01

Developers build, test & push the code to VCS git. Code is getting merged on VCS git regularly, after sometime when code is together build & tested it generate bugs & conflicts. So we need integration to reslove these errors & bugs regular as code is getting merged

**Continuous integration : Whenever developer commit the code, it is build & tested automatically if anything wrong developer gets notified & developer fix the code. This process called continuous integration.**

**Jenkins is continuous integration tool.Jenkins is opensource & extensible**(lots of plugins like VCS, Build, Cloud, testing plugin) these are 2 main features of Jenkins.

Insert image 25 26 27 from F:\Notepad++\_Images

**Whenever code change by the developer jenkins fetch the code build the code evaluate it & notifies to the developer.**

Pre-requisite : 1.JAVA, JRE, JDK 2. Any OS

**Continuous Delivery - deployment to the production environment is a manual decision**

**Continuous Deployment – It is full automation & deployment to the production environment also automated**

02

**Jenkins is a Java-based application, so you'll need to have a compatible JRE installed on your system.** **Jenkins 2.277.4 and later require Java 11.Min 2GB RAM & 10 GB hard disk space.Default port 8080.**

Minimum hardware requirements: 256 MB of RAM 1 GB of drive space (although 10 GB is a recommended minimum if running Jenkins as a Docker container)

Recommended hardware configuration for a small team: 4 GB+ of RAM 50 GB+ of drive space

Launch EC2 with name JenkinsServer > Ubuntu 20.04 LTS free > t2.micro(t2.small) > key : jenkins-key pem > SG Jenkins-SG : JenkinSG ssh 22 MyIP, Custom TCP 8080 MyIP > Userdata paste script below

**#/bin/bash**

**sudo apt get update**

**sudo apt install openjdk-11-jdk openjdk-8-jdk -y**

**sudo apt install maven -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**

**systemctl status jenkins**

curl publicIP/latest/user-data : to check user data script

**/var/lib/jenkins home directory of jenkins** all the data of jenkins in this directory

ssh public ip with 8080 > copy password from **/var/lib/jenkins/secrets/initialAdminPassword** > Select plugins manually > Uncheck Ant Check NodeJS > Username : admin Password > Start jenkins

03

**JOBS : jobs means workloads which jenkins runs for you**

1.**Freestyle : Freestyle jobs are graphical jobs without pipeline.**If you want to do same job in other project you need to do all process again

2.**Pipeline as a code : Create pipeline as a code in groovy language**

Forgot jenkins password : **vi /var/lib/jenkins/config.xml** > change **usersecurity from true to false** > systemctl restart jenkins

**Manage Jenkins > Security > Security Realm : Jenkins’ own user database > Authorization : Logged-in users can do anything** > check **Allow anonymous read access** > save

We will redirected to Create new user page > admin > password

04

**Global tool configuration contains the tools that we will use for our code**

**java --version** : we see openjdk version 11.0.16, **jenkins runs on openjdk 11**

To run jenkins on EC2 we need java-1.11.0 yum install openjdk-11-jdk -y

**To run the jenkins jobs on jenkins server we need java-1.8.0** yum install openjdk-8-jdk -y

**ls /usr/lib/jvm**

copy path of openjdk-8-jdk : **/usr/lib/jvm/java-1.8.0-openjdk-amd64**

We are specifying the name & path of the tool that we are going to use in our code :

**jenkins > manage jenkins > Tools > add jdk > OracleJDK8 > Uncheck install automatically > JAVA\_HOME : /usr/lib/jvm/java-1.8.0-openjdk-amd64**

**Add maven > Name : MAVEN3 > check install automatically**

05

New item > Build > freestyle project > Ok

It contains :

**General**

**Source Code Management**

**Build Triggers**

**Build Environment**

**Build Steps**

**Post-build Actions**

> Description : vprofile maven build project > Source code management : Git > https://github.com/devopshydclub/vprofile-project.git (credentials used for private repository) > \*/vp-rem > Build step > Add build step > Execute shell > mvn install OR best way is below

> Build step > Add build step > Invoke top-level Maven targets > Maven version: MAVEN3 > Goals: install > save

We have installed maven plugin already so we Invoke top-level Maven targets coz its better to use installed plugins instead of shell commands.

Dashboard > Build > Build now > check console output Finished: SUCCESS

Dashboard > **Workspace : temporary directory that is unique for each build and is typically used for tasks such as checking out source code from a version control system, compiling code, running tests, and generating artifacts.** **Also it stores project's files during the build process.Maven creates artifact inside target directory of workspace vprofile.war. Dont store data on workspace, coz we clean workspaces when there is an issue**

> Build > Configure > Post build actions > Files to archive : \*\*/\*.war > Build now > refresh page > we will see Last Successful Artifacts vprofile.war

Artifact is an output generated after a Maven project build. It can be a jar, war or any other executable file.Maven artifacts include five key elements groupId, artifactId, version, packaging, and classifier.

06

apt remove maven -y

Dashboard > New item > Build-test > Freestyle project > Copy from Build > OK > save > Build now

We see **mvn install FATAL: command execution failed. It is due to we uninstalled maven on EC2**

apt install maven -y

**Having plugin or feature in jenkins doesnt mean we can run the tool. We need to have the tool. One way is install tool in OS. 2nd way is Dashboard > Tools > Add maven > Name : Maven-3.6.3 > Version : 3.6.3 > save**

Build-test > Build steps > select version MAVEN-3.6.3 > save > Build now

In console log we see maven version 3.6.3 used

07

Workspace :

**Everytime we run the job it creates new artifacts replacing old one.** Let us say the job will run for every code change i.e for every code change new artifacts gets created by replacing old one.We deployed new artifact to the server & things break. To make it stable we need old artifact but its already replaced. **So to preserve old artifacts we do versioning through 1.shell commands 2.installing plungins.**

1.shell commands :

Dashboard > New item > Versioning-Builds > Freestyle project > copy from Build > OK > Remove Post build actions > save > Build now : we see artifact created in workspace target directory

Versioning-Builds > Configure > Build > Add build step > Execute shell >

mkdir -p versions

cp target/vprofile-v2.war versions/ vprofile-V$BUILD\_ID.war #BUILD\_ID is jenkins enviornment variable which returns build id > save

Build now > 2-3 times build now #check workspace versions directory we see different versions of artifact

What if i want to use my own artifact like a user ?

Versioning-Builds > General > This project is parameterized > Add parameter > String parameter > Name: VERSION > Build > Add build step > Execute shell >

mkdir -p versions

cp target/vprofile-v2.war versions/ vprofile-V$VERSION.war > save

Versioning-Builds > Build with parameters > VERSION : 4.5 > Build

check workspace versions directory we see artifact of vesrion 4.5

2.installing plungins :

Dashboard > Manage jenkins > Plugins > Available > Zentimestamp > install without restart

If you dont find plungins i.e you are behind proxy server. We need to use Advanced options in plugins & configure proxy server > restart jenkins IP:8080/restart > yes

Versioning-Builds > Uncheck This project is parameterized > Check Change date pattern for the BUILD\_TIMESTAMP > yy-MM-dd-HH-mm > Execute shell

mkdir -p versions

cp target/vprofile-v2.war versions/ vprofile-V$BUILD\_ID-$BUILD\_TIMESTAMP.war > save > Build now 2-3 times

check workspace versions directory we see artifacts with build id & timestamp

08

Insert image 25 26 27 28 from F:\Notepad++\_Images

We are understanding the flow by using these tools primarily, Jenkins then we have git, Maven, sonarqube, nexus. So we have the developer. The developer's job is to write the code and developer will write the code, make the changes to the code test it locally. If they are good with the changes, they will push it to a centralized repository like GitHub. So developers will have git tool which will integrate with GitHub repository and the code will be committed to GitHub repository. As soon as there is a code change. Jenkins will detect a change and fetch the code by using git tool. So Jenkins will have git tool and git plugin, which will help accomplish this task to fetch the code, whenever there is a change. After that in the pipeline, the code will be bulid. We will be using MAVEN to build the code because we have Java code and our code can be built with MAVEN tool, but it could be any other source code and other build tools as well. Once the build completes, it will generate artifacts. Next we will conduct unit test again by using Maven. Maven will have some unit testing framework that developer will use. Unit testing will be part of your source code. So being a DevOps, you don't need to do much here. You just need to execute some steps that will run this test and generate reports mostly in xml format. Once you have the reports ready, we will conduct another kind of test called us code analysis. Now unit test checks whether the unit of the code works or not. Code analysis checks if the code has any vulnerability, are you following the best practices? Do you have any bug in the code? And there are many other parameters on which code analysis will judge your code. We will be using sonarQube scanner to scan the code. Also, we will be using checkstyle. So there are many code analysis tools available in the market. We are using sonarQube scanner and checkstyle to scan the code and this will generate reports in a xml format. These reports will be uploaded to Sonar qube server. In sonarqube, You can have proper graph, charts and you can see what are the bugs, vulnerabilities and many other things in your code. We can also set a quality gate and we can say, if my code does not follow these practices, then fail the build. And if it fails, the pipeline will stop. If it passes. We have then a verified copy of the artifact.

