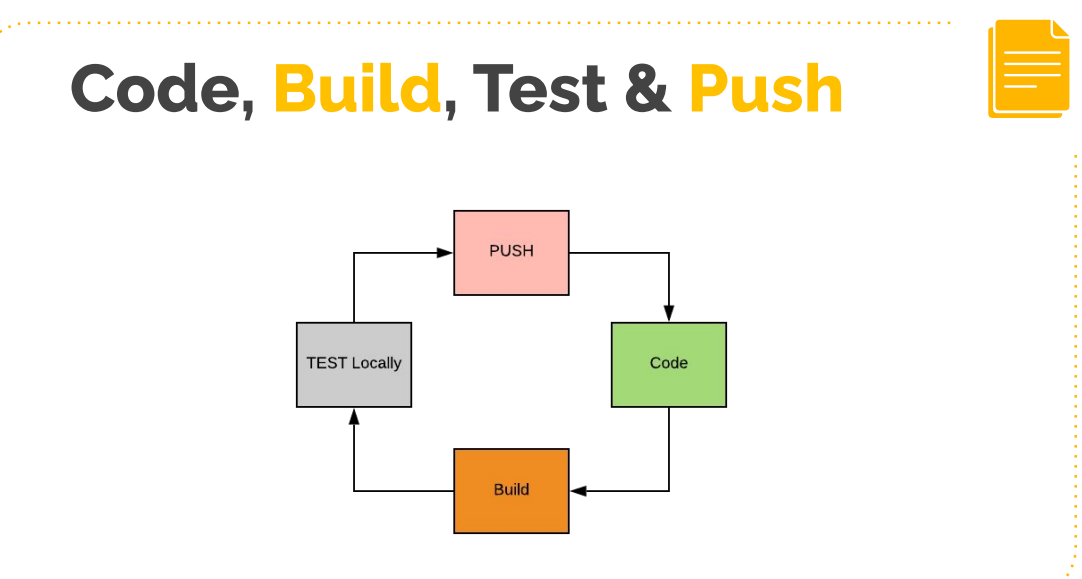
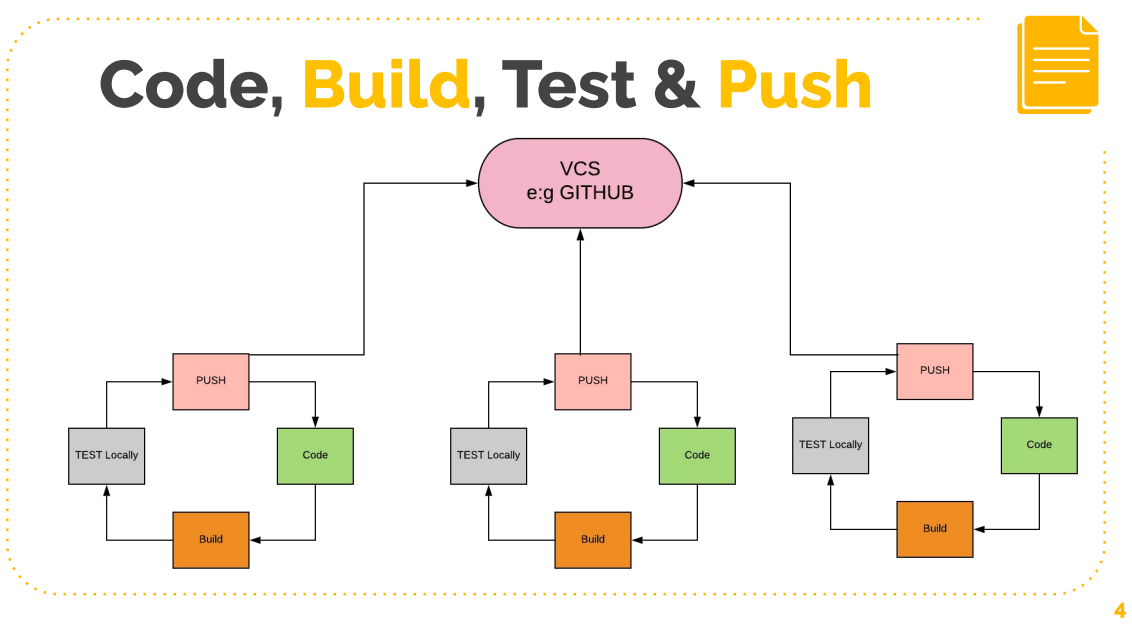
# Jenkins

What is Continuous Integration?

When developer, working on a product development, will code, build the code, test it and push it in the centralized version control system.



Continuous process : right the code, build & test locally, if everything is good then push into the centralized control system like that all the developer will be doing same thing in the team



