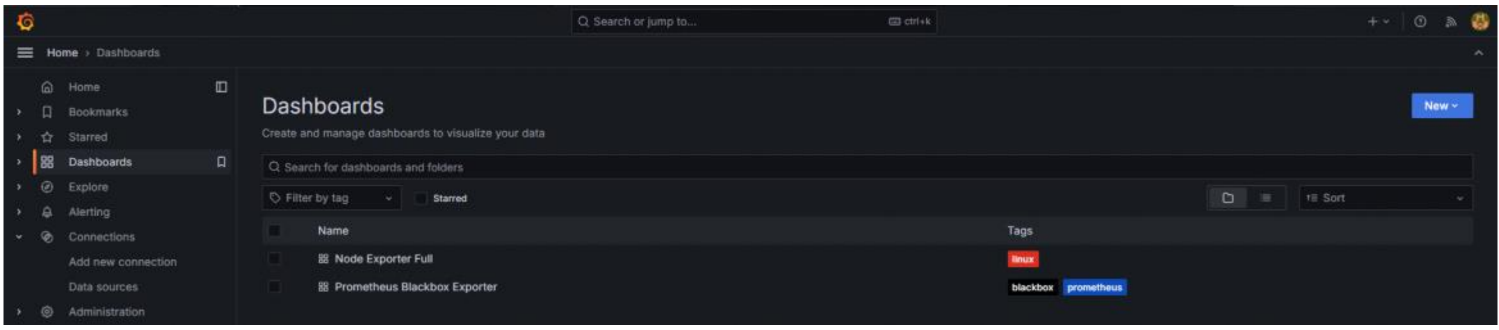
```
pgrep prometheus
```

```
kill processid
```

```
./prometheus &
```

Now inside grafana we can add another dashboard for the node exporter.

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



Dashboards

Create and manage dashboards to visualize your data

Name	Tags
Node Exporter Full	linux
Prometheus Blackbox Exporter	blackbox prometheus

Jenkins Pipeline -

```
pipeline {
    agent any
    tools {
        jdk 'jdk17'
        maven 'maven3'
    }
    environment {
        SCANNER_HOME= tool 'sonar-scanner'
    }
    stages {
        stage('Git Checkout') {
            steps {
                git branch: 'main', credentialsId: 'git-cred', url: 'https://github.com/sansugupta/Boardgame.git'
            }
        }

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

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

         stage('SonarQube Analsyis') {
            steps {
                withSonarQubeEnv('sonar') {
                    sh ''' $SCANNER_HOME/bin/sonar-scanner -Dsonar.projectName=BoardGame -
Dsonar.projectKey=BoardGame \
                            -Dsonar.java.binaries=. '''
                }
            }
        }
         stage('Quality Gate') {
            steps {
                script {
                  waitForQualityGate abortPipeline: false, credentialsId: 'sonar-token'
                }
            }
        }

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

     stage('Publish To Nexus') {
    steps {
        withCredentials([usernamePassword(credentialsId: 'nexus-credentials', usernameVariable: 'NEXUS_USER',
passwordVariable: 'NEXUS_PASS')]) {
            withMaven(globalMavenSettingsConfig: 'global-settings', jdk: 'jdk17', maven: 'maven3',
mavenSettingsConfig: '', traceability: true) {
                sh "mvn deploy -X -Dmaven.deploy.username=${NEXUS_USER} -Dmaven.deploy.password=
${NEXUS_PASS}"
            }
        }
    }
}
        stage('Build & Tag Docker Image') {
            steps {
               script {
                   withDockerRegistry(credentialsId: 'docker-cred', toolName: 'docker') {
                        sh "docker build -t sansugupta/boardshack:latest ."
                        }
                    }
               }
            }
            stage('Docker Image Scan') {
            steps {
                sh "trivy image --format table -o trivy-image-report.html sansugupta/boardshack:latest "
            }

        }
        stage('Push Docker Image') {
            steps {
               script {
                   withDockerRegistry(credentialsId: 'docker-cred', toolName: 'docker') {
                            sh "docker push sansugupta/boardshack:latest"
                    }
               }
            }
        }
        stage('Deploy To Kubernetes') {
            steps {
              withKubeConfig(caCertificate: '', clusterName: 'kubernetes', contextName: '', credentialsId: 'k8-cred',
namespace: 'webapps', restrictKubeConfigAccess: false, serverUrl: 'https://172.31.15.190:6443') {
                    sh "kubectl apply -f deployment-service.yaml"
              }
            }
    }
        stage('Verify the Deployment') {
            steps {
               withKubeConfig(caCertificate: '', clusterName: 'kubernetes', contextName: '', credentialsId: 'k8-cred',
namespace: 'webapps', restrictKubeConfigAccess: false, serverUrl: 'https://172.31.15.190:6443') {
                        sh "kubectl get pods -n webapps"
                        sh "kubectl get svc -n webapps"
                }
            }
        }

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

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

            emailext (
                subject: "${jobName} - Build ${buildNumber} - ${pipelineStatus.toUpperCase()}",
                body: body,
                to: 'sanskargupta966@gmail.com',
                from: 'jenkins@example.com',
                replyTo: 'jenkins@example.com',
                mimeType: 'text/html',
                attachmentsPattern: 'trivy-image-report.html'
            )
        }
    }
}

}
```