So we build the code, we test the code, we analyze the code. And now we can distribute the artifact to be deployed on the servers. But before deploying it to the server, these artifacts will be versioned and will be uploaded to Nexus sona type repository. Now all this pipeline will be happening in Jenkins. As I said previously, you may have a different tool in later like you may have GitLab, circleCI, Bamboo there are many CI tools, but the process will be almost same.

Fetch the code, build the code, test it, analyze it and then publish the artifact. Whatever CI tool you're using, you have to integrate it with other tools like GitHub, SonarQube Nexus or any other tool.

09

Setup Jenkins Nexus SonarQube SecurityGroup

Install Plugins

Integrate Nexus with jenkins

Integrate SonarQube with jenkins

Write pipeline script

Set notification

10

Launch EC2 > JenkinsServer > Ubuntu 20.04 > t2.small > jenkinskey - pem > JenkinsSG : ssh MyIP, CustomTCP 8080 Anywhere, CustomTCP 80 Anywhere - SonarQube is going to connect to jenkins for quality gate check on port 80 > In UserData :

**#!/bin/bash**

**sudo apt update**

**sudo apt install openjdk-11-jdk openjdk-8-jdk -y**

**sudo apt install maven -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**

Launch EC2 > Nexus > CentOS 7 > t2.medium > NexusKey - pem > NexusSG : ssh MyIP, CustomTCP 8081 Anywhere, Nexus will be running service on port 8081 which will accessed from the broswer & jenkins will access it through command line also (we can set up a specific rule allowed from jenkins security group JenkinsSG) > In UserData :

#!/bin/bash

yum install java-1.8.0-openjdk.x86\_64 wget -y

mkdir -p /opt/nexus/

mkdir -p /tmp/nexus/

cd /tmp/nexus/

NEXUSURL="https://download.sonatype.com/nexus/3/latest-unix.tar.gz"

wget $NEXUSURL -O nexus.tar.gz

sleep 10

EXTOUT=`tar xzvf nexus.tar.gz`

NEXUSDIR=`echo $EXTOUT | cut -d '/' -f1`

sleep 5

rm -rf /tmp/nexus/nexus.tar.gz

cp -r /tmp/nexus/\* /opt/nexus/

sleep 5

useradd nexus

chown -R nexus.nexus /opt/nexus

cat <<EOT>> /etc/systemd/system/nexus.service

[Unit]

Description=nexus service

After=network.target

[Service]

Type=forking

LimitNOFILE=65536

ExecStart=/opt/nexus/$NEXUSDIR/bin/nexus start

ExecStop=/opt/nexus/$NEXUSDIR/bin/nexus stop

User=nexus

Restart=on-abort

[Install]

WantedBy=multi-user.target

EOT

echo 'run\_as\_user="nexus"' > /opt/nexus/$NEXUSDIR/bin/nexus.rc

systemctl daemon-reload

systemctl start nexus

systemctl enable nexus

Working userdata for nexus sonatype :

#!/bin/bash

# Remove existing Java packages

yum remove java\* -y

# Install Java 1.8

sudo yum install java-1.8.0-openjdk.x86\_64 -y

# Change directory to /opt

cd /opt

# Download Nexus

wget https://sonatype-download.global.ssl.fastly.net/nexus/3/nexus-3.0.2-02-unix.tar.gz

tar -zxvf nexus-3.0.2-02-unix.tar.gz

mv nexus-3.0.2-02 /opt/nexus

# Create Nexus user

sudo adduser nexus

# Grant sudo privileges without a password prompt

echo "nexus ALL=(ALL) NOPASSWD: ALL" | sudo tee -a /etc/sudoers

# Change ownership of Nexus installation directory

sudo chown -R nexus:nexus /opt/nexus

# Modify nexus.rc

echo 'run\_as\_user="nexus"' | sudo tee /opt/nexus/bin/nexus.rc

# Create symbolic link for init.d

sudo ln -s /opt/nexus/bin/nexus /etc/init.d/nexus

# Start Nexus service

sudo su - nexus -c "service nexus start"

Launch EC2 > SonarServer > Ubuntu 18.04 > t2.medium > SonarKey - pem > SonarSG : ssh MyIP, CustomTCP 80 Anywhere, CustomTCP 9000 Anywhere : **SonarQube runs the service on port 9000 but we have nginx service in this instance which will forward the request to sonarQube service.** In userdata One of the property in sonar.properties file is sonar.web.port=9000 so sonar service runs on port 9000 but we are also setting nginx service in this instance which runs on port 80. **Nginx configuration says listen on port 80(line listen 80) & route the request to port 9000 locally (line proxy\_pass http://127.0.0.1:9000). So we can directly access sonarQube on port 80 or 9000.** Since we have nginx we are accessing sonarQube on port 80 > In UserData :

#!/bin/bash

cp /etc/sysctl.conf /root/sysctl.conf\_backup

cat <<EOT> /etc/sysctl.conf # set various kernel parameters,

vm.max\_map\_count=262144

fs.file-max=65536

ulimit -n 65536

ulimit -u 4096

EOT

cp /etc/security/limits.conf /root/sec\_limit.conf\_backup

cat <<EOT> /etc/security/limits.conf #set user-level resource limits

sonarqube - nofile 65536 #sets limits for the number of open files

sonarqube - nproc 409 #sets limits for the number of processes

EOT

sudo apt-get update –y #ensures that the package manager has the latest information about available packages

sudo apt-get install openjdk-11-jdk -y

sudo update-alternatives --config java #configures the default Java version on the system

java -version

sudo apt update

wget -q https://www.postgresql.org/media/keys/ACCC4CF8.asc -O - | sudo apt-key add -

sudo sh -c 'echo "deb http://apt.postgresql.org/pub/repos/apt/ `lsb\_release -cs`-pgdg main" >> /etc/apt/sources.list.d/pgdg.list'

sudo apt install postgresql postgresql-contrib -y

#sudo -u postgres psql -c "SELECT version();"

sudo systemctl enable postgresql.service

sudo systemctl start postgresql.service

sudo echo "postgres:admin123" | chpasswd

runuser -l postgres -c "createuser sonar" #PostgreSQL is configured by creating a new user ("sonar")

sudo -i -u postgres psql -c "ALTER USER sonar WITH ENCRYPTED PASSWORD 'admin123';"

sudo -i -u postgres psql -c "CREATE DATABASE sonarqube OWNER sonar;" #PostgreSQL is configured by creating a database ("sonarqube")

sudo -i -u postgres psql -c "GRANT ALL PRIVILEGES ON DATABASE sonarqube to sonar;"

systemctl restart postgresql

#systemctl status -l postgresql

netstat -tulpena | grep postgres #checks whether PostgreSQL is listening on network ports

sudo mkdir -p /sonarqube/

cd /sonarqube/

sudo curl -O https://binaries.sonarsource.com/Distribution/sonarqube/sonarqube-8.3.0.34182.zip

sudo apt-get install zip -y

sudo unzip -o sonarqube-8.3.0.34182.zip -d /opt/

sudo mv /opt/sonarqube-8.3.0.34182/ /opt/sonarqube

sudo groupadd sonar

sudo useradd -c "SonarQube - User" -d /opt/sonarqube/ -g sonar sonar

sudo chown sonar:sonar /opt/sonarqube/ -R

cp /opt/sonarqube/conf/sonar.properties /root/sonar.properties\_backup

# <<EOT indicates the start of the "here document," and **EOT** marks the end.

cat <<EOT> /opt/sonarqube/conf/sonar.properties

sonar.jdbc.username=sonar

sonar.jdbc.password=admin123

# Specifies the URL for the SonarQube database connection, assuming a PostgreSQL database running on the local

#machine.

sonar.jdbc.url=jdbc:postgresql://localhost/sonarqube

# means that SonarQube will listen on all available network interfaces

sonar.web.host=0.0.0.0

sonar.web.port=9000

sonar.web.javaAdditionalOpts=-server

sonar.search.javaOpts=-Xmx512m -Xms512m -XX:+HeapDumpOnOutOfMemoryError

sonar.log.level=INFO

sonar.path.logs=logs

EOT

cat <<EOT> /etc/systemd/system/sonarqube.service

[Unit]