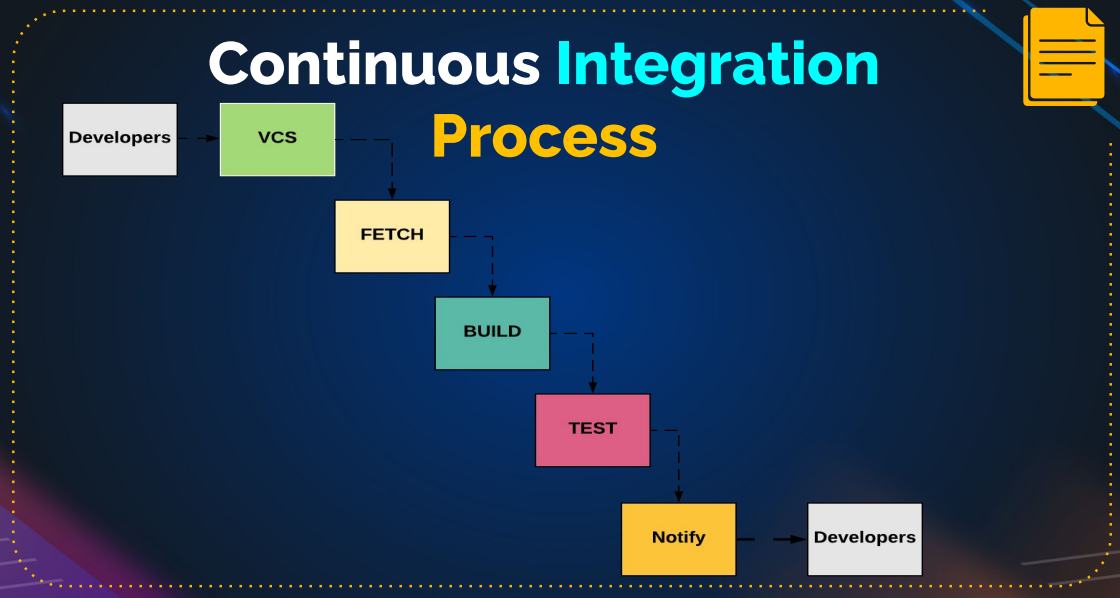
But they is problem in it, well, this is happening many, many times in a day by all the different developers and the code gets marge in the centralized repository system on a regular basis and days past week pass, sometimes even month pass the code keep getting merge and then after some time , if the code is together build and tested it generates a lot of bugs, issues and conflicts.

**That is happening because the code is getting merge, but really not getting integrated**

Developer keeps merging the code to the version control system multiple time in a day and several times in a week/month and there could be so many bugs. well, this code is getting merged and after a long time, when it is tested together, it generated those errors, those bugs, those conflicts and it is very natural , but the problem starts after that everything need to be fixed so developers needs to do a lot of rework, invest a lot of time in fixing all this maybe deadline is coming close/past the deadline

So integration is a painful, but we can resolve it, they is solution for this whenever the code changes in centralized repository or for every single commit that done by the any developer. We build the code, build code for every commit. Build and test the code and this has to done by every commit then this process needs to be automated whenever the code change, the code be fetched, build and tested. If anything is wrong the developer gets notify’s and fix the bug/issues and build the code again. **This process is called Continuous Integration**

the code get’s build & test in automated way is is called continuous integration .

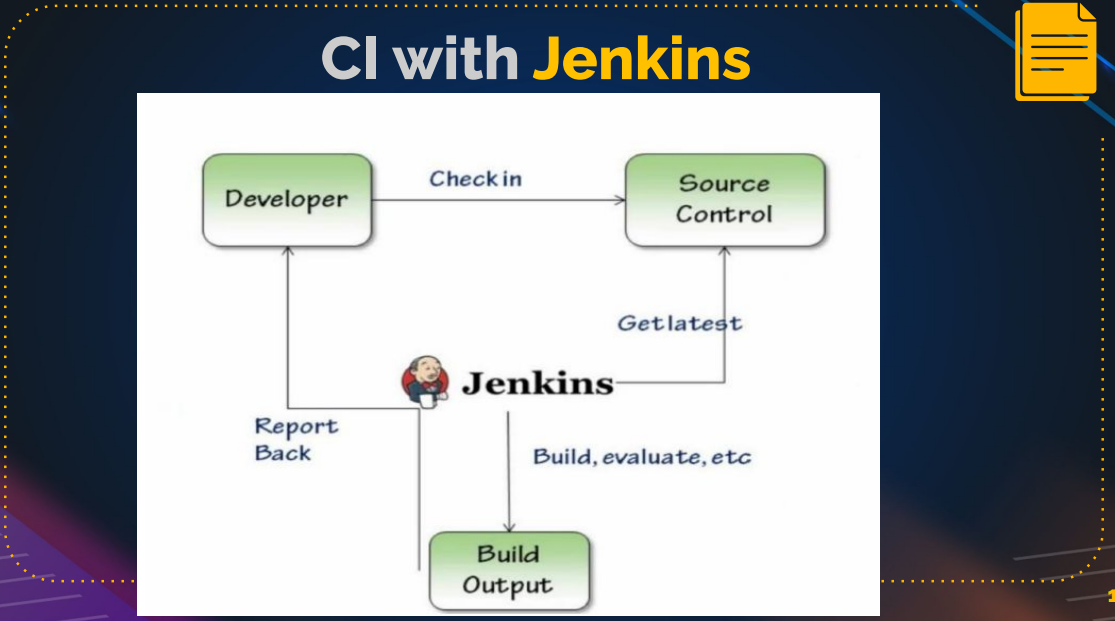


In the process (vcs🡪fectch🡪build🡪test🡪notify )Dev got the error any step the fix the issue and the continuous the process.

So not only just keep merging the code but also for every commit which will test and notify.

If this is a continuous process, then we need to automate this process , if we automate this process this called a continuous integration process and Jenkins & nexus is one of the tools for CI /CD process.

