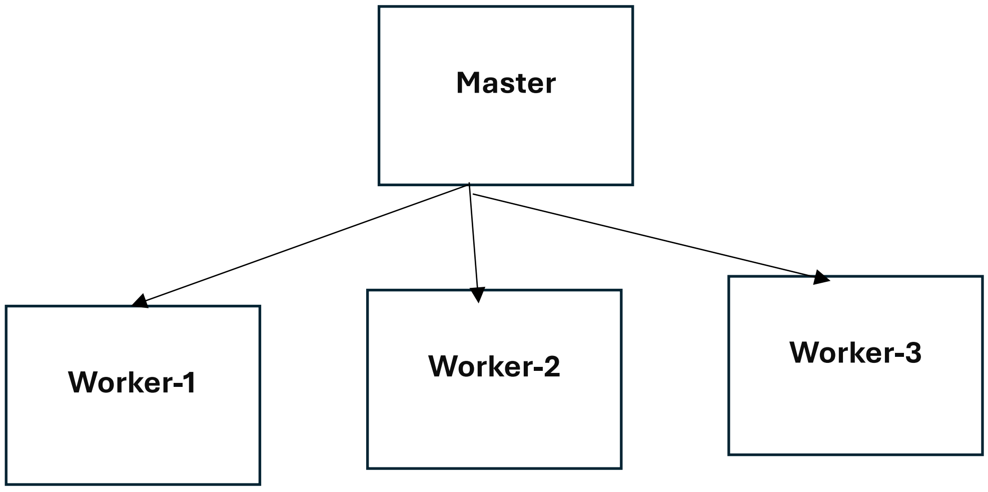
PROJECT-1

SPRING BOOT APPLICATION DEPLOYMENT

- ⇨ To deploy a application, we fixed to containerize so we can do it using docker and Kubernetes
- ⇨ We can deploy small applications in docker like travel companies(100-1000 people will use). Low cost
- ⇨ We can deploy large applications in Kubernetes like amazon, Flipkart, Facebook, Instagram etc lakhs of people will use (high cost)
- ⇨ We will use the stack to deploy the application because it has swarm and compose
- ⇨ Usually we will run our application in multiple data centers (Availability Zone). Now we will take the single server and now the application will be accessed from that data center
  - if something happens to that then that would be risky
  - if multiple users access at a time it is difficult to handle
- ⇨ Now we will take different servers in different available zones and for that we will use a load balancer, if we want to deploy the application in these 3 servers using load balancer we need master
- ⇨ Master will take the responsibility of the slaves using docker swarm we create cluster but for creating applications we need multiple servers so we use compose to use this compose and swarm we will use docker stack
- ⇨ Now we will create a cluster, using docker swarm init we will establish connection between the master and 3 workers
- ⇨ Now we will install compose in master



- ⇨ We need to know the requirements of the developers regarding CPU, RAM, Storage, operating system. Based on that we will create a server called Dev server
- ⇨ So we need to write the pipeline for that

CI/CD server:

Code => build => test => deploy (adding webhooks for automation) if it works fine then the pipeline will run in the testing server

Testing server:

Code => build => test => deploy => testing engineers will perform all the test cases

- ⇨ Now we have to containerize
  Code => CQA (failed then inform developers) => build (maven/npm/pip) => deploy => build the Dockerfile => scan the image => push the image to Docker Hub => create container (using compose file) ➔ first deploy in dev environment

- ⇨ If it is success, then the main pipeline for test environment if that also works then we need to maintain pipeline for production
  Code => build (maven/npm/pip) => deploy => build the Dockerfile => push the image to Docker Hub => stack deployment => slack integration

We need 4 servers for cluster and jenkins server

1. Master => t2.large
   - Install docker and git
   - Install trivy
   - Install compose
   - Swarm initialization and copy the command and paste in workers
     docker swarm init
2. Workers - 3 => t2.micro
3. Jenkins => t2.large
   - Docker
   - Git
   - Nexus:8081
     a. Sign in with username admin and then copy the path and paste in server for password
     b. Reset the password