Description=SonarQube service

After=syslog.target network.target

[Service]

Type=forking

ExecStart=/opt/sonarqube/bin/linux-x86-64/sonar.sh start

ExecStop=/opt/sonarqube/bin/linux-x86-64/sonar.sh stop

User=sonar

Group=sonar

Restart=always

LimitNOFILE=65536

LimitNPROC=4096

[Install]

WantedBy=multi-user.target

EOT

systemctl daemon-reload

systemctl enable sonarqube.service

#systemctl start sonarqube.service

#systemctl status -l sonarqube.service

apt-get install nginx -y

rm -rf /etc/nginx/sites-enabled/default

rm -rf /etc/nginx/sites-available/default

cat <<EOT> /etc/nginx/sites-available/sonarqube

server{

listen 80;

server\_name sonarqube.groophy.in;

access\_log /var/log/nginx/sonar.access.log;

error\_log /var/log/nginx/sonar.error.log;

proxy\_buffers 16 64k;

proxy\_buffer\_size 128k;

location / {

proxy\_pass http://127.0.0.1:9000;

proxy\_next\_upstream error timeout invalid\_header http\_500 http\_502 http\_503 http\_504;

proxy\_redirect off;

proxy\_set\_header Host \$host;

proxy\_set\_header X-Real-IP \$remote\_addr;

proxy\_set\_header X-Forwarded-For \$proxy\_add\_x\_forwarded\_for;

proxy\_set\_header X-Forwarded-Proto http;

}

}

EOT

ln -s /etc/nginx/sites-available/sonarqube /etc/nginx/sites-enabled/sonarqube

systemctl enable nginx.service

#systemctl restart nginx.service

sudo ufw allow 80,9000,9001/tcp

echo "System reboot in 30 sec"

sleep 30

reboot

11

Login JenkinsServer & check status of jenkins

ssh -i jenkinskey.pem ubuntu@publicIPOfJenkinsServer

sudo -i

systemctl status jenkins

If any error take raw link of jenkins-setup.sh &

wget https://github.com/devopshydclub/vprofile-project/blob/ci-jenkins/userdata/jenkins-setup.sh

/bin/bash jenkins-setup.sh

ssh -i SonarKey.pem ubuntu@publicIPOfSonarServer

sudo -i

systemctl status sonarqube

ssh -i NexusKey.pem centos@publicIPOfNexusServer

sudo -i

systemctl status nexus

Check Jenkins running > In broswer : publicIPOfJenkinsServer:8080

In broswer : publicIPOfSonarServer:80

In broswer : publicIPOfNexusServer:8081

Go to https://github.com/devopshydclub/vprofile-project.git > switch branch ci-jenkins > userdata > jenkins-setup.sh,

nexus-setup.sh, sonar-setup.sh use these scripts to launch 3 EC2 instances with OS viz Ubuntu 18, Centos7 & Ubuntu 18

1.JenkinsServer : Ubuntu 20.04 > t2.small > JenkinsKey > JenkinSG ssh 22 MyIP Custom TCP 8080 MyIP

Custom TCP 80 Anywhere : Sonarqube will connect jenkins for quality gate check on port 80 Custom TCP 8080 Anywhere > userdata paste jenkins-setup.sh

2.NexusServer : Centos7 > t2.medium > NexusKey > NexusSG > ssh 22 MyIP

Custom TCP 8081 Anywhere : Nexus run on port 8081 access from browser, we can use JenkinSG instead of Anywhere > userdata paste nexus-setup.sh

3.SonarServer : Ubuntu 18.04 > > t2.medium > SonarKey > SonarSG

ssh 22 MyIP Custom TCP 80 Anywhere : sonarqube runs on port 9000, but we have nginx service which listen on port 80 & route the request to sonarqube service on port 9000 locally in this instance In sonar-setup.sh > sonar.web.port=9000 sonar runs on 9000. But we have nginx service that runs on port 80 "apt-get install nginx -y server { listen 80" & route request on 9000 locally location / { proxy\_pass http://127.0.0.1:9000 Custom TCP 9000 Anywhere > userdata paste sonar-setup.sh

systemctl status jenkins

if service not running go to jenkins-setup.sh in github > Raw > copy link > paste it in JenkinsServer wget link

download the file jenkins-setup.sh run it /bin/bash jenkins-setup.sh

systemctl status sonarqube

systemctl status nexus

Admin admin Admin123

JenkinsServer : in browser publicIP:8080 > ssh JenkinsServer > copy password from /var/lib/jenkins/secrets/initialAdminPassword & paste

SonarServer : in browser publicIP:80 > ssh SonarServer > username admin password admin NexusServer : in browser publicIP:8081 > ssh NexusServer > username admin copy password from /opt/nexus/sonatype-work/nexus3/admin.password & paste. Also reset the password

Jenkins > manage jenkins > manage plugins > **Install Nexus Artifact Uploader, SonarQube Scanner, Build Timestamp (this will used for version our artifact), Pipeline Maven Integration, Pipeline Utility Steps** > Install without restart

013

Pipeline as code means setting up pipeline automatically by using Jenkinsfile. Jenkinsfile define stages in CI/CD pipeline

Jenkins file has 2 syntax : Scripted, Declarative(We use this)

**Pipeline code : pipeline > agent > tools > environment > Stage > environment > steps > post**

**Scripted Pipelines offer more flexibility and control but are more complex. Declarative Pipelines are easier to start with, especially for simple use cases, and promote best practices.**

Pipeline : block

agent/Node : **where pipeline job get executed on which node or agent**

tools : **from global tool configuration mention tools like sonic scanner, maven, JDK**

enviornment : **set enviornment variables to use them in steps(of stages)**

Stage : **Actual execution happens**

enviornment :

steps : steps are commands

post : post installation steps like success

FETCH Build & test code using pipeline :

Before this go to Dashboard > Manage jenkins >

1. JDK > Add jdk > OracleJDK8 > JAVA\_HOME : /usr/lib/jvm/java-1.8.0-openjdk-amd64 > Uncheck Install automatically

2. Maven > MAVEN3 > Save

**pipeline {**

**// Define the agent where the pipeline will run.**

**agent any**

**// Define the tools used in this pipeline.**

**tools {**

**// Use the specified Maven tool named "MAVEN3."**

**maven "MAVEN3"**

**// Use the specified JDK tool named "OracleJDK8."**

**jdk "OracleJDK8"**

**}**

**// Define the stages of the pipeline.**

**stages {**

**// Stage 1: Fetching code from a Git repository.**

**stage('Fetch code') {**

**steps {**

**// Use Git to clone the repository and specify the branch.**

**git branch: 'vp-rem', url: 'https://github.com/devopshydclub/vprofile-repo.git'**

**}**

**}**

**// Stage 2: Building the code.**

**stage('Build') {**

**steps {**

**// Run Maven to build the code, skipping tests.**

**sh 'mvn install -DskipTests'**

**}**

**// Define post-build actions.**

**post {**

**// Execute this block on successful completion of the stage.**

**success {**

**echo 'Now Archiving it...'**

**// Archive built artifacts (WAR files) for future reference.**

**archiveArtifacts artifacts: '\*\*/target/\*.war'**

**}**

**}**

**}**

**// Stage 3: Running unit tests.**

**stage('UNIT TEST') {**

**steps {**

**// Run Maven to execute unit tests.**

**sh 'mvn test'**

**}**

**}**

**}**

**}**

format this code in chatgpt

Jenkin > new item > sample-paac > pipeline > OK > paste above script > Save > Build now

**Every build has workspace**. Click on build id > scroll down select workspaces/target/.war

014

Code analysis : Code analysis is one more type of test that is conducted on the code.

Code analysis check your code against best practices & find problems which further fixed by developers.