Jenkins:



This as two majar features

1)opensource

2) extensible(Plugins: ● VCS Plugin ● Build Plugin ● Cloud Plugin ● Testing Plugin ● Etc etc etc)

## Jenkins Installation

Prerequisite 1. Java- JRE, JDK 2. Any OS

<https://www.jenkins.io/> - refer the Jenkins links for installation (refer documents)

hands-on: create a ec2 ubuntu instance 🡪 followed the installation steps

ex: steps for ubuntu mahine

#!/bin/bash

sudo apt update

sudo apt install openjdk-17-jdk -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

## Jobs in Jenkins

They are different of jobs mainly comparing the freestyle vs pipeline as a code

**Jobs :** jobs basically means workload, which Jenkins runs for you

Freestyle jobs : (not recommended now-a-days)

* Graphical jobs
* Learning, understanding & exploring Jenkins.
* Not recommended in real time now

Pipeline As a Code

* Pipleline created in groovy
* Recommended now

# Installing the tools in Jenkins

Now we can create a job in Jenkins, which is going to fetch our source code and build that source by using mvn install command

We required three thing

* Git 🡪 to fetch the source code
* Maven
* Installing the dependency of maven 🡪 jdk

Whenever you are using tools in Jenkins, we need to remember two things

* One is the tool itself(you can install the tools by logging in jenkins instance, and install the tool by using APT commands)
* Plugins

In order to install the tools in **Jenkins 🡪 manage Jenkins 🡪tools🡪add the path for jdk and maven**

Jdk (/usr/lib/jvm/java-17.0-openjdk-amd64)

## Create a new job from freestyle

Add the description

Execute the shell completed like “whoami id pws”

2) echo "i am trying to learn devops from imron" >> jenkins.txt

echo "##################################"

echo "this is my jenkins job $USER"

ls

echo "##################################"

cat jenkins.txt

create a new job for build vprofile using maven

* Job: build 🡪 description build the artifacts from vprofiles
* Add the git url 🡪 https://github.com/hkhcoder/vprofile-project.git -->\*/main
* Jdk 🡪 OracleJDK17(normally jdk11 I had used jdk 17)
* Select :MAVEN3 🡪 clean install (config the version in tools)

If you going to default then you need to **install the maven in server region**

And run the job then it create a artifact

If I am not able to run the job due to low space in ec2 instance them

* Stop the ec2 server 🡪 instance setting 🡪change the instance type(to increase the cpu and ram)

In post build:

We can give the \*\*/\*.war to archive the artifacts

Job : Build\_test

We can copy the configuration one job to other(copy build 🡪 Build\_test)

# **Plugins, Versioning & more**

### **Versioning**

**🡪normally when the build completed then output is generated in the workspace, in the target directory, we are seeing the artifact.**

**Workspace🡪target 🡪 artifact**

🡪every time we run the job; this artifact will get replaced with newer version of artifact. But here we cannot retrieve older version of artifact. so, to preserver all the artifacts, you can do some versioning (we can use some shell commands)

Mkdir -p versions

Cp target/vprofile-v2.war versions/vprofile.V$BUILD\_ID.war

We parameterized so that we can give the version at run time(string parameterized)

Mkdir -p versions

Cp target/vprofile-v2.war versions/vprofile.V$versions.war

Now you will get build with parameters 🡪(it is case sensitive)

But tried avoiding parameter from use

**See the plugins in Jenkins**

**Ex:-** install the plugin for timestamp **🡪 zentimestamp**

Now we can able to see the timestamp in configuration 🡪 yyyymmddhhmm

Mkdir -p versions

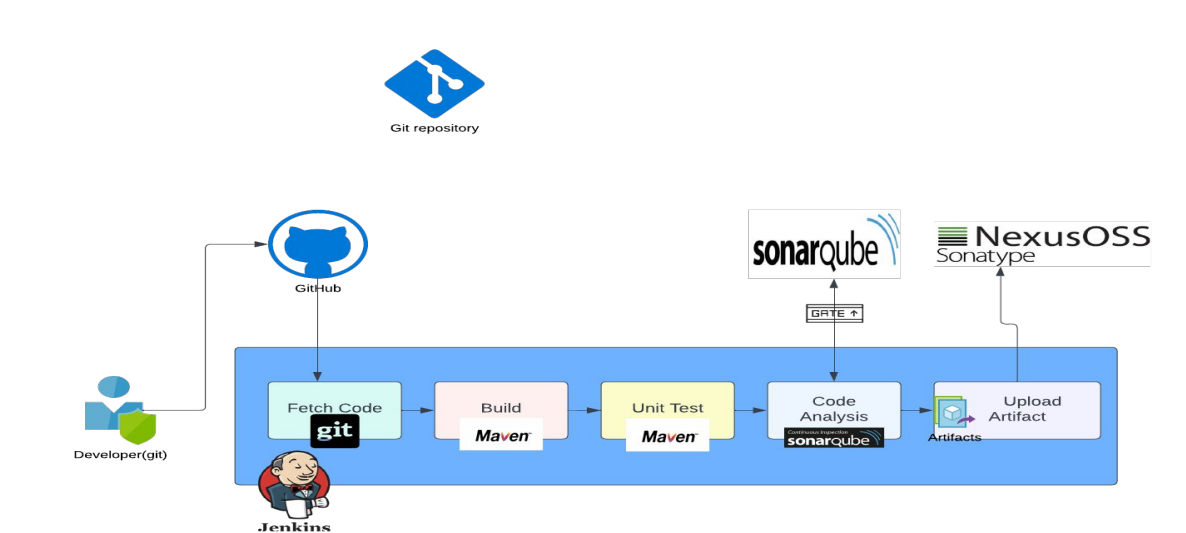
Cp target/vprofile-v2.war versions/vprofile.V$BUILD\_ID -$BUILD\_TIMESTAMP.war

## **Jenkins Set Environment Variables**

When a Jenkins job executes, it sets some environment variables that you may use in your shell script, batch command, Ant script or Maven POM [#1](https://wiki.jenkins.io/display/JENKINS/Building+a+software+project#Buildingasoftwareproject-1). The following table contains a list of all of these environment variables.

