**Setup Jenkins (Ubuntu/Debian)**

1. **We can setup Jenkins in multiple ways, Setting it up manually using Jenkins Package**

Jenkins need Java as prerequisites

* sudo apt-get update

sudo apt-get install -y openjdk-11-jre (*Install java in ubuntu Linux Machine* )

* curl -fsSL https://pkg.jenkins.io/debian-stable/jenkins.io-2023.key | sudo tee \

/usr/share/keyrings/jenkins-keyring.asc > /dev/null

* 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 -y jenkins (*This will create symlink too*)

check the status once its done setting up by executing

sudo systemctl status Jenkins (*should show below active running status*)

A picture containing screenshot, text

Description automatically generated

**Post Installation wizard:**

After installation is done, open Jenkins with Public IP of ec2 Machine with Port 8080 as Jenkins Default Port which can be seen in /etc/default/Jenkins file, once the screen gets loaded it will ask for InitialPassword which will be in **/var/lib** folder

*sudo cat /var/lib/jenkins/secrets/initialAdminPassword*

Next it would be prompting to create an admin user or skip it, provide username and password and create admin user.

Select Plugins to install (GIT, GitHub, Credentials Binding, Junit, Parameterized Trigger, NodeJs, Git Parameter, Role-based Auth Strategy etc), Jenkins setup will be done and re-directed to Login Page

Enter Admin Username and Password to login

**2) Setup Jenkins using Docker**

Install docker in ec2

sudo apt-get update

sudo apt-get install docker -y

Add ubuntu user to docker group

sudo usermod -a -G docker ubuntu

create a bridge network

docker network create Jenkins

Run container using below (using baseimage myjenkins-blueocean:2.401.1-1)

Provide certificate path if we have it enabled for HTTPS

docker run \

--name jenkins-blueocean \

--restart=on-failure \

--detach \

--network jenkins \

--env DOCKER\_HOST=tcp://docker:2376 \

--env DOCKER\_CERT\_PATH=/certs/client \

--env DOCKER\_TLS\_VERIFY=1 \

--publish 8080:8080 \

--volume jenkins-data:/var/jenkins\_home \

--volume jenkins-docker-certs:/certs/client:ro \

myjenkins-blueocean:2.401.1-1

Once Jenkins runs, Proceed to Post Installation Steps as discussed above

Check whether its installed and running

docker ps -a ( *see status if its showing as Running* )

**Authentication/Authorization Users:**

Login as admin user into Jenkins server

Navigate to Manage Jenkins

Click on Users, Create User by entering details (*This option is for test purpose if Jenkins is not integrated with LDAP Server*)

After user has been created, Navigate to Manage Jenkins, Go to Security

In Security Realm Choose LDAP or Unix user/group database if its configured

Under Authorization Section

Select Project-Based Matrix Authorization Strategy

You will find users/groups please give necessary permissions (whether able to create pipeline or just Read)

**Pipeline Creation and Integrate with Github Repository:**

If you are admin user and has Manage Jenkins access

1. Go to Manage Jenkins, Select Credentials and add credentials scoped to Jenkins User

Click on Add Credentials, Select Username with Password from Kind

We have multiple options in Kind alongside Username with Password

***Secret file***: When we have secret file to connect

***Secret text*** : If we have key-value secret (Ex: aws-access-key, aws-secret-key etc)

***SSH username with private key****:* If we want to connect to any Third Party App or Linux Machine using SSH use this option

Provide username and password as Personal Access Token from GitHub Settings

A screenshot of a computer

Description automatically generated with medium confidence

1. Go to Manage Jenkins, Select Plugins and Install Multibranch Scan Webhook Trigger which will help to scan Multi Branch Pipeline
2. *Configure MultiBranch Pipeline*

A multi-branch pipeline in Jenkins is used to automate the building, testing, and deployment of software projects with multiple branches, such as feature branches or release branches. It allows you to define a Jenkins pipeline that automatically creates and manages individual pipelines for each branch in your repository.

The main advantage of using a multi-branch pipeline in Jenkins is that it eliminates the need to manually create and configure separate Jenkins jobs for each branch. Instead, it automatically detects branches in your source code repository and creates corresponding pipelines dynamically.