Vulnarabilities : Unknowingly developer create many vulnerabilities in the code(search top in OWASP vulnerabilities), using these vulnerabilities hackers can exploit application. code analysis will also check vulnerabilities. OWASP tool special for checking vulnerabilities. Code analysis also look for bugs & functional errors before deployment. To improve the code quality we do code analysis

TOOLS FOR CODE ANALYSIS : Checkstyle Coburtura mstest owasp SonarQube-Scanner

We are using SonarQube-Scanner & Checkstyle

**INTEGRATE SONARQUBE WITH JENKINS & SETUP SONAR SCANNER TOOL:**

manage jenkins > Tools > SonarQube Scanner > Add SonarQube Scanner > Name : sonar4.7 (this will be used in pipeline code ) > check install automatically > save

configure system > SonarQube servers > check enviornment variables > Add SonarQube > sonar > server url : http://publicIP of SonarServer EC2 > Server authentication token > (Go to browser sonarquebe server > My account > Security > token name : jenkins > Generate > Copy token) > Add (if token not seen save & come back again to configure system) > jenkins (token) > kind : secret text > Secret : paste token > ID : MySonarToken > Add

015 We are going to do code analysis using checkstyle & sonarqube

pipeline {

// Define the agent where the pipeline will run (in this case, "any" agent).

agent any

// Define the tools used in this pipeline (Maven and JDK).

tools {

// Use the specified Maven tool named "MAVEN3" (configured in Jenkins global tool configuration).

maven "MAVEN3"

// Use the specified JDK tool named "OracleJDK8" (configured in Jenkins global tool configuration).

jdk "OracleJDK8"

}

// Define the stages of the pipeline.

stages {

// Stage 1: Fetching code from a Git repository.

stage('Fetch code') {

steps {

// Use Git to clone the repository and specify the branch.

git branch: 'vp-rem', url: 'https://github.com/devopshydclub/vprofile-repo.git'

}

}

// Stage 2: Building the code.

stage('Build') {

steps {

// Run Maven to build the code, skipping tests.

sh 'mvn install -DskipTests'

}

// Define post-build actions.

post {

// Execute this block on successful completion of the stage.

success {

echo 'Now Archiving it...'

// Archive built artifacts (WAR files) for future reference.

archiveArtifacts artifacts: '\*\*/target/\*.war'

}

}

}

// Stage 3: Running unit tests.

stage('Test') {

steps {

// Run Maven to execute unit tests.

sh 'mvn test'

}

}

// Stage 4: Running Checkstyle analysis.

stage('Checkstyle Analysis') {

steps {

// Run Maven to perform Checkstyle analysis.

sh 'mvn checkstyle:checkstyle'

}

}

}

}

Jenkin > new item > Pipeline\_As\_A\_Code > pipeline > OK > paste above script > Save > Build now

google sonar scanner pipeline script

pipeline {

// Define the agent where the pipeline will run (in this case, "any" agent).

agent any

// Define the tools used in this pipeline (Maven, JDK, and SonarQube Scanner).

tools {

// Use the specified Maven tool named "MAVEN3" (configured in Jenkins global tool configuration).

maven "MAVEN3"

// Use the specified JDK tool named "OracleJDK8" (configured in Jenkins global tool configuration).

jdk "OracleJDK8"

}

// Define the stages of the pipeline.

stages {

// Stage 1: Fetching code from a Git repository.

stage('Fetch code') {

steps {

// Use Git to clone the repository and specify the branch.

git branch: 'vp-rem', url: 'https://github.com/devopshydclub/vprofile-repo.git'

}

}

// Stage 2: Building the code.

stage('Build') {

steps {

// Run Maven to build the code, skipping tests.

sh 'mvn install -DskipTests'

}

// Define post-build actions.

post {

// Execute this block on successful completion of the stage.

success {

echo 'Now Archiving it...'

// Archive built artifacts (WAR files) for future reference.

archiveArtifacts artifacts: '\*\*/target/\*.war'

}

}

}

// Stage 3: Running unit tests.

stage('Test') {

steps {

// Run Maven to execute unit tests.

sh 'mvn test'

}

}

// Stage 4: Running Checkstyle analysis.

stage('Checkstyle Analysis') {

steps {

// Run Maven to perform Checkstyle analysis.

sh 'mvn checkstyle:checkstyle'

}

}

// Stage 5: SonarQube Analysis.

stage('Sonar Analysis') {

environment {

// Define an environment variable for the SonarQube Scanner tool.

scannerHome = tool 'sonar4.7'

}

steps {

// Use the SonarQube Scanner tool to analyze the code.

withSonarQubeEnv('sonar') {

sh """

\${scannerHome}/bin/sonar-scanner \

-Dsonar.projectKey=vprofile \

-Dsonar.projectName=vprofile \

-Dsonar.projectVersion=1.0 \

-Dsonar.source=src/ \

-Dsonar.java.binaries=target/test-classes/com/visualpathit/account/controllerTest/ \

-Dsonar.junit.reportsPath=target/surefile-reports/ \

-Dsonar.jacoco.reportsPath=target/jacoco.exec \

-Dsonar.java.checkstyle.reportsPath=target/checkstyle-result.xml

"""

}

}

}

}

}

Jenkin > new item > PAAC-Analysis > pipeline > OK > paste above script > Save > Build now > SonarQube Quality gate Passed Success

Browser SonarServer > refresh > you will see bugs vulnerabilities code-smells etc

Browser SonarServer > Quality gates > Sonar way is default Quality gate

Browser SonarServer > project > project settings > Quality gate > Default Sonar way

016

pipeline {

agent any

tools { : must add in every pipeline important

maven "MAVEN3" : give name same as given in global tool configuration of jenkins

jdk "OracleJDK8"

}

stages {

stage('Fetch code') {

steps {

git branch: 'vp-rem', url:'https://github.com/devopshydclub/vprofile-repo.git'

}

}

stage('Build') {

steps{

sh 'mvn install -DskipTests'

}

post {

success {

echo 'Now Archiving it...'

archiveArtifacts artifacts: '\*\*/target/\*.war'

}

}

}

stage('Test') {

steps {

sh 'mvn test'

}

}

stage('Checkstyle Analysis'){

steps {

sh 'mvn checkstyle:checkstyle'

}

}

stage('Sonar Analysis'){

enviornment {

scannerHome = tool 'sonar4.7'

}

steps {

withSonarQubeEnv('sonar') {

sh '''${scannerHome}/bin/sonar-scanner -Dsonar.projectKey=vprofile \

-Dsonar.projectName=vprofile \

-Dsonar.projectVersion=1.0 \

-Dsonar.source=src/ \

-Dsonar.java.binaries=target/test-classes/com/visualpathit/account/controllerTest/ \

-Dsonar.junit.reportsPath=target/surefile-reports/ \

-Dsonar.jacoco.reportsPath=target/jacoco.exec \

-Dsonar.java.checkstyle.reportsPath=target/checkstyle-result.xml'''

}

}

}

stage("Quality Gate") {

steps {

timeout(time:1, unit:'HOURS') { # timeout means wait for 1 hour for quality gate

// set pipeline to UNSTABLE i.e true if Quality gate fails pipeline aborted

waitForQualityGate abortPipeline: true

}

}

}

}

}

Browser SonarServer > Quality gates > create > vprofile-QG > Add condition > On overall code > Quality gate fails when :

Bugs is greater than 60 > Add condition

Browser SonarServer > Projects > vprofile > Project settings > Quality gate > select vprofile-QG

SonarQube server sends the information about quality check to jenkins by using webhooks

Browser SonarServer > Projects > vprofile > Project settings > Webhooks > Create > jenkins-CI-webhook > URL : http://publicIPofJenkins:8080/sonarqube-webhook > create

Jenkins > PAAC-Analysis > configure > paste above code > save > Build Now

Refresh sonarqube in browser, it shows Failed, coz we have set bugs more than 60 means fail & we already have 82 bugs

Increase threshold of Bugs from 60 to 100, so our pipeline will pass & further developer fix these bugs when pipeline pass

OR change Quality gate back to Sonar way > Build now

017 upload artifact to nexus repository

Nexus OSS sonatype is software repository. We can download dependencies from Nexus repository

Types of repository :

Maven - store maven dependencies

apt - store packages of debian based system

yum - store packages of Redhat based system

nuget - store package manager for .net

Npm - store package manager for javascript

Docker - store docker images

**Nexus sonatype repository runs on java.We use it to store artifact & dependencies.It support all of the above types of repository.**

**Once artifact gets uploaded to nexus repository OPS team or automation script fetch that artifact & deploy it to server.**

Copy public IP of NexusServer > paste it in broswer with port 8081 > sign in > settings > Repositories > Create repository > \*maven2 (hosted) hosted is for storing artifact, proxy is for download dependencies, group is for group both repositories > vprofile-repo > Create repository

we are going to use pipeline as a code & store artifact to vprofile-repo repository in nexus sonatype

Jenkins > Manage jenkins > Credentials > Jenkins > Global credentials > Add credentials > admin passwordOfNexus

> ID : nexuslogin > ok

018

google search nexus artifact uploader pipeline https://github.com/jenkinsci/nexus-artifact-uploader-plugin