|  |  |
| --- | --- |
| **Environment Variable** | **Description** |
| BUILD\_NUMBER | The current build number, such as "153" |
| BUILD\_ID | The current build id, such as "2005-08-22\_23-59-59" (YYYY-MM-DD\_hh-mm-ss, [defunct](https://issues.jenkins-ci.org/browse/JENKINS-26520) since version 1.597) |
| BUILD\_URL | The URL where the results of this build can be found (e.g. http://buildserver/jenkins/job/MyJobName/666/) |
| NODE\_NAME | The name of the node the current build is running on. Equals 'master' for master node. |
| JOB\_NAME | Name of the project of this build. This is the name you gave your job when you first set it up. It's the third column of the Jenkins Dashboard main page. |
| BUILD\_TAG | String of jenkins-${JOB\_NAME}-${BUILD\_NUMBER}. Convenient to put into a resource file, a jar file, etc for easier identification. |
| JENKINS\_URL | Set to the URL of the Jenkins master that's running the build. This value is used by [Jenkins CLI](https://wiki.jenkins.io/display/JENKINS/Jenkins-CLI.html) for example |
| EXECUTOR\_NUMBER | The unique number that identifies the current executor (among executors of the same machine) that's carrying out this build. This is the number you see in the "build executor status", except that the number starts from 0, not 1. |
| JAVA\_HOME | If your job is configured to use a specific JDK, this variable is set to the JAVA\_HOME of the specified JDK. When this variable is set, PATH is also updated to have $JAVA\_HOME/bin. |
| WORKSPACE | The absolute path of the workspace. |
| SVN\_REVISION | For Subversion-based projects, this variable contains the revision number of the module. If you have more than one module specified, this won't be set. |
| CVS\_BRANCH | For CVS-based projects, this variable contains the branch of the module. If CVS is configured to check out the trunk, this environment variable will not be set. |
| GIT\_COMMIT | For Git-based projects, this variable contains the Git hash of the commit checked out for the build (like ce9a3c1404e8c91be604088670e93434c4253f03) (all the GIT\_\* variables require git plugin) |
| GIT\_URL | For Git-based projects, this variable contains the Git url (like git@github.com:user/repo.git or [https://github.com/user/repo.git]) |
| GIT\_BRANCH | For Git-based projects, this variable contains the Git branch that was checked out for the build (normally origin/master) |

Flow of continuous integration pipeline:



Developer(dev’s write the code/make the changes in code and test it locally if they are good with changes, they will push it to a centralized repository like GitHub.)

🡪 so developer will have a tool which integrate with github repository and the code will be committed to github repository as soon as there it a code change Jenkins will detect and fetch the code by using git tool (**Jenkins as git tool and git plugin which will help accomplish this task to fetch the code**)

🡪code pipeline **MAVEN** Using to build the code because we have a java code(our code can be build with maven(vprofile) , we can also use **ANT🡪 ONE BUILD COMPLETE IT WILL GENERATE ARTIFECT**

**🡪** we will conduct the unit test by using maven(unit testing is a part of source code) , need to execute some step that run the testand generate reports mostly in xml format

* Code analysis

Unit test 🡪 it check the unit of the code works or not.

Code analysis 🡪 it check if the code as any vulnerability, are you following the best practices, bug in the code, many other analysis to judge your code.

We will use **sonarqube** scanner , checkstyle to scan the code, this will generate the reports in xml format

* This xml reports will uploaded to sonarqube server , so that we can see the info in graph, charts to see vulnerable, bugs, best particles etc. if the test failed the pipeline will stoped
* If the test completed the we have verified copy of artifacts 🡪 before distribute artifact to be deployed on server, this artifact will properly versioned and store in NEXUS OSS sonartype repository

## Steps for Continuous Integration pipeline

Steps:

* Jenkins setup, Nexus setup & sonarqube setup 🡪 this done by bash script in ec2
* Security group
* Plugins
* Integrate

Nexus

Sonarqube

Write pipeline script

Set notification

# Jenkins, Nexus and SonarQube setup

Nexus

#!/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

Setup

## Sonar-analysis properties:

sonar.projectKey=vprofile

sonar.projectName=vprofile-repo

sonar.projectVersion=1.0

sonar.sources=src/

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

sonar.junit.reportsPath=target/surefire-reports/

sonar.jacoco.reportsPath=target/jacoco.exec

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

## sonar setup

#!/bin/bash

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

cat <<EOT> /etc/sysctl.conf

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

sonarqube - nofile 65536

sonarqube - nproc 409

EOT

sudo apt-get update -y

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

sudo update-alternatives --config java

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"

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

sudo -i -u postgres psql -c "CREATE DATABASE sonarqube OWNER sonar;"

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

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

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

sonar.jdbc.username=sonar

sonar.jdbc.password=admin123

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

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

# Jenkins Plugins:

Nexus

SonarQube

Git

Pipeline Maven integration plugin

Pipeline utility steps

build Timestamp

# PIPELINE AS A CODE INTRO

Jenkinsfile declares stages in Pipeline

❖ Automate pipeline setup with Jenkinsfile

❖ Jenkinsfile defines Stages in CI/CD Pipeline

❖ Jenkinsfile is a text file with Pipeline DSL Syntax

❖ Similar to groovy

❖ Two Syntax

➢ Scripted

➢ Declarative

## Pipeline Concepts

★ Pipeline

★ Agent

★ Stage

★ Step

pipeline {

agent any stages {

stage('Build') {

steps {

//

}

}

stage('Test') ) {

steps {

//

}

}

stage('Deploy') { steps { // }

}

}

}

pipeline { ..