- SonarQube:9000
    a. docker run -itd --name sonar -p 9000:9000 sonarqube:lts-community
    b. Access with public_IP:9000 and set the credentials
- Jenkins:8080
    a. Install suggested plugins
    b. Set the credentials
- In real-time we will maintain separate servers for the sonarqube and nexus

Implement master and slave

Master and slave setup is required to run the pipeline. So, go to manage jenkins => Nodes => Add new node => node name and then create (No.of Executors will be depends on master and slaves, Remote root directory - /home/ec2-user/myjenkins/, labels – dev, 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) => Host key verification strategy (non-verifying verification strategy and then save

Tools Used:

1. Jenkins
2. Sonarqube
3. Maven
4. Nexus
5. Docker
6. Trivy
7. Slack

Plugins:

1. Pipeline stage view
2. Sonarqube scanner
3. Eclipse termurin installer
4. Nexus Artifact Uploader
5. Docker pipeline
6. Slack notification
7. Nodejs

Add agent using label

```
agent{
        node{
                label 'dev'
        }
}
```
dev => label name which I gave while setting up the node

Pipeline stages:

Stage-1: Clean WorkSpace

```
cleanWs()
```

a. Code will be on workspace when we run the pipeline, so this will clean the data of the previous pipeline
b. When we run the pipeline 2nd time, 1st time data will be cleared

Stage-2: Get the code

a. Get the code from Git hub
b. Using git 'Repo_URL'
   git 'https://github.com/PrathimaVyas/dockerwebapp.git'

Stage-3: Code Quality Analysis

a. Official document
b. Go to Sonarqube => manually => give the name of the project => select Jenkins => GitHub => analysis prerequisites => we need to follow those steps or we can directly
c. Install SonarQube Scanner plugins
d. Go to sonarqube => My account => security => generate and use the token

```
tools{
        maven 'mymaven'
        jdk 'jdk17'
        node 'node16'
}
```

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. Manage Jenkins => tools => give the name in maven and save
b. Add maven, in tools before the stages
    sh 'mvn clean package'
c. We will get the target folder now that folder we need to send this folder to Docker-app folder because we have Dockerfile in that repo
    Sh 'cp -r target

Stage-6: Deploy application to Nexus

a. Install nexus artifact uploader plugin
b. Add the credentials
c. Create repository => maven 2 hosted => allow redeploy => save
d. Add stage for nexus, Pipeline syntax => NexusArtifactUploader => version 3 => Nexus URL (IP:port_number) => add credentials (username and password and then add) => group ID, version will be there in pom.xml file => Nexus repo name => add artifact (artifactID will be there in pom.xml, type => war, file => target/file.war) => generate script => build
e. Go to browse in nexus there we can see the artifacts

Stage-7: 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 Docker-app, Docker-db
d. Create image name with docker hub username and repo name and then tag
    sh 'docker build -t prathima77/docker-app:appimage Docker-app'
    sh 'docker build -t prathima77/docker-db:dbimage Docker-db'

Stage-8:Scan the images (no plugins required for trivy)

a. To check the vulnerabilities of the images to scan we use trivy
b. Install Trivy in Cluster Master
    vim .bashrc (add below line at the last)

export PATH=$PATH:/usr/local/bin

source .bashrc (to install trivy we need to make this as source)

c. Steps to install trivy

   wget https://github.com/aquasecurity/trivy/releases/download/v0.18.3/trivy_0.18.3_Linux-64bit.tar.gz

   tar -zxvf trivy_0.18.3_Linux-64bit.tar.gz

   mv trivy /usr/local/bin

   trivy --version

d. To scan the image

   trivy image image-name (manual)

e. In pipeline stage we need to run the commands below

   sh 'trivy image prathima77/docker-app:appimage'

   sh 'trivy image prathima77/docker-db:dbimage'

Stage-9: Push the images 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/docker-app:appimage'

   sh 'docker push prathima77/docker-db:dbimage'

Stage-10: Docker Stack Deploy

a. Write a compose file and we need to give the image names aswell
b. Now in the stage

   sh 'docker stack deploy myapp –compose-file=compose.yml'

Stage-11: 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

a. Now go to Master server and run the **'docker stack ls'** command to list the services
b. Now we can list those services using **'docker service ls'** command to list the services
c. Check the container and copy the public IP of the server and then paste it with the port number mentioned for the container