GCP WORKING SCENARIO WITH JENKINS

1. Create CICD Pipeline to build and deploy java application using plain maven build and export war file to GCS Bucket. Deploy the War file from bucket to Server

Step1: Create a Compute Instance VM with Default - Service Account (SA) and Default VPC (in VPC firewall Add-fire wall rule: All-Ports – Any Where).

Step2: Goto IAM – Select the Default Compute Engine- SA - Assign Roles: Storage Admin & Storage Object User & Generate the **JSON Key**

Step3: Login to VM – Instance SSH – Install Maven-Jenkins-java17- Git and Tomcat.

Copy and paste the below scripts for Installation Process:

RUN MANUALLY -

- **Installing your desired maven version in particular path:**
- Download Maven:

wget https://apache.osuosl.org/maven/maven-3/3.9.5/binaries/apache-maven-3.9.5-bin.tar.gz

- **Extract the archive:** tar xzvf apache-maven-3.9.5-bin.tar.gz
- **Move Maven to /opt directory (optional):** sudo mv apache-maven-3.9.5 /opt
- **Set environment variables:** vim ~/.bashrc (copy paste below command)

```
export M2_HOME=/opt/apache-maven-3.9.5

export PATH=$M2_HOME/bin:$PATH
```

- **Run This Command:** source ~/.bashrc
- **Verify installation:** mvn --version

Jenkins & Git - Installation

Vim jenkins.sh (Copy & Paste the below script in this file):

```
sudo apt update

sudo apt install git -y

sudo apt install openjdk-17-jre -y

 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 jenkins –y
```

To Run The Script : sh jenkins.sh

Tomcat Installation:

Vim tomcat.sh (Copy & Paste the below script in this file):

```
sudo apt install default-jdk -y

wget https://dlcdn.apache.org/tomcat/tomcat-9/v9.0.89/bin/apache-tomcat-9.0.89.tar.gz

tar -zxvf apache-tomcat-9.0.89.tar.gz

sed -i '56  a\<role rolename="manager-gui"/>' apache-tomcat-9.0.89/conf/tomcat-users.xml

sed -i '57  a\<role rolename="manager-script"/>' apache-tomcat-9.0.89/conf/tomcat-users.xml

sed -i '58  a\<user username="tomcat" password="tomcat" roles="manager-gui, manager-script"/>' apache-tomcat-9.0.89/conf/tomcat-users.xml

sed -i '59  a\</tomcat-users>' apache-tomcat-9.0.89/conf/tomcat-users.xml

sed -i '56d' apache-tomcat-9.0.89/conf/tomcat-users.xml

sed -i '21d' apache-tomcat-9.0.89/webapps/manager/META-INF/context.xml

sed -i '22d'  apache-tomcat-9.0.89/webapps/manager/META-INF/context.xml

sh apache-tomcat-9.0.89/bin/startup.sh
```

To Run Script : sh jenkins.sh

Note: Tomcat Version may Change Refer the below link and Update the Script.

https://dlcdn.apache.org/tomcat

NOTE: In the script for Login given: PASSWORD=USERNAME=tomcat

Step4: After Installation of Jenkins and Tomcat – Change Port Number for Jenkins.

Follow the below script to change the port number:

vim script.sh (copy paste below code and run script)

```
#Changing port from 8080 to 8082

sudo sed -i 's/Environment="JENKINS_PORT=8080"/Environment="JENKINS_PORT=8081"/' /usr/lib/systemd/system/jenkins.service

sudo systemctl daemon-reload

sudo systemctl restart jenkins.service
```

To Run the script: sh script.sh

Step5: Now login to the Jenkins Console with the use of External IP and Jenkins Port Number

Step6: Setup Your Jenkins Console & create a Pipeline Job.

Setp7: Goto Manage Jenkins -> Plugins -> Install **Google Cloud Storage & Deploy to Container**

Step8: Manage Jenkins -> Credentials -> Global -> Add Credentials -> kind (Google Service Account from Private Key) -> project ID -> select JSON Key.