…

…..

}

pipeline {

agent { }

tools { }

environment { }

stages { }

}

pipeline {

agent { label "master" }

Tools { maven "Maven" }

}

## EXAMPLE

pipeline {

environment {

NEXUS\_VERSION = "nexus3" NEXUS\_PROTOCOL = "http"

NEXUS\_URL = "you-ip-addr-here:8081" NEXUS\_REPOSITORY = "maven-nexus-repo" NEXUS\_CREDENTIAL\_ID = "nexus-user-credentials" ARTVERSION = "${env.BUILD\_ID}"

TIME = “${BUILD\_TIMESTAMP}”

}

}

pipeline {

stages {

stage("Clone code from VCS") { }

stage("Maven Build") { }

stage("Publish to Nexus Repository Manager") { } } }

pipeline { stages { stage("Clone code from VCS") { steps { } post { } } }

pipeline { stage('BuildAndTest'){ steps { sh 'mvn install' } post { success { echo 'Now Archiving...' archiveArtifacts artifacts: '\*\*/target/\*.war' } } } }

pipeline {

agent any

tools{

maven "MAVEN3"

jdk "OracleJDK17"

}

stages{

stage('Fectch code'){

steps{

git branch: 'main', url: 'https://github.com/hkhcoder/vprofile-project.git'

}

}

stage('build'){

steps{

sh 'mvn install -DskipTests'

}

post {

success {

echo 'archiving artifact now'

//archiveArtifacts artifects: '\*\*/\*.war'

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

}

}

}

stage('test'){

steps{

sh 'mvn test'

}

}

}

}

## <https://www.jenkins.io/doc/book/pipeline/jenkinsfile/>

# CODE ANALAYSIS

## Why Code Analysis?

● Best Practices

● Vulnerabilities in code

● Functional Errors before deployment

Variety of test performed on the CODE.

Top 10 owas vulnerabilities

## Tools

* Checkstyle
* Cobertura
* mstest
* owasp
* SonarQube Scanner

Etc ..

stage('Checkstyle Analysis'){

steps {

sh 'mvn checkstyle:checkstyle'

}

}

stage('Sonar Analysis') {

environment {

scannerHome = tool 'sonar4.7'

}

steps {

withSonarQubeEnv('sonar') {

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

-Dsonar.projectName=vprofile \

-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'''

}

}

}

We can search in chatgpt also like “Jenkinsfile to scan the source code with sonar scanner and upload the result to nexus sonarqube server dashboard and with sonar scanner with properties of junt , jacoco & checkstyle report path”

<https://www.jenkins.io/doc/pipeline/steps/sonar/>

# Quality gate

In sonar qube🡪quality gate 🡪we can create our own quality gate(we can add the condition like if the bugs is more then 60th then build should fail)

If the quality gate is running it hold the more time then fail , in that case create the webhooks(sonarqube🡪project setting 🡪webhooks🡪set the url of Jenkins and create a webhook) or sonerqube security group