A screenshot of a computer

Description automatically generated with medium confidence

1. In the configuration section of pipeline

Under Build Configuration

Provide HTTPS URL of GITHUB Private Repository

Select credentials which is used to connect to GitHub Repo from Credentials Dropdown which was created earlier

1. Under Scan Repository Triggers Section

Check Scan by Webhook and enter Trigger token

1. Under Orphaned Item Strategy

Discard old Items, limit no of days of deleted branches and PRs to exist

1. Finally Apply and Save Pipeline

***Integration with Github Repository:***

Go to Repository Settings

Under Settings Left Side Pane Click on Webhooks

Click on Add Webhook

Enter Payload URL with http://<ec2-public-ip>:8080/multibranch-webhook-trigger/invoke?token=<token name at step 5>

Content-Type: application/json

Event: Just the push event

Active and Click Create Webhook

Once Pipeline and Webhook has been set, Clone Repo and create your own branch from main default branch, make any changes and commit changes, Pipeline will trigger automatically new build will be started. Watch Logs for Build in Console Output of each build inside Jenkins Pipeline

**Adding Slave Nodes to Jenkins Master**

1. Create slave ec2 machine and connect to slave ec2
2. Install java on slave node (match java version as master)
3. Create Jenkins user and add ssh keys

useradd Jenkins

ssh-keygen -t rsa

1. Copy public key to authoried\_keys and give permissions

cd .ssh

cat id\_rsa.pub > authorized\_keys

sudo chmod 700 authorized\_keys

1. Copy Keys to Master Node (Connect to Master EC2)

mkdir /var/lib/Jenkins/.ssh

sudo chown Jenkins:Jenkins /var/lib/Jenkins/.ssh

1. Copy public key from Slave Node to /var/lib/Jenkins/.ssh/known\_hosts
2. ssh-keyscan -H <Private-ip-slave-node> >> /var/lib/Jenkins/.ssh/known\_hosts

Once above keys are configured in Slave Node and Copied Key from Slave to Master

Navigate to Jenkins Server, Login as Admin User, Navigate to Manage Jenkins -> Nodes and Clouds -> New Node

Enter Node Name and Choose Permanent Agent and Create Node

A picture containing text, screenshot, font

Description automatically generated

Create button will take to below screen

Provide Remote root directory of Slave Machine

Provide Labels name as “Jenkins-slave” or “<something>”

Choose Launch Method as Below “Launch agents via SSH”

Provide Host as <private-ip-slave-node>

Click Add Credentials which will take to below screen

Select Kind as SSH Username with private key

ID as Jenkins

Private Key -> Enter directly -> Copy and Paste Private SSH key from Slave Node and Save

A screenshot of a computer

Description automatically generated with low confidence

Choose Credentials from dropdown once saved above

Select Host Key Verification Strategy as “**Known Hosts file verification strategy**”

Save it Slave node will be added, Check the Status of Slave node.

A picture containing text, screenshot, line, number

Description automatically generated

**Setup Email Notification in Jenkins**

1. Install Email Extension Plugin
2. Navigate to Manage Jenkins (Admin User) -> Configure System -> E-mail Notification

SMTP Server -> smtp.office365.com / smtp.gmail.com

Advanced :

Check Use SMTP Authentication

Provider UserName with Email

Entire Password

Use SSL

Port 25 (office365), Port 465 for smtp.gmail.com

Click Apply and Save

**Setup Postman using newman CLI in Node**

1. Install nodejs and npm

sudo apt-get install nodejs -y

sudo apt-get install npm -y

1. Install newman

npm install -g newman

**Publishing Artifacts to S3**

Create S3 Bucket for each environment and Enable Versioning

Create IAM Role with IAM Policy Permissions of S3Object Read and PutObject

Attach that IAM Role with Jenkins EC2 Instance Profile

And upload build artifacts in Pipeline to AWS S3 using AWS CLI

Install all necessary plugins for this pipeline to Execute (Pipeline-Utility and Maven Integration)

*pipeline {*

*agent any*

*options {*

*//Disable concurrentbuilds for the same job*

*disableConcurrentBuilds()*

*// Add timestamps to console log*

*timestamps()*

*skipDefaultCheckout true*

*}*

*environment {*

*ARTIFACTID = readMavenPom().getArtifactId()*

*VERSION = readMavenPom().getVersion()*

*S3\_BUCKET = '<replace this with bucket name>’*

*}*

*stages {*

*stage ('Checkout') {*

*steps {*

*checkout([$class: 'GitSCM', branches: [[name: "refs/remotes/origin/main"]], doGenerateSubmoduleConfigurations: false, extensions: [], submoduleCfg: [], userRemoteConfigs: [[credentialsId: 'raghu-github', url: 'https://github.com/Raghupatik/java-test.git']]])*