Step9: Now write the below Pipeline-Script:

```
pipeline {
  agent any
  tools {
    maven 'maven' // Use the name configured in Jenkins
  }
  stages {
    stage('checkout') {
      steps {
        git 'https://github.com/summu97/monolythic-project.git'
      }
    }
    stage('package') {
      steps {
        sh 'mvn clean package'
      }
    }
    stage('storage') {
      steps {
        googleStorageUpload bucket: 'gs://YOUR_BUCKET_NAME/', credentialsId:
'YOUR_JSON_KEY_CREDENTIAL_ID', pattern: 'target/*.war'
      }
    }
    stage ('Download from GCS') {
      steps {
        googleStorageDownload bucketUri: ' YOUR_gsutil_URI ', credentialsId:
'YOUR_JSON_KEY_CREDENTIAL_ID', localDirectory: '.'
      }
    }
    stage('Deploy') {
      steps {
        deploy adapters: [
          tomcat9(
            credentialsId: 'YOUR_TOMCAT_CREDENTIAL_ID',
            path: '',
            url: 'TOMCAT_SERVER_URL'
            )
            ],
            contextPath: 'tomcat-netflix',
            war: 'target/NETFLIX-1.2.2.war'
      }
    }
  }
}
```

❖ **ANOTHER APPROACH TO STORE WAR FILE TO STORAGE & THEN DOWNLOAD AND DEPLOY TO TOMCAT:**

In VM Instance Terminal - Set password to Jenkins-Change permissions to sudoers file:

- passwd jenkins

enterpassword:

re-enterpassword:

- chmod 640 /etc/sudoers
- vim /etc/sudoers (add below line and :wq)

```
jenkins    ALL=(ALL:ALL) ALL
```

- chmod 400 /etc/sudoers (run again after saving above line in the file)

In Jenkins UI: Add above Jenkins password in credentials: Kind (secret text)

```
pipeline {
  agent any
  tools {
    maven 'maven' // Use the name configured in Jenkins
  }
    environment {
    MY_PASSWORD = credentials('Paste-jenkins-password-ID')
  }
  stages {
    stage('checkout') {
      steps {
        git 'https://github.com/summu97/monolythic-project.git'
      }
    }
    stage('package') {
      steps {
        sh 'mvn clean package'
      }
    }
    stage('storage') {
      steps {
        googleStorageUpload bucket: 'gs://YOUR_BUCKET_NAME', credentialsId: 'Paste-json-key-ID', pattern:
'target/*.war'
      }
    }
    stage('Copying war') {
      steps {
        sh 'gsutil cp gs://YOUR_BUCKET_NAME/target/NETFLIX-1.2.2.war .'
      }
    }
    stage('Deploy to container') {
      steps {
        sh '''
          echo $MY_PASSWORD | sudo -S  mv  NETFLIX-1.2.2.war /root/apache-tomcat-9.0.88/webapps
        '''
      }
    }
  }
}
```

2. **How can one back up the configuration of a parent Jenkins server, compress it, and store it in a Google Cloud Storage (GCS) bucket, then retrieve the compressed Jenkins folder from GCS to apply the same configuration to a newly created Jenkins server?**

Procedure for Backing Up and Restoring Jenkins Configuration Using Google Cloud Storage (GCS):

Pre-requisites:

- 2-VM-instance (Parent Jenkins & Child Jenkins VM) having Service-Account-roles: Storage Legacy Bucket Owner & Storage Object Admin
- Create A bucket In GCS and Assign Permission for above used Service account in VM Creation

In Parent Jenkins – VM Instance – SSH Console:

Step1 – Stop Jenkins

- systemctl stop jenkins

Step2 – Compress (tar) the Jenkins directory

- tar –zcvf jenkins.restore.tar.gz /var/lib/jenkins/

Step3 – Store the compressed Jenkins Directory in GCS-Bucket

- gsutil cp jenkins.restore.tar.gz gs://BUCKET_NAME/jenskins.restore.tar.gz

In Child VM Instance – SSH Console:

Step1 – Stop Jenkins

- systemctl stop jenkins

Step2 – Restore the Parent Jenkins directory from GCS-Bucket

- gsutil cp gs://BUCKET_NSME/jenkins.restore.tar.gz jenkins.restore.tar.gz

Step3 – Empty the Present Jenkins directory in Child VM

- rm -rf /var/lib/Jenkins

Step4 – Untar the Jenkins directory

- tar -zxvf jenkins.restore.tar.gz -C /

Step5 – Start the Jenkins and Verify the Parent Jenkins Job are reflected in Child Node

- systemctl start jenkins

NOTE: For Jenkins and other Installation Refer the Page 2-3

3. How to deploy a react application to Server Using Jenkins-GCP Using Master Slave Concept.

❖ Master Slave Setup in Jenkins with GCP:

Step1 – Create a VM-instance (slave1) - install java-git-maven

Step2 – In slave instance terminal Generate SSH keys and copy public key in authorized_keys and put private key aside

Process:

- ssh-keygen
- cd .ssh
- cat id_rsa.pub (copy paste content in authorized keys)i.e
- vim authorized keys (paste content of id_rsa.pub here)
- cat id_rsa (copy this content and keep a side you can use it when configuring slave credentials)