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

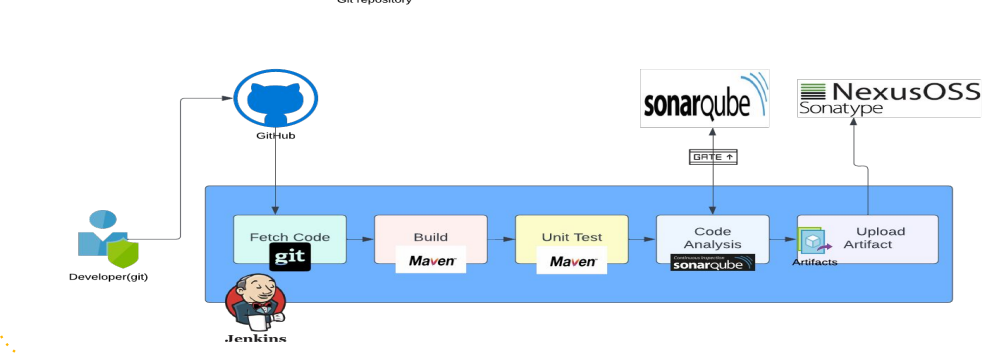
waitForQualityGate abortPipeline: true

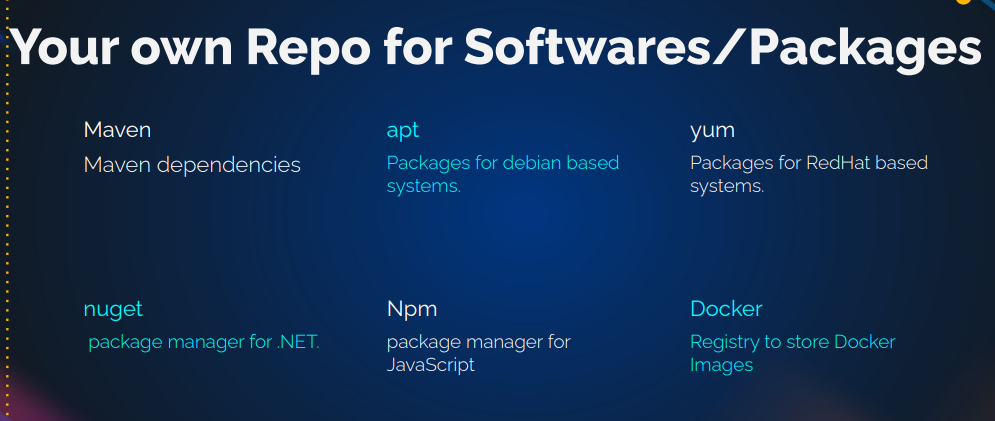
}

}

}

# Software repositories intro (Nexus)





## Nexus Software Repository Manager.

Key Points

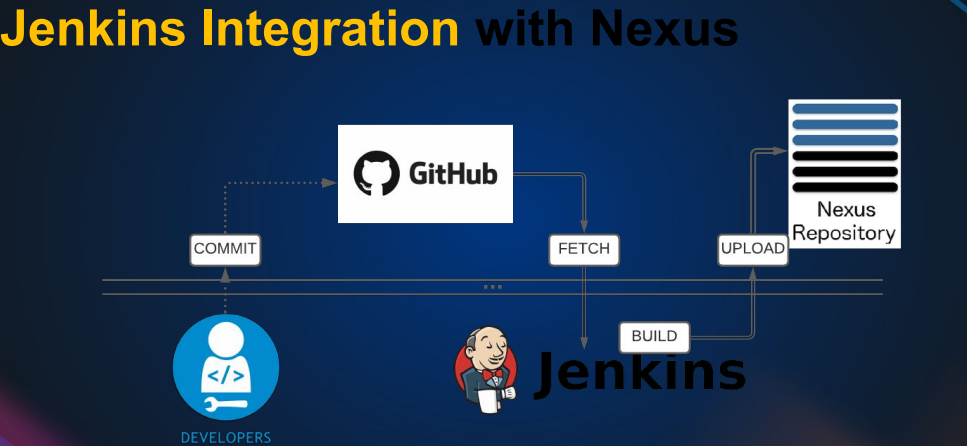
● Runs on java

● Used to store artifacts

● Used as a Package manager for dependencies

● Opensource & Enterprise Versions

● Supports Variety of repo like maven,apt,docker, Ruby gems etc…



One Jenkins build the artifacts and code analysis though sonar scanner then upload in Nexus repo, after that ops team will takes the artifact and deploy in servers

**Login to the nexus repo 🡪 repository 🡪 create repository( host**

Host 🡪store the artifacts

Proxy 🡪 download a dependences

Group 🡪 it is too group the both the repo together(host,proxy)

Create a maven host repo to the reports and artifacts

* Add the nexus credential in Jenkins( dashboard 🡪credentials🡪system🡪global credentials 🡪add the nexus credentials and save it)

stage("UploadArtifact"){

steps{

nexusArtifactUploader(

nexusVersion: 'nexus3',

protocol: 'http',

nexusUrl: '172.31.18.28:8081',

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']

]

)

}

}

<https://plugins.jenkins.io/nexus-artifact-uploader/>

Jenkins 🡪manage Jenkins 🡪configure system 🡪 update SonarQube IP to private (no need to change ip once restart)

# NOTEFICATION & SLACKS

def COLOR\_MAP = [

'SUCCESS': 'good',

'FAILURE': 'danger',

]

pipeline {

agent any

tools {

maven "MAVEN3"

jdk "OracleJDK8"

}

stages{

stage('Print error'){

steps{

sh 'fake comment'

}

}

stage('Fetch code') {

steps{

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

}

}

stage('Build') {

steps {

sh 'mvn clean install -DskipTests'

}

post {

success {

echo "Now Archiving."

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

}

}

}

stage('Test'){

steps {

sh 'mvn test'

}

}

stage('Checkstyle Analysis'){

steps {

sh 'mvn checkstyle:checkstyle'

}

}

stage('Sonar Analysis') {

environment {

scannerHome = tool 'sonar4.7'

}

steps {

withSonarQubeEnv('sonar') {

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

-Dsonar.projectName=vprofile \

-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("UploadArtifact"){

steps{

nexusArtifactUploader(

nexusVersion: 'nexus3',

protocol: 'http',

nexusUrl: '172.31.18.28:8081',

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']

]

)

}

}

}

post {

always {

echo 'Slack Notifications.'

slackSend channel: '#jenkinscicd',

color: COLOR\_MAP[currentBuild.currentResult],

message: "\*${currentBuild.currentResult}:\* Job ${env.JOB\_NAME} build ${env.BUILD\_NUMBER} \n More info at: ${env.BUILD\_URL}"

}

}

}

## Note:

Slack is used for notification (used to integrate with it) 🡪 <https://slack.com/intl/en-in/get-started#/createnew>

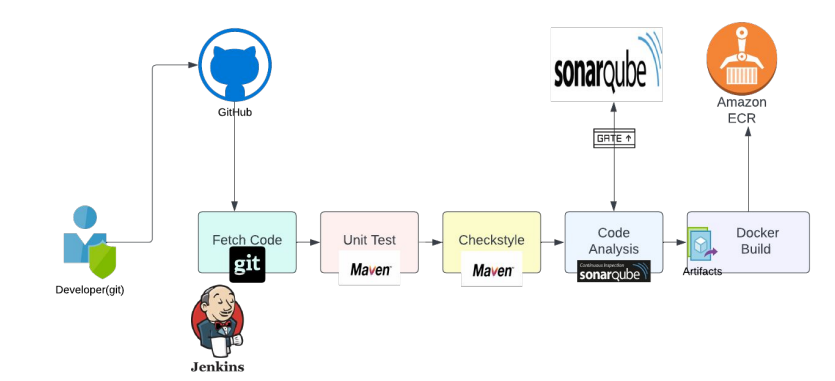
Create a workspace 🡪 name: vprofile 🡪working on devops cicd 🡪create a chennel (it like a group)

To create a token 🡪https://slack.com/apps/A0F7VRFKN-jenkins-ci 🡪add the slack🡪 channel name

Save the token

Jenkins 🡪 install the slack nofication plugin 🡪 add the token to secret text 🡪 give the channel name

# CI for DOCKER



In this we are using docker to containze the artifact

Develops develop the code and commit the github from github Jenkins will detect it fetect the code and will run unit test, code analysis by using checksytyle, sonar qube and upload the result into sonarqube server then we are go into build docker image.