Jenkins > configure system > Build Timestamp > check Enable build timestamp > Pattern : yy-MM-dd\_HH-mm (this is for

BUILD\_TIMESTAMP variable in pipeline code) > save

pipeline {

agent any

tools { : must add in every pipeline important

maven "MAVEN3" : give name same as given in global tool configuration of jenkins

jdk "OracleJDK8"

}

stages {

stage('Fetch code') {

steps {

git branch: 'vp-rem', url:'https://github.com/devopshydclub/vprofile-repo.git'

}

}

stage('Build') {

steps{

sh 'mvn install -DskipTests'

}

post {

success {

echo 'Now Archiving it...'

archiveArtifacts artifacts: '\*\*/target/\*.war'

}

}

}

stage('Test') {

steps {

sh 'mvn test'

}

}

stage('Checkstyle Analysis'){

steps {

sh 'mvn checkstyle:checkstyle'

}

}

stage('Sonar Analysis'){

enviornment {

scannerHome = tool 'sonar4.7'

}

steps {

withSonarQubeEnv('sonar') {

sh '''${scannerHome}/bin/sonar-scanner -Dsonar.projectKey=vprofile \

-Dsonar.projectName=vprofile \

-Dsonar.projectVersion=1.0 \

-Dsonar.source=src/ \

-Dsonar.java.binaries=target/test-classes/com/visualpathit/account/controllerTest/ \

-Dsonar.junit.reportsPath=target/surefile-reports/ \

-Dsonar.jacoco.reportsPath=target/jacoco.exec \

-Dsonar.java.checkstyle.reportsPath=target/checkstyle-result.xml'''

}

}

}

stage("Quality Gate") {

steps {

timeout(time:1, unit:'HOURS') {

// set pipeline to UNSTABLE i.e true if Quality gate fails

waitForQualityGate abortPipeline: true

}

}

}

stage("UploadArtifact") {

steps {

nexusArtifactUploader(

nexusVersion: 'nexus3',

protocol: 'http',

nexusUrl: 'privateIPofNexusServer:8081',

groupId: 'QA',

version: "${env.BUILD\_ID}-${env.BUILD.TIMESTAMP}",

repository: 'vprofile-repo',

credentialsId: 'nexuslogin',

artifacts: [

[artifactId: 'vproapp',

classifier: '',

file: 'target/vprofile-v2.war',

type: 'war']

]

)

}

}

}

}

Jenkin > new item > vprofile-ci-pipeline > pipeline > OK > paste above script > save

Jenkins > manage jenkins > Configure system > SonarQube servers > server URL : http://privateIPofSonarServer > Server authentication token : MySonarToken > save

Browser sonarqube > project > vprofile > project settings > Webhooks > update > URL : http://privateIPofJenkins:8080/sonarqube-webhook> update Build now

Browser nexus sonatype > Browse server content(near settings) > you will see vproapp.war artifact > click on it > right side

click on link we can download it from here

Every time you build now new artifact gets created in nexus sonatype

019

login slack > Create workspace > vprofilecicd > devopscicd > email > use slack in broswer > Add channel > jenkinscicd > create > automatically add anyone who joins > save

020 CI for Docker :

Our previous continuous integration pipeline publishes artifacts, the war file. In this continuous integration pipeline, we will be publishing Docker images.

Please check Introduction to Containers section to see how Docker images are built and publish. We will be doing similar thing, but we will be doing it through pipeline as a code from Jenkins.

So **whenever developer makes a change to the code, it comes on the GitHub, Jenkins will detect it, fetch the code and will run the unit test. Then we will do code analysis with check style. And then again, code analysis with SonarQube. Upload the result to SonarQube server, wait for the quality gates, If everything is good, then we are going to build Docker image. The Docker image will contain the artifact. So this is the build process here. The build process is the Docker build process where we get the Docker image and we publish the Docker image to a registry. In this example, we will be using Amazon ECR, Elastic Container Registry. There could be here a GCR Google Container Registry or Azure Registry Service or Docker Hub or your own solution like Nexus.** In this example, we will see ECR because we are using AWS already. It doesn't matter where you publish your image. The only process that will change over here is the login process to your registry service. Otherwise all of this remains similar.

021

In short :

update-install depedencies-create directory keyrings-add official GPS keys of docker repository-setup docker repository-update-install docker-add jenkins user to docker group-install aws cli-reboot

Create IAM user jenkins with ecr & ecs access-create ECR repository-start jenkins & install plugins required(docker pipeline, ECR, AWS SDK, CloudBees Docker Build and Publish)-Manage-Credentials-Enter access & secret key of jenkins IAM user-run pipeline with fill in the blanks(change enviornment parameters only)

IMP : for VM aws configure the jenkins VM with access & secret keys of jenkins user

Remember we are fetching code from github & our dockerfile is there

We need :

IAM user with AWS ECR permissions

Store AWS credentials in jenkins

Create ECR repo on AWS

Plugin docker pipeline

Plugin ECR

Install docker engine on jenkins server

enviornment

registryCredential - used by jenkins authenticate ECR, 1.need IAM user with ECR permission & store aws credential in jenkins

appRegistry - URL for docker image in ECR

vprofileRegistry - URL of ECR registry, 2.Need ECR repo on AWS

stages : 1st fetch code from docker branch & run mvn test. There is no build stage over here for maven.Build stage will be done in docker (docker build will build artifact & image together using Dockerfile)

2nd checkstyle code analysis

3rd SonarQube server analysis

4th Quality gates

5th Build docker image : for docker.build we need to 3.install docker pipeline plugin & ecr plugin in jenkins,

for this pipeline we need 4.to install docker engine on jenkins server

docker pipeline plugin just execute the command docker.build to run the docker build we need to install docker engine on jenkins server

In script to run docker build command we passed arguments : 1.appRegistry (ECR registry URL) 2.BUILD\_NUMBER as tag which uses build id as image tag for versioning 3. path of the Dockerfile from source code vprofile-project

./Docker-files/app/multistage/ is path of Dockerfile

go to https://github.com/devopshydclub/vprofile-project.git > switch docker branch > Dockerfiles > app > multistage > Dockerfile. This is multistage Dockerfile which will gets executed to build our docker image.

This Dockerfile has 2 stages. In first stage it uses openJDK image, clones the source code & run mvn install which generates the artifact.

In the second stage Dockerfile takes tomcat image & copy artifact inside tomcat image & builds that image. This image will be published to Amazon ECR.

6th docker image is the artifact which will be uploaded to ECR.In a function docker.withRegistry we are passing registr URL & registry credentials which are AWS access key & secret key & docker.push will push docker image with BUILD\_NUMBER & latest as a tag to it

STEPS :

1. Install docker engine on jenkins server & add jenkins user to docker group coz jenkins user will execute docker build command & reboot

2.Install AWS cli

3.Create IAM user for access & secret key

4.Create ECR repo where we store docker images

5.Plugins : ECR, docker pipeline, aws sdk to store credentials in jenkins

6.Store AWS credentials in jenkins i.e access key & secret key

7.Run the pipeline

pipeline {

agent any

tools {

maven "MAVEN3"

jdk "OracleJDK8"

}

environment {

registryCredential = 'ecr:us-east-2:awscreds'

appRegistry = "951401132355.dkr.ecr.us-east-2.amazonaws.com/vprofileappimg"

vprofileRegistry = "https://951401132355.dkr.ecr.us-east-2.amazonaws.com"

}

stages {

stage('Fetch code') {

steps {

git branch: 'docker', url: 'https://github.com/devopshydclub/vprofile-project.git'

}

}

stage('Test') {

steps {

sh 'mvn test'

}

}

stage('CODE ANALYSIS WITH CHECKSTYLE') {

steps {

sh 'mvn checkstyle:checkstyle'

}

post {

success {

echo 'Generated Analysis Result'

}

}

}

stage('build && SonarQube analysis') {

environment {

scannerHome = tool 'sonar4.7'

}

steps {

withSonarQubeEnv('sonar') {

sh """

${scannerHome}/bin/sonar-scanner -Dsonar.projectKey=vprofile \

-Dsonar.projectName=vprofile-repo \

-Dsonar.projectVersion=1.0 \

-Dsonar.sources=src/ \

-Dsonar.java.binaries=target/test-classes/com/visualpathit/account/controllerTest/ \

-Dsonar.junit.reportsPath=target/surefire-reports/ \

-Dsonar.jacoco.reportsPath=target/jacoco.exec \

-Dsonar.java.checkstyle.reportPaths=target/checkstyle-result.xml

"""