Step3 – Create remote root directory:

- mkdir /root/jenkins
- chmod 755 /root/jenkins

Step4 – Goto Jenkins Dashboard -> Manage Jenkins -> Nodes

-> New Node -> Give Name -> Permanent Agent (select) -> Create

-> Number of executors (2) ->Remote root directory (/root/jenkins) ->Labels (slave1)

->Usage (only build jobs with label expression matching this node)

->Launch method (Launch agents via SSH)

-> Host (private-ip of slave)

->Credentials (jenkins)

->SSH username and private key ->Username (root) -> Private Key

->Enter directly (paste private key) ->ADD

->Host Key Verification Strategy (Non verifying strategy)->Tool Locations (give maven tool location) ->save

NOTE: Make sure your Jenkins server's service account have Permissions to login to another server.

Step5 – Setup Node and npm installation in Salve1 – VM instance terminal

(Note If you need deploy without slave then install in Jenkins server)

click below Github url for installation

https://github.com/SaravanaNani/react.js.git

In Slave Terminal: Set password to sudo user:

- passwd sudo

enterpassword:

re-enterpassword:

In Jenkins UI: Add above Jenkins password in credentials: Kind (secret text)

```
pipeline {

  agent {

    label 'slave1'

  }

    environment {

    MY_PASSWORD = credentials('SLAVE1 SUDO USER CREDENTIAL ID')

  }

  stages {

    stage('Checkout') {
```

```
    steps {
      git branch: 'master', url: ' https://github.com/SaravanaNani/react.js.git '
    }
  }
  stage('Build') {
    steps {
      sh '''
      npm install
      npm run build
      '''
    }
  }
  stage('deploy') {
    steps {
      sh '''
      echo $MY_PASSWORD | sudo -S cp -r build/* /var/www/html
      '''
    }
  }
  stage('Zip') {
    steps {
```

```
                sh 'zip -r package_release.zip build/'
            }
        }
    }
}
```

4. **Create CICD Pipeline to build and deploy reactjs Application with the multiple branch pipeline.**

When i execute master branch, it should generate package_release.zip file (Expected output for master - package_release_1.X.X.zip) and

when i execute other branches, it should generate package_snapshot.zip file (Expected output for master - package_snapshot_1.X.X.zip)

Pass branch name using parameters: Build with parameters or you can directly pass the value i.e, master or main.

```
pipeline {
  agent any
  stages {
    stage('Checkout') {
      steps {
        script {
          def gitInfo = checkout([$class: 'GitSCM', branches: [[name: '*/${branch}']],
userRemoteConfigs: [[url: 'https://github.com/summu97/react.js.git']]])
          def branchName = gitInfo.GIT_BRANCH.tokenize('/')[1]
          echo "Branch name: ${branchName}"
        }
      }
    }
    stage('building code') {
      steps {
        sh '''
        npm install
        npm run build
```

```
            '''
          }
        }
        stage ('creating zip') {
          steps {
            script {
              def version = "1.${env.BUILD_NUMBER}"
              def gitInfo = checkout([$class: 'GitSCM', branches: [[name: '*/${branch}']],
userRemoteConfigs: [[url: 'https://github.com/summu97/react.js.git']]])
              def branchName = gitInfo.GIT_BRANCH.tokenize('/')[1]
              def fileName = branchName == 'master' ? 'package_release' :
'package_snapshot'
              fileName += "_${version}.zip"
              sh "zip -r ${fileName} build/"
            }
          }
        }
      }
}
```

5. How Can Jenkins be Configured to Trigger Job Failures in Case of Environment Differences?

- **Build should be failed if environment is different**

Step1 – Add Parameters in Jenkins Job

This Project is Parameterized ->Choice Parameters ->Name=ENVIRONMENT

->Choices (dev, test, prod).

NOTE: Modify the code below according to your requirement.

Post build actions: Events that happen after build is Scuuess or Failure or Always.

```
pipeline {
  agent any
   tools {
     maven 'maven' // Use the name configured in Jenkins
  }
  stages{
     stage('Checking Environment') {
        steps {
           script {
              def targetEnvironment = env.ENVIRONMENT
              if (targetEnvironment != 'dev') {
                 error "wrong environment selected "
              }
           }
        }
```

```
    }
    stage('checkout') {
        steps {
            git 'https://github.com/SaravanaNani/jenkins-java-project.git'
        }
    }
    stage ('compile') {
        steps{
            sh 'mvn compile'
        }
    }
    stage ('test') {
        steps {
            sh 'mvn test'
        }
    }
    stage ('artifact') {
        steps {
            sh 'mvn clean package'
        }
    }
}
```

```
    post {
        failure {
            echo "please select the correct environment"
        }
    }
}
```

6. How do I set up a Multi-Branch Pipeline?

- **Setting Up a Multi-Branch Pipeline: A Step-by-Step Guide**

Pre-requesites: You need to have a repository with multiple branches and Jenkinsfile's.

