REACT APPLICATION DEPLOYMENT

Tools Used:

1. Jenkins
2. Sonarqube
3. Npm (NodeJS)
4. Docker
5. Trivy
6. Slack

Required Plugins:

1. Pipeline stage view
2. SonarQube Scanner
3. NodeJS
4. Docker pipeline
5. Slack Notification

We need to use 5 servers

1. Jenkins (t2.large) : access with 8080
    - Install git
    - Install jenkins and setup jenkins
        a. Install suggested plugins
        b. Install required plugins as per the app requirements
    - Install docker and start docker
    - Install Sonarqube using docker image and access with 9000 port number
        a. docker run -itd –name sonar -p 9000:9000 sonarqube:lts-community
        b. Access with public_IP:9000 and login with Default credentials (admin, admin)

   c. Set the credentials

2. Cluster Master (t2.large)
   - Install docker and start docker
   - Install java
   - Install compose
   - Initialize with docker swarm
   - Install Trivy
   - Install git
3. Workers (t2.micro)
   - Install and start docker
   - Copy the swarm command and paste, so that these will work as worker nodes

Implement Master and slave in Jenkins: Setup new node

a. Master and slave setup is required to run the pipeline
b. So, go to manage jenkins => Nodes => Add new node => node name and then create (No.of Executors, Remote root directory - /home/ec2-user/myjenkins/, labels – dev)
c. Usage – only build jobs with label expression matching this node, launch method – launch via ssh (cluster Master private IP in host, credentials, kind – ssh username with private key, username – ec2-user, private key enter directly, click on add and give the pem file, click on add and select the credentials, availability – keep this agent as much as possible, save)
d. Host key verification strategy (non-verifying verification strategy and then save

Stage-1: Clean Workspace

cleanWs()

a. Code will be on the workspace when we run the pipeline, so this will clean the data of the previous pipeline

b. When we run the pipeline 2nd time, the 1st time data will be cleared

Stage-2: Code

a. Get the code from Git Hub
b. Using git ‘repo_URL’

```
git ‘https://github.com/PrathimaVyas/Zomato-Project.git’
```

Stage-3: Code Quality Analysis

a. Go to sonarqube => My accounts => security => generate token
b. Copy the token and while adding credentials select secret text and paste the token

Stage-4: Quality Gates

a. Even if the quality gates are passed or fail it will go to the next stage to avoid that we should follow the below steps
b. Node js should be installed
c. New stage => pipeline syntax => waitforqualitygate and select the sonarqube credentials => paste the script in stage

Stage-5: Build the code

a. Install NodeJS plugin
b. Go to Manage Jenkins => tools => NodeJS => click on add and then add name and version
c. Then add the name of nodejs in tools at the starting of the pipeline

```
tools {
        nodejs ‘node16’
}

sh ‘npm run build’
```

Stage-6: Build dockerfile

a. First we need to go to the Master of the cluster and run the command below otherwise it will fail

    chmod 777 /var/run/docker.sock

b. Know the path of Dockerfile

c. In my case I have Dockerfile in the main directory

d. Create image name with docker hub username and repo name and then tag the image

    sh 'docker build -t prathima77/zomato-app:zomatoimage .'

Stage-7: Scan the images(no plugins required for trivy)

a. After installing the trivy, we need to run the command below

    sh 'trivy image prathima77/zomato-app:zomatoimage'

Stage-8: Push the Image to Docker Hub

a. Install Docker pipeline plugin

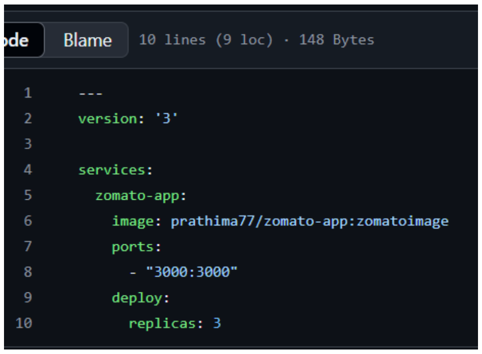
b. Pipeline syntax => withdockerregistry => give the dockerhub credentials => generate pipeline script

c. Now we need to add the commands inside the generated script

    sh 'docker push prathima77/zomato-app:zomatoimage'

Stage-9: Docker Stack Deploy

a. Added compose.yml



```
---
version: '3'

services:
  zomato-app:
    image: prathima77/zomato-app:zomatoimage
    ports:
      - "3000:3000"
    deploy:
      replicas: 3
```

b. Now run the below command

    sh 'docker stack deploy zomato-app --compose-file=compose.yml'

Stage-10: Slack Integration

a. Install Slack notification plugin in the jenkins
b. Create a slack account
c. Give the workspace name => next => add teammates if required => next => give the name of class => continue with free version
d. On top left, will get the dropdown => tools and settings => manage apps => go to installed apps => search for Jenkins CI => add to slack => give the channel name which we gave earlier => add Jenkins CI integration
e. Go to manage Jenkins => system => search for slack => workspace name => configure, click on add => select the Jenkins => kind (secret text) => copy the integration token ID in slack and paste it in secret => workspace will be the team subdomain in slack (step-3) => channel name => Test connection

```
post {
        always {
                slackSend channel: 'all-docker-project', message: "*${currentBuild.currentResult}:* Job ${env.JOB_NAME}\n Build ${env.BUILD_NUMBER}\n More info at: ${env.BUILD_URL}"
        }
}
```

Now we can access the application with port number 3000 from master and workers