}

}

}

stage("Quality Gate") {

steps {

timeout(time: 1, unit: 'HOURS') {

// Parameter indicates whether to set pipeline to UNSTABLE if Quality Gate fails

// true = set pipeline to UNSTABLE, false = don't

waitForQualityGate abortPipeline: true

}

}

}

stage('Build App Image') {

steps {

script {

dockerImage = docker.build(appRegistry + ":$BUILD\_NUMBER", "./Docker-files/app/multistage/")

}

}

}

stage('Upload App Image') {

steps {

script {

docker.withRegistry(vprofileRegistry, registryCredential) {

dockerImage.push("$BUILD\_NUMBER")

dockerImage.push('latest')

}

}

}

}

}

}

Check : go to ECR we will see docker image uploaded there. We have published docker image now we can deploy it to kubernetes, docker or ECS

022

ssh JenkinsServer >

sudo apt-get update -y

sudo apt-get install ca-certificates curl gnupg -y

sudo install -m 0755 -d /etc/apt/keyrings

curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /etc/apt/keyrings/docker.gpg

sudo chmod a+r /etc/apt/keyrings/docker.gpg

echo \

"deb [arch="$(dpkg --print-architecture)" signed-by=/etc/apt/keyrings/docker.gpg] https://download.docker.com/linux/ubuntu \

"$(. /etc/os-release && echo "$VERSION\_CODENAME")" stable" | \

sudo tee /etc/apt/sources.list.d/docker.list > /dev/null

sudo apt-get update -y

sudo apt-get install docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-plugin -y

id jenkins

usermod -a -G docker jenkins

id jenkins : we see jenkins user in docker group

apt install awscli -y

reboot

go to aws > IAM > Users > Add users > jenkins > check Access key - Programmatic access > Attach existing policies > AmazonEC2ContainerRegistryFullAccess & AmazonECS\_FullAccess > Create user > Download .csv

ECR > Create repository > Repository name : vprofileappimg > Create repository

Broswer jenkins > manage jenkins > manage plugins > Docker pipeline, Amazon ECR, Amazon Web Services SDK :: All, Cloudbees

Docker Build and Publish > Install without restart

Broswer jenkins > manage jenkins > manage credentials > Jenkins > Global credentials > Add credentials > AWS credentials >

ID : awscreds > Paste Access & Secret keys > OK

In pipeline script >

registryCredential = 'ecr:us-east-2:awscreds'

appRegistry = "paste url of ECR created" this is image name & tag, but we are mentioning tag in stage ('Upload App Image')

vprofileRegistry = "paste url of ECR remove /vprofileappimg & add https:// at the start" this is registry URL

IMP: Always use global enviornment variables

Jenkins > docker-ci-pipeline > pipeline > OK > paste the script with changes as above > Build now

ECR > we will see image in it

023 Deploying our Docker images OR Extend continuous integration pipeline to continuous delivery.

Developers commits the code to GitHub.Jenkins is going to fetch the code.Run the test.Do code analysis.Upload the result to sonarqube.sonarqube Check for the quality gate.If everything is good, it is going to publish our Docker image

to Amazon ECR.

we need to host this Docker image or the Docker container to a Docker solution like ECS.ECS is a Docker container hosting platform.

IMP : We package our image into Docker images and then we host our Dockerized application on Amazon ECS.

Container hosting platforms :

1.Docker Engine - for testing & local developement enviornment

2.Kubernetes - for production EKS, AKS, GKE, OpenShift

3.Amazon ECS

We can host Dockerized application in 2 ways :

1st way : Using docker engine - docker run imageName, Used for testing and local development environment

2nd way : Using Kubernetes OR Amazon EKS, Azure Kubernetes service AKS, Google Kubernetes engine GKE, OpenShift from RedHat

024

In short :

In enviornment-create ECS cluster-create ECS service-install plugin pipeline: aws steps-Step deploy to ECS withAWS is plugin we installed-In withAWS we mention credentials:awscreds & region us-east-1 & we are going to execute AWS shell command given

Add 2 enviornment variables :

cluster : ECS cluster name

service : service in a cluster is a task which will run container. It fetch image from ECR & run the container.We can also add ELB in a service

Create ECS cluster & ECS service, install plugin "pipeline: aws steps"

We added stage Deploy to ECS. We need "pipeline: aws steps" plugin to use withAWS option in pipeline code. With awscreds

& region we execute the shell command

025

ECS > New ECS experience > Create cluster > Name : vprofile > keep default VPC & subnets > Infrastructure :

1.Dont select Amazon EC2 instances : launch ASG and run container on EC2 instances. Managed by ECS

2.Select this : AWS Fargate: serverless solution for containers.No need to worry wherecontainer runs all managed by AWS.AWS run your container behind the scene > Monitoring : Use container insights (for logs of container) > Create(If any error try with different name)

In Task definitions we mention the image that needs to be fetched & run.Our image is on ECR.

ECS > Task definitions > Create new task > vprofileapptask > Container Name : vproapp > Image URI : paste image URI from ECR > Port 8080 (our tomcat server run on this port) > Next > AWS Fargate > Memory : 2GB > Next > Create

This will create task definition doesnt create a task.We are gonna use this task definitions to create service in the cluster.

ECS > Clusters > vprofile > Service > Deploy > Service > Family (select task definition name) : vprofileapptask > Service name : vprofileappsvc > Desired tasks (how many tasks): 1 > Deplyment option : Rolling update (one container updated at a time) > Create new security group > vproappecselb-sg > HTTP Anywhere > Load balancer > Type : Application load balancer > Create a new Load Balancer > vproappecselb > 80 (elb listen on port 80 & route request to target group & target group has our tomcat container with port 8080) > target group : vprocstg - it contains your containers > HTTP > /login > Deploy

We see service & task is created (ECS > clusters > vprofile > service & task)

GO to load balancer > listener > front end is 80 & backend is target group > click on target group name > vprocstg > Health check > Edit > Advanced health check > Override : 8080 our app vprofileapp runs on tomcat container at

port 8080 > Healthy threshold : 2 > Save changes

EC2 > Security groups > vproappecselb-sg > Inbound rule > edit > Custom TCP 8080 Anywhere IPV4, Custom TCP 8080 Anywhere IPV6

EC2 > Target group > Targets > You will see Healthy status of task / container. This is due to we changed Health check option in target group so it keep deleting & re-creating these tasks/containers

ECS > Clusters > vprofile > Services > vprofileappsvc > Networking > DNS Names > Open address - App is deploying & accessible from load balancer

OR

ECS > Clusters > vprofile > Services > Tasks > Click on running task > copy paste public IP with 8080

026

in pipeline script : enviornment -

cluster = 'vprofile'

service = "vprofileappsvc"

Jenkins > manage jenkins > manage plugin > availabile > Pipeline: AWS Steps > Install without restart

Jenkins > New item > cicd-pipeline-ecs > pipeline > OK > Paste the below script > Build now

aws ecs update-service command will create new container, that container will fetch latest image & run it. The old container

will be deleted

ECS > Clusters > vprofile > Services > vprofilesvc > Deplyment and events > Deplyments > Status primary is newly created

container & active is old container which is going to be deleted after. We just have one single task for high availability

you can have multiple tasks

ECS > Clusters > vprofile > Tasks > we see different container id which is new container > click on new container id >

Logs .After some time previous task gets deleted/stopped & you will have only one task

MUST CHANGE cluster, service names & at the last stage change your region also according to your aws configure.Install plugin pipeline: AWS steps (search with pipeline: only in Available plungins)

Make sure in ECS > Service is Active & Task is Running. Copy container id which is in Tasks > Click on Running task > Copy container id save somewhere. This container id is going to delete after we run the pipeline coz in last stage of pipeline command "aws ecs update-service..." this command will create a new container & old container is slowly deleted.