ECR 🡪 Elastic container registry(AWS)

GCR 🡪 Google container registry(google)

Azure registory service

Docker hub

Nexus etc

# Docker PAAC Prereqs info

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')

}

}

}

}

}

}

IAM user with ECR permissions

Store aws creditentials in Jenkins

ECR repos on AWS

Plugin docker pipeline

ECR Plugin

Install docker engine on jenkins

## Docker ci in Jenkins

Install docker enginer in Jenkins

Add Jenkins user to docker group & reboot

Install AWS CLI

Create IAM USER

Create ECR repo

Plugin 🡪 ECR, Docker pipeline, aws sdk, cloudbee docker build and publish for credentials

In Jenkins server:

Sudo apt update && sudo apt install awscli -y

Install docker engine:

Run the following command to uninstall all conflicting packages:

**for** pkg in docker.io docker-doc docker-compose podman-docker containerd runc**;** **do** sudo apt-get remove $pkg**;** **done**

Set up Docker's Apt repository.

*# Add Docker's official GPG key:*

sudo apt-get update

sudo apt-get install ca-certificates curl gnupg

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

*# Add the repository to Apt sources:*

echo \

"deb [arch="**$(**dpkg --print-architecture**)**" signed-by=/etc/apt/keyrings/docker.gpg]

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

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

sudo apt-get update

Install the Docker packages.

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

## <https://docs.docker.com/engine/install/ubuntu/>

docker images

su -jenkins

docker images🡪 will get access denied because Jenkins is not on the group of docker

usermod -a -G docker Jenkins

id Jenkins

reboot

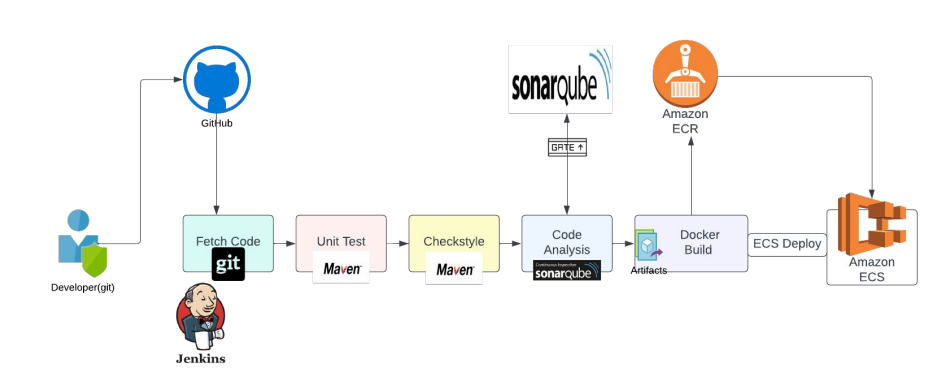
create a AWS IAM user with name – Jenkins: ECRFULLACCESS, AMAZONEC2containerRegistoryFullAcess

create a access key

Jenkins user🡪 security credentials🡪access key 🡪ceate accesskey🡪cli🡪create key🡪download key

Amazon ECR 🡪 create a repo 🡪 name jenkinsimg

# DOCKER CICD INTRO



We know ci pipeline upto now.

Now we need to host this docker image or the docker or the docker container to a docker solution like ECS.

ECS 🡪 Docker container hosting platform,

We package our images and then we host our dockerized application on ECS.

## Container hosting platform

Docker engine

We can run(docker run image) then your application on the docker engine but it good for testing and local devopment environment.

Kubernetes:

Kubernetes you can use for production environment due to less setting high availability self-healing etc.

-🡪 standalone, EKS,AKS, GKE, Openshift etc

AWS ECS : Elastic container service

pipeline {

agent any

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"

stage('Deploy to ecs') {

steps {

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

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

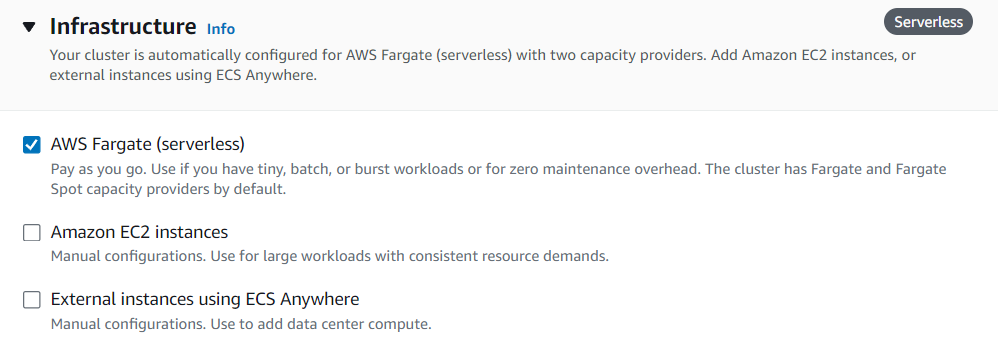
}

}

}

# ECS SETUP

Create a clusters



**Task definition** ( which provides all the information about container)

Provide the cpu, memory

Conatianer info 🡪ECR url and set the port(if service is running in tomcat use 8080 as default)

And check mark(**use log collection)**

Add the tag 🡪name : vprofileapptask

Create a task

In the above task the IAM role is created for default so

Task definition🡪ecs task execution role(in this defaultly you have **amazonecstaskexecutionrolepolicy** is they , add(Attach the roles) 🡪 the other role **cloudwatchlogsfullaccess**

**Create service**

Clusters 🡪service🡪create service

Here we an setup the application type, service type,networking, load balancer and autoscaling etc

And create it

Once service is create check the all the service is up and running like load balancer , health check ,ec2 instance etc

Cicd docker 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"

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'

// it going to create new container and old container will delete

// we can see ECS🡪cluster 🡪 vprofile🡪task🡪cleck the task🡪we can see the container

}

}

}

}

}

// **install plugin pipeline:steps**

# **Job Triggers**

1. Git Webhook

In github/git repository will send JSON playload whenever there is a commit in the repository.

