=======

JENKINS

=======

\* If you have a repetative tasks to be automated

- What to do

- When to run

- Where to run

\* Execute a task with authentication & user doesnt need to know the details

\* Ability to run a task on a regular intervals, on demand basis and also on scenario based

\* Include various steps in the task

\* an automation instead of us - go to a server, run applications/cmds

Advantages of jenkins:

\* connect & run the task automatically

\* periodically execute

\* dashboard

\* trigger a job based on a event

\* trigger on demand

\* group of machine

----------------------------Setup Master-----------------------

Install JDK 11

$ apt update

$ apt install -y openjdk-11-jre

Add the repository key to the system:

$ wget -q -O - https://pkg.jenkins.io/debian-stable/jenkins.io-2023.key | sudo apt-key add -

Append the Debian package repository:

$ sudo sh -c 'echo deb https://pkg.jenkins.io/debian-stable binary/ > /etc/apt/sources.list.d/jenkins.list'

Install Jenkins Package

$ apt update

$ apt install -y jenkins

Status of Jenkins

$ systemctl status jenkins | systemctl start jenkins

$ netstat -an | grep 8080

- Type the hostnamectl command :

$ sudo hostnamectl set-hostname jenkinsmaster

-------------------------Setup Slave-------------