931c85781bf14f27883077cc85719c7c this container is going to be deleted

make sure you added right enviornment variables correctly

During running the pipeline check ECS > Service > vprofileappsvc > Deployments & events > we see new task has been started

ECS > Tasks > click on new container name > Logs we can check logs of container here

pipeline {

agent any

tools {

maven "MAVEN3"

jdk "OracleJDK8"

}

environment {

registryCredential = 'ecr:us-east-2:awscreds'

appRegistry = "951401132355.dkr.ecr.us-east-2.amazonaws.com/vprofileappimg"

vprofileRegistry = "https://951401132355.dkr.ecr.us-east-2.amazonaws.com"

cluster = "vprofile"

service = "vprofileappsvc"

}

stages {

stage('Fetch code') {

steps {

git branch: 'docker', url: 'https://github.com/devopshydclub/vprofile-project.git'

}

}

stage('Test') {

steps {

sh 'mvn test'

}

}

stage('CODE ANALYSIS WITH CHECKSTYLE') {

steps {

sh 'mvn checkstyle:checkstyle'

}

post {

success {

echo 'Generated Analysis Result'

}

}

}

stage('build && SonarQube analysis') {

environment {

scannerHome = tool 'sonar4.7'

}

steps {

withSonarQubeEnv('sonar') {

sh """

${scannerHome}/bin/sonar-scanner -Dsonar.projectKey=vprofile \

-Dsonar.projectName=vprofile-repo \

-Dsonar.projectVersion=1.0 \

-Dsonar.sources=src/ \

-Dsonar.java.binaries=target/test-classes/com/visualpathit/account/controllerTest/ \

-Dsonar.junit.reportsPath=target/surefire-reports/ \

-Dsonar.jacoco.reportsPath=target/jacoco.exec \

-Dsonar.java.checkstyle.reportPaths=target/checkstyle-result.xml

"""

}

}

}

stage("Quality Gate") {

steps {

timeout(time: 1, unit: 'HOURS') {

// Parameter indicates whether to set pipeline to UNSTABLE if Quality Gate fails

// true = set pipeline to UNSTABLE, false = don't

waitForQualityGate abortPipeline: true

}

}

}

stage('Build App Image') {

steps {

script {

dockerImage = docker.build(appRegistry + ":$BUILD\_NUMBER", "./Docker-files/app/multistage/")

}

}

}

stage('Upload App Image') {

steps {

script {

docker.withRegistry(vprofileRegistry, registryCredential) {

dockerImage.push("$BUILD\_NUMBER")

dockerImage.push('latest')

}

}

}

}

stage('Deploy to ecs') {

steps {

withAWS(credentials: 'awscreds', region: 'us-east-2') {

sh 'aws ecs update-service --cluster ${cluster} --service ${service} --force-new-deployment'

}

}

}

}

}

027 Clean up

ECS > clusters > vprofile > services > vprofileappsvc > Edit > Desired tasks > 0 > Update

ECS > clusters > vprofile > services > vprofileappsvc > Delete

ECS > clusters > vprofile > Delete cluster

if it gave error ECS > clusters > vprofile > Tasks > Stop the running task & then delete cluster

028 Build triggers - Jobs gets executes automatically by using triggers. No need to click on build now to trigger a jenkins job

Popular Triggers :

1.Git webhook - Github will send àdjacent payload whenever there is a commit in the repository. Github repository will trigger your jenkins job

2.Poll SCM - Jenkins will check for commit in git repository at time interval that you specify like 5 minutes. Every 5 minutes jenkins check for new commit in git, if any commit found the job will get triggered.

3.Scheduled jobs - Like cron job, you mention date & time in cron job format & jenkins will run your job at that particular date & time or intervals

4.Remote triggers - We can trigger jenkins jobs from anywhere from a script or from an Ansible playbook. You get API call which we use to trigger jenkins job

5.Build after other projects are build - When previous job completed , your next job gets triggered.

STEPS :

1.Create git repository on github

2.ssh authentication to github repository

3.Create Jenkinsfile & place it in repository & commit it

4.Create jenkins job to access Jenkinsfile from git repository

5.Testing triggers

Github account > Create new repository jenkinstriggers > Private > Create repository

Create ssh keys - ssh-keygen > cat ~/.ssh/id\_rsa.pub > copy & paste in GitHub > settings > SSH & GPG keys > New

SSH keys > Title : myLaptop & Paste key > Add SSH key

Go to jenkinstriggers repository & copy ssh path

mkdir -p gitRepos && cd gitRepos

git clone paste\_SSH\_URL

cd jenkinstriggers

vim Jenkinsfile

pipeline {

agent any

stages {

stage('Build') {

steps{

sh 'echo "Build completed."'

}

}

}

}

gir add .

git commit -m "first commit"

git push origin master

If you get Host key verification failed in jenkins try below :

Jenkins > manage jenkins > Configure global security > Git host key verification configuration > Accept first connection

> Apply > Save

Jenkins > New item > Build > Pipeline > OK > Pipeline script from SCM > SCM : Git > Repository URL : Paste SSH URL of

jenkinstriggers repository > Credentials > Add > Jenkins > Kind : SSH Username and private key > ID : gitsshkey > Username

: github account username > Private key : Enter directly > Add > Paste private key from ~/.ssh/id\_rsa > Add > Credentials :

select credential just created (with (gitsshkey)) > Script Path : Jenkinsfile > Save > Apply > Build now

029

1.Git webhook : based on different events like commits in github jenkins jobs will get triggered

Github > jenkinstriggers > settings (repository settings) > Webhooks > Add webhook > Paste URL of jenkins

http://publicIPJenkins:8080/github-webhook/ > Content type : application/json > Just the push event > Add webhook >

refresh the page you will see green tick > Click on it > Recent deliveries

Jenkins > Build > Configure > Build Triggers > Check Github hook trigger for GITScm polling > save

touch testfile.txt

git add .

git commit -m "testtrigger1"

git push origin master

You will second build automatically generated in jenkins after we commit & push

2.Poll SCM : Jenkins will check for commits for every minute & whenever there is commit github will send JSON payload

and trigger a job

Jenkins > Build > Configure > Build Trigger > Check Poll SCM > MINUTE HOUR DOM MONTH DOW \* \* \* \* \* \* > save

Jenkins > Build > Git polling log > You will find No changes at the end

touch testfile2.txt

git add .

git commit -m "testtrigger2"

git push origin master

After a minute check Jenkins > Build > Git polling log > You will find Changes found at the end & new build is created

3.Scheduled jobs : It does not check SCM or Github, it will run the job at that particular time

Jenkins > Build > Configure > Build Trigger > Build periodically only > MINUTE HOUR DOM MONTH DOW 30 20 \* \* 1-5 > save

This runs a job from mon to fri every night at 8:30 pm (0 is sunday)

4.Remote Triggers : From anywhere you can trigger jenkins job, anywhere means from script, from other jenkins server,

from server, from laptop as long you have network access to the jenkins server

Check 029 Build-Triggers-Remotely.pdf

1.JOB URL :

Jenkins > Build > Configure > Build Trigger > Check Trigger builds remotely > Authentication token : mybuildtoken >

Jenkins\_URL\_with\_8080/job/Build/build?token=TOKEN\_NAME Jenkins\_URL\_with\_8080/job/Build/build?token=mybuildtoken > save

2.TOKEN :

Jenkins > Admin > Configure > API Token > Add new Token > Generate > copy User:Token admin:paste\_token >

3.CRUMB :

For windows download wget 64bit zip file > open zip file > wget.exe > extract to c:/program files/Git/mingw64/bin

wget -q --auth-no-challenge --user username --password password --output-document - 'http://JENNKINS\_IP:8080/crumbIssuer/api/xml?xpath=concat(//crumbRequestField,":",//crumb)'

wget -q --auth-no-challenge --user admin --password admin123 --output-document - 'http://JENNKINS\_IP:8080/crumbIssuer/api/xml?xpath=concat(//crumbRequestField,":",//crumb)'

Enter jenkins user password & URL in above command. After that copy output of above command & paste it in below command

also paste JOB URL & TOKEN in below command

curl -I -X POST http://username:APItoken @Jenkins\_IP:8080/job/JOB\_NAME/build?token=TOKENNAME -H "Jenkins-Crumb:CRUMB"

e.g : Curl -I -X POST http://admin:110305ffb46e298491ae082236301bde8e@52.15.216.180:8080/job/vprofile-Code-Analysis/build?token=testtoken -H "Jenkins-Crumb:8cb80f4f56d6d35c2121a1cf35b7b501"

Run above command in git bash, you will see new build in jenkins

5.Build after other projects are built :

Jenkins > New item > testjob > freestyle > OK > Build > Add build step > Execute shell > echo "test test test" > save

Jenkins > testjob > configure > Build Trigger > Build after other project are built > Projects to watch : Build > save

After Build job completes, then testjob will be run

Jenkins > Build > Build now > Console log - you will see at the end "Triggering a new build of testjob #1" > click

on testjob > You will see one build there

030 Master slave jenkins

So far, whatever we are running on Jenkins,we are running it from Jenkins master.

But there will be situations, there will be scenario or use cases where you need to run jobs from some other machine.

Those other machines will be the slave machines for Jenkins,

Used cases of Master-slave jenkins :

1.load distribution or Distributed builds :

The most common use cass load distribution or distributed builds.

You're using Jenkins at an organization level, and there are many, many jobs that are getting triggered automatically.