1. Poll SCM

It is opposite to git webhook, in poll SCM Jenkins will check the commit in git repository. In an interval that you Specified.(like every 5 minutes or minutes or hour)

1. Scheduled jobs

You mention date and time like a alarm clock in crown job format and Jenkins will make sure the job will run in particular time or intervels.

1. Remote triggers

You can trigger Jenkins jobs from anywhere, from ascript, from a ansible playbook, from anywhere for that matter

1. Build after other projects are build

🡪after one job completed run the specified job

Step to do:

1. Create git repository on github 🡪 Jenkinstriggers
2. Ssh auth 🡪 ssh-keygen.exc
3. Create a jenkinsfile in git repo & commit
4. Create Jenkins job to access jenkinsfile from git repo
5. Test triggers

Create a git repository on github(jenkinstriggers) and

Generate the ssh key (ssh-keygen.exe)🡪it will create a public and private key 🡪 copy the public key and 🡪jenkins-setting🡪ssh and gpg key🡪add the key

Create a folder🡪intized git(git init)🡪git add . 🡪 commit the code or clone the repo

Create a Jenkinsfile and place the git repo 🡪 from they we can trigger the file by using webhooks, poll SCM, Schedules etc

If you getting “host key verification failed error”

🡪jenkins🡪manage Jenkins🡪security 🡪 git host key verification 🡪set accept first connection

Create a pipeline🡪pipeline script from SCM 🡪use git 🡪 add url and private key 🡪 add the Jenkinsfile path🡪run the job

pipeline{

agent any

stages{

stage('build'){

steps{

echo "this this jenkins build"

}

}

}

}

Webhooks:-

Copy Jenkins url 🡪go to github setting 🡪Webhooks 🡪 payload URL (<http://localhost:8080/github-webhook/> ) 🡪 content type ( application/json) 🡪 add webhook

In jenkinstrigger folder 🡪 (create folder🡪add , commit & push) 🡪 now job will run

Poll SCM :

It is smillar 🡪 min hour dom(1-31) month dow(1-7)

Scheduled jobs

Build periodly (simllar to Poll SCM)

Remote job:

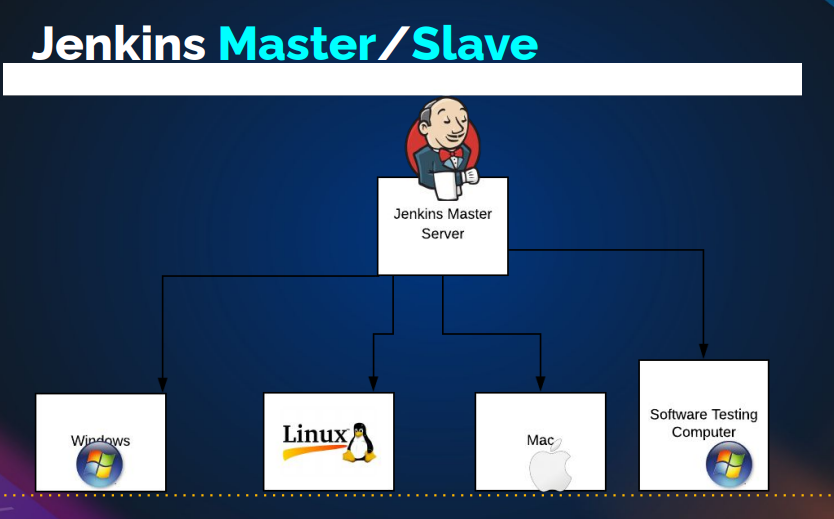
Usr url , token, crum 🡪 combine all create a curl command ( follow the document devops learning)

# Jenkins Master & Note

Distributed Builds, Cross Platform builds and much more

Use Cases

1. Load Distribution
2. Jenkins Master Executes Build Job on Node it selected.
3. Cross Platform Builds Executing Build of other platforms like .net(Windows), IOS(Mac OS) from Jenkins Master(Linux)
4. Software Testing Execute Testers Test Automation Scripts from Node.



Execute Anything

Can add any computer in network as Jenkins Node and execute commands on Nodes. e:g scripts, commands, test scripts, etc

Prerequisites for Node Setup

Any OS

2. Network access from Master Note: Check Firewall rules

3. Java, JRE, JDK

4. User

5. Directory with User ownership

6. Tools as required by the Jenkins job e:g Maven, Ant, Git etc

# Authentication & Authorization

Authentication 🡪 login

Authorization 🡪 privilege

Once the pipeline is created, need to hand it over to developer(so don’t share the login detail , don’t full access)

## Securing Jenkins

1. User Login
2. Jenkins own database.
3. Sign Up
4. LDAP Integration

### Permissions on Jenkins

● Admin

● Read

● Jobs

● Credentials

● Plugins etc

### Permissions on Jobs

● View

● Build

● Delete

● Configure

● etc