1. Create a Repository in GitHub with name multi
2. Create 2 branches (master and multi) and
3. In master branch and multi branch have a file in it as Jenkinsfile

Note: both the branches should have file named as Jenkinsfile

4. Now paste the below example Pipeline code in Github Jenkinsfile of both branch

Pipeline for Master branch Jenkinsfile:

```
pipeline {
  agent any
  stages {
    stage('multi') {
      steps {
        sh 'touch file1'
      }
    }
  }
}
```

Pipeline for Multi branch Jenkinsfile:

```
pipeline {
  agent any
  stages {
    stage('multi') {
            steps {
               sh 'touch file2'
            }
         }
      }
   }
```

Workflow:

Step1 – Jenkins Dashboard ->New-item (give name) ->Multi Branch Pipeline

Step2 – Display Name (give it) -> Branch Sources (Git) -> Project Repository (give your repo url) ->Build Configuration -> Script Path (specify correct path for your Jenkinsfile) (mutli/Jenkinsfile - for above repo) -> save

NOTE: You need to have your pipeline code in 'Jenkinsfile'(J- capital)

7. How can I seamlessly integrate Google Cloud Platform with Jenkins using Terraform?

Integrating Google Cloud Platform with Jenkins Using Terraform:

Step1 – Install terraform in Jenkins VM – terminal using the script below. (copy & paste)

wget -O- https://apt.releases.hashicorp.com/gpg | sudo gpg --dearmor -o /usr/share/keyrings/hashicorp-archive-keyring.gpg

echo "deb [signed-by=/usr/share/keyrings/hashicorp-archive-keyring.gpg] https://apt.releases.hashicorp.com $(lsb_release -cs) main" | sudo tee /etc/apt/sources.list.d/hashicorp.list

sudo apt update && sudo apt install terraform

Step2 – Login to Jenkins Console and Install plugin - Terraform

Step3 – Goto IAM – Select the Default Compute Engine- SA - Assign the required Roles for creation of resources using Terraform - (general purpose Assign –OWNER) & Generate the JSON Key

Step4 – Setup Your Jenkins Console & create a Pipeline Job.

Step5 – Manage Jenkins -> Credentials -> Global -> Add Credentials -> kind (Google Service Account from Private Key) -> project ID -> select JSON Key.

Step6 – Parameterize this Job

-> Choice Parameter -> Key: action | Value: apply, destroy

Note: Fork the below given repository and change the project ID

```
pipeline {
  agent any
    environment {
    GOOGLE_CREDENTIALS = credentials('JSON-KEY- CREDENTIAL_ID')
  }
  stages {
    stage('Checkout') {
      steps {
        git branch: 'main', url: 'https://github.com/SaravanaNani/terra-sample.git'
      }
    }
    stage('init, plan, apply') {
      steps {
        sh '''
        terraform init
        terraform plan
        terraform ${action} --auto-approve
        '''
      }
    }
  }
}
```

8. **How can I configure a Jenkins Pipeline to store artifacts in Nexus, and get the war file from Nexus and deploy Java applications to Tomcat, and enable email notifications after the build using email configuration?**

Automating Artifact Storage in Nexus, Java Application Deployment in Tomcat, and Email Notifications with Jenkins Pipeline

Step1 – Create VM-Instance for Nexus and Install nexus in the Instance using script below (Note: use e2 medium, 30 gb, Ubuntu-OS)

vim nexus.sh - (copy & paste script here) and to Run the Script (sh nexus.sh)

```
sudo apt update && sudo apt upgrade -y

sudo apt install openjdk-8-jdk -y

sudo mkdir /app && cd /app

sudo wget https://download.sonatype.com/nexus/3/nexus-3.67.1-01-java8-unix.tar.gz

sudo tar -xvf nexus-3.67.1-01-java8-unix.tar.gz

sudo mv nexus-3.67.1-01 nexus

sudo useradd nexus

sudo passwd nexus

sudo chown -R nexus:nexus /app/nexus

sudo chown -R nexus:nexus /app/sonatype-work

echo 'run_as_user="nexus"' | sudo tee /app/nexus/bin/nexus.rc
```

Step2 – Start the nexus using the below command in nexus Instance SSH terminal

- /app/nexus/bin/nexus start
- /app/nexus/bin/nexus status

Step3 – Create a Jenkins Server and process the installation of Jenkins and Tomcat as mentioned in above 1st Scenario. Make sure to change the Port Number for Jenkins.