People are executing jobs.So it's not possible for Jenkins to run all those jobs.So if you add a node as a slave to

Jenkins, then Jenkins can decide if the job has to be running on master or on the slave.

So Jenkins will pick up a slave from its arsenal and execute the job on the slave.for e.g cloning the source code,you

are running MAVEN commands, you're running software tests.These things will be running on the slave.

2.Cross-platform builds :

You're running Jenkins on a Linux machine and you need to build a Windows based package.So you need some Windows tools

like MS build, and you cannot execute that on Linux machine.So you will add a Windows machine as a slave to Jenkins

and you can then specifically say that run my Windows job only on this Windows machine or same for Mac OS.

3.Software testing :

If you're doing continuous delivery, you should really include the software testing.So software testers or the QA team

are going to write software test cases.They will be mostly executing it from a Windows machine, so it will open like

a browser graphically and execute the test cases.

So mostly software testers will have some machine.You can add those machine as a slave and you can run the software

test cases from Jenkins.So then you can include that job also in your pipeline.

You want to run your scripts, shell script, Python script, your Ansible playbook.So you have a separate machine that

runs your script.You can add it as a slave or for that matter, anything that you want to run, any command, any script

that will be running from some machine.You can add that some machine as a slave to the Jenkins master.

YOU CAN EXECUTE ANBYTHING (Any script, any command, any test cases) FROM JENKINS

You can use Jenkins as a centralized platform to execute anything in your infrastructure.

Prerequisites for Node / slave setup :

1.You can have any operating system

2.There should be network access between Jenkins master to the slave.And what I mean by that is not just the network

connectivity, but there could be firewalls like security group or any SQL or any third party firewall in between

master and the slave.So you need to check those firewall rules.

3.You need to have Java, JRE or JDK based on your requirement.

4.A user in the node that Jenkins will use to connect.

5.A directory with the user's ownership.So Jenkins will use this user to connect to your slave machine, to your node,

and will create files or access files in a directory.So the user should have ownership on that directory.

6.The tools that you require to run a job.You can manage this from global tool configuration also, or you can log into

the slave and install the right tools like you need maven or git or ANT

Practical :

Launch EC2 > Ubuntu 18.04 > tags - Name : Slave-jenkins > Create SG : Slave-SG SSH MyIP > Use same key used for

JenkinsServer

ssh this instance :

apt update && apt install openjdk-11-jdk -y

adduser devops

mkdir /opt/jenkins-slave

chown devops.devops /opt/jenkins-slave -R

vim /etc/ssh/sshd\_config > PasswordAuthentication yes

systemctl restart ssh

Jenkins > Manage jenkins > Manage nodes and clouds > New node > silver-node > Click Permanent Agent > OK >

# of executors - how many jobs you want to run parallely : 5

Remote root directory : /opt/jenkins-slave

Labels : SILVER

Usage : 1.Use this node as much as possible - use this option for load distribution

2.Only build jobs with label expressions matching this node - You want to run windows build only on windows

machine.In jenkins job you have to make entry of this node

Launch methods : Launch agent with SSH >

Host : Paste the private IP of slave machine i.e Slave-jenkins > In Slave-SG

> Edit inbound rule > Custom TCP 22 Custom JenkinSG Allows jenkins to do ssh > save rule

Credentials > Add > Jenkins > Kind : Username with password > Username : devops Password : password of devops user >

ID : silver-login > Add > Credentials > Add > Jenkins >

Kind : SSH Username with private key >

ID : silver-login-key Username : ubuntu > Private

key : cat jenkins-key.pem copy & paste key here > Add >

Credentials : devops/\*\*\*\*\*\*(silver-login)

Host key verification strategy : Non verifying verification strategy (when ssh for machine first time it stores

machines identity into known host file & it will ask whether you want to login yes or no) > save

Click on silver-node > see log for more details > `You will see "Agent successfully connected and online"

If error of connection timeout > check SG of slave is it allowing SSH 22 from jenkins

If error of permission denied > check ownership of directory

If error of authentication failure > check username & password or password based login enabled or not

Jenkins > New item > test-slave > Freestyle > OK > Build > Add build step > Execute shell > pwd whoami ls -ltr > save

> Build now > Console output > You will see 3rd line "Building remotely on silver-node" & SUCCESS

ssh Slave-jenkins > cd /opt/jenkins-slave && ls > you see remoting folder and remoting.jar thats the agent which makes

connection to the master & in the workspace you have test-slave directory which is the jobs workspace directory

1.WE DID IT ON A CHANCE BY USING THIS OPTION 1.Use this node as much as possible THIS MEANS JOB WILL RUN ON EITHER MASTER OR SLAVE WE CANT CONTROL WHERE JOB WILL RUN. ONLY FOR SILVER

2.BUT IF WE WANT OUR JOB RUN ONLY ON SPECIFIC NODE / SLAVE STRICTLY THEN : 100% chance of using Slave-node to run

test-slave job

3.IF WE WANT TO DISABLE AUTOMATIC SELECTION OF THIS NODE :

Jenkins > Manage jenkins > Manage nodes and clouds > silver-node > configure > Usage : 2.Only build jobs with label

expressions matching this node > save

NOW JENKINS WILL NOT USE THIS NODE(silver-node) AUTOMATICALLY WHEN YOU RUN ANY JOB

A) 1.Use this node as much as possible : automatically selection of silver-node(slave) when running the job

B) 2.Only build jobs with label expressions matching this node : silver-node(slave) will not be selected automatically,

silver-node will be selected only if we specify it in the job.

C) This will run Build-Test job on silver-node only :

Jenkins > Build-Test > Configure > General > Check Restrict where this project can be run > Label Expression : Give

Node name SILVER > You can select Tools used for silver-node Git MAVEN. If you select Default in maven job will fail

> save > Build now > console output : Failed

Jenkins > Build-Test > Configure > General > Check Restrict where this project can be run > Label Expression : Give

Node name SILVER > You can select Tools used for silver-node Git MAVEN. If you select version from drop down job will

run > save > Build now > console output : SUCCESS

031 Authentication is login & authorization privilege :

**The developer will write the code, make the changes to it, and test it locally & push the code to a centralised repository like GitHub.** So developers will have a git tool that will integrate with the GitHub repository, and the code will be committed to the GitHub repository. **As soon as there is a code change, Jenkins will detect a change and fetch the code by using the git tool. So Jenkins will have a git tool and a git plugin, which will help accomplish this task of fetching the code whenever there is a change.** After that the code will be built. We will be using MAVEN to build the code because we have Java code and our code can be built with the MAVEN tool, but it could be any other source code and other build tools as well. Once the build is complete, it will generate artifacts. Next, we will conduct a unit test again using Maven. Maven will have some unit testing frameworks that developers will use. Unit testing will be part of your source code. We just need to execute some steps that will run this test and generate reports, mostly in XML format. Once you have the reports ready, we will conduct another kind of test called code analysis. Now the unit test checks whether the unit of the code works or not. Code analysis checks if the code has any vulnerabilities. Are you following the best practices? Do you have any bugs in the code? And there are many other parameters on which code analysis will judge your code. We will be using a SonarQube scanner to scan the code. Also, we will be using checkstyle. So there are many code analysis tools available on the market. We are using a SonarQube scanner and CheckStyle to scan the code, and this will generate reports in XML format. These reports will be uploaded to the Sonar Qube server. In Sonarqube, you can have proper graphs and charts, and you can see what bugs, vulnerabilities, and many other things are in your code. We can also set a quality gate, and we can say that if my code does not follow these practices, then it will fail the build. And if it fails, the pipeline will stop. If it passes, We then have a verified copy of the artifact.

**Push > jenkins detect changes using git tool > Maven > unit tests > code analysis sonarqube & checkstyle > Nexus sonatype repository**

1.Developers write, modify, and test their code on their local machines.As soon as developer push code to git repository jenkins detects & fetch the updated code using git tool.

2.Jenkins build the code using build tool maven. Once the code is build it will create artifact.Jenkins runs unit tests using the testing frameworks provided by Maven. Maven is a tool for building Java code, but you can use different tools like npm for nodejs, Ant, Gradle etc.

3.We run unit tests check if different parts of the code work properly.Developers write these unit tests.Jenkins collects test results and generates reports in XML format.These reports are uploaded to sonarqube so we can have proper graphs and charts

4.After that code analysis is done by SonarQube and CheckStyle.We can set quality gates to check bugs & vulnaribilities in the source code.If everything is good we push the artifacts to Nexus Sonatype repository which keeps record of different versions of artifact.