*}*

*}*

*stage('Build') {*

*steps {*

*script {*

*sh 'mvn clean install'*

*}*

*}*

*}*

*stage('Push to artifactory') {*

*steps {*

*script {*

*JARNAME = ARTIFACTID+'-'+VERSION+'.jar'*

*echo "JARNAME: ${JARNAME}"*

*sh "aws s3 cp target/${JARNAME} s3://$S3\_BUCKET"*

*currentBuild.result = ‘SUCCESS’*

*}*

*}*

*}*

*stage ('Run postman test on Production') {*

*parallel {*

*stage ('Run postman test on Tomcat Production') {*

*agent { label 'master'}*

*steps {*

*script {*

*retry(2) {*

*try {*

*echo 'Run test on staging server'*

*sh '/apps/tools/node-v8.10.0-linux-x64/bin/newman run src/test/java/com/example/resources/postman-testscript.json -d src/test/java/com/example/resources/tomcat-env.json -k'*

*currentBuild.result = 'SUCCESS'*

*} catch(Exception err) {*

*currentBuild.result = 'FAILURE'*

*}*

*}*

*}*

*}*

*}*

*}*

*}*

*stage ('Notify when all stages are done') {*

*agent { label 'master' }*

*steps {*

*script {*

*if (currentBuild.result == 'SUCCESS') {*

*echo "The deployment process is done, new version is deployed successfully"*

*mail (to: 'ragupathi.kommidi@gmail.com', subject: "SUCCESS: The job '${env.JOB\_NAME}' (${env.BUILD\_NUMBER}) is done, new version is deployed successfully", body: "Please go to ${env.BUILD\_URL} for the detail")*

*} else {*

*mail (to: 'ragupathi.kommidi@gmail.com', subject: "ERROR: The job '${env.JOB\_NAME}' (${env.BUILD\_NUMBER}) failed, rollback to last stable version", body: "Please go to ${env.BUILD\_URL} for the detail")*

*echo "ERROR: The job ${env.JOB\_NAME} (${env.BUILD\_NUMBER}) failed, rollback to last stable version"*

*sh 'exit 1'*

*}*

*}*

*}*

*}*

*}*

*}*

**Storing Credentials (Using Jenkins Credentials and Ansible-Vault)**

Creating Jenkins credentials is explained in Pipeline Creation Section, now how can we consume those credentials in pipeline, Install *Credentials Binding* Plugin

Fetch credentials in environment section an option

*pipeline {*

*agent {*

*// Define agent details here*

*}*

*environment {*

*AWS\_ACCESS\_KEY\_ID = credentials('jenkins-aws-secret-key-id')*

*AWS\_SECRET\_ACCESS\_KEY = credentials('jenkins-aws-secret-access-key')*

*}*

*stages {*

*stage('Example stage 1') {*

*steps {*

*withCredentials([usernamePassword(credentialsId: <provide the id stored in Jenkins credentials>, usernameVariable: 'USERNAME', passwordVariable: 'PASSWORD')]) {*

*// available as an env variable, but will be masked if you try to print it out any which way*

*// note: single quotes prevent Groovy interpolation; expansion is by Bourne Shell, which is what you want*

*sh 'echo $PASSWORD'*

*// also available as a Groovy variable*

*echo USERNAME*

*// or inside double quotes for string interpolation*

*echo "username is $USERNAME"*

*}*

*}*

*}*

*}*

*}*

Credentials using Ansible-Vault

Create a configuration file and encrypt it at same time.

*Ansible-vault create <file>*

Encrypt vault

Ansible-vault encrypt <filename>

Provide *password as it prompts*

*Install ansible Jenkins Plugin and then* add a new "Credential" as "Secret text". Set the value as vault and ID as Ansiblevault then save it.

*pipeline {*

*agent any*

*stages {*

*stage('Deploy') {*

*steps {*

*echo '> Deploying the application ...'*

*ansiblePlaybook(*

*vaultCredentialsId: 'AnsibleVault',*

*inventory: 'cicd/provision/stag/hosts.yml',*

*playbook: 'cicd/provision/stag/site.yml'*

*)*

*}*

*}*

*}*

*}*

*Host file example:*

all:

hosts:

staging:

ansible\_connection: ssh

ansible\_user: vagrant

ansible\_host: 192.168.99.30

ansible\_port: 22