Setp4 – In your Gmail console -> click on Gmail Icon -> Manage your google account

->security -> enable 2-step Verification (go through steps and turn it on)

Now search for '**app passwords**'-->app name(**jenkins**)-->it will generate a password save it.

Step5 – Now Login to Jenkins console -> Manage Jenkins -> Plugin -> Install (**Nexus Artifact Uploader & Deploy to container**)

Step6 – In Jenkins Console -> Manage Jenkins -> System -> E-mail Notification:

-->SMTP server (smtp.gmail.com)

-->Advanced -> Use SMTP Authentication (select it)

-->Username (give your email)

-->Password (provide password that you generated in gmail)

-->Use SSL (select it)

-->SMTP Port (465)

-->Test configuration by sending test e-mail

-->Test e-mail recipient (give your email here) -->Test configuration (click on it)

NOTE: you must get 'Email was successfully sent'

Step7 ->Manage Jenkins -->System -->Extended E-mail Notification:

SMTP server(smtp.gmail.com)

-->SMTP Port (465)

-->Advanced -->Credentials -> add -> Jenkins -> kind (username and password)

-->Username (give configured email)

-->password (provide password that you generated in gmail)

-->id(email) --> description(optional) -->ADD -->select added credential

-->use ssl (click on it)

-->go through all other sections like Default Recipients (who will get notified)

-->Reply To List (who they can reply to)

Step8 – Create a Pipeline Syntax for Nexus Stage -> click on -> Pipeline Syntax

-->Sample Step (Nexus Artifact Uploader)

-->Nexus Version (NEXUS3)

-->Protocol (HTTP)

-->Nexus URL (copy & Paste nexus EXTERNALIP:8081)

--> Credentials -> Add (Jenkins) ->kind (Username & Password)

->Username (nexus-username [admin]) Password (nexus login password)

->Description(nexus) -> Add

Note: The below given details are for the Jave application Code used here in this Task. Below Details vary for other Java application Code, you find it in POM.XML file

--> GroupID (in.RAHAM)

--> Version (1.2.2)

--> Repository (give the repo name created in Nexus console: war-file)

To create Repository:

-> Login to Nexus Console (using nexus VM - ExternalIP:8081)

-> Initial Username & Password = admin -> Click on Setting Icon -> Repositories

-> Create repositories -> (maven2 hosted) -> Repo Name (war-file)

-> Deployment Policy Select (Allow Redeploy) -> Create Repository

----

--> Click on Artifacts -> ArtifactId (NETFLIX) -> Type (.war) -> Classifier (leave it blank)

-> File (target/NETFLIX-1.2.2.war) -> Generate Pipeline Syntax (copy & paste in pipeline stage nexus)

Step9 – Create a Pipeline Syntax for Email configuration -> click on -> Pipeline Syntax

--> Sample Step (emailext: extented Email) -> To (give Email ID) -> Subject & body (write the subject and body for the mail here)

NOTE: Sample Pipeline is Mentioned in next pages

```
pipeline {
  agent any
  tools {
    maven 'maven' // Use the name configured in Jenkins
  }
  stages {
    stage('checkout') {
      steps {
        git 'https://github.com/summu97/monolythic-project.git'
      }
    }
    stage('package') {
      steps {
        sh 'mvn clean package'
      }
    }
    stage('nexus') {
      steps {
        nexusArtifactUploader artifacts: [[artifactId: 'NETFLIX', classifier: '', file: 
'target/NETFLIX-1.2.2.war', type: '.war']], credentialsId: '$NEXUS-CREDENTIAL-ID-', 
groupId: 'in.RAHAM', nexusUrl: '$NEXUS-URL', nexusVersion: 'nexus3', protocol: 'http', 
repository: '$REPO-NAME, version: '1.2.2'
      }
    }
    stage('getting war') {
      steps {
          sh 'wget --user=admin --password=nexus 
http://34.118.95.240:8081/repository/nexus/in/RAHAM/NETFLIX/1.2.2/NETFLIX-
1.2.2..war'
```

```
            }
        }
         stage('deploy') {
            steps {
                deploy adapters: [
                    tomcat9(
                        credentialsId: 'TOMCAT-CREDENTIALID',
                        path: '',
                        url: '$TOMAT-URL'
                        )
                    ],
                    contextPath: 'netflix',
                    war: './*.war'
            }
        }
        stage('email'){
            steps {
                emailext body: 'Success', subject: 'Build status', to: 'example22@gmail.com'
            }
        }
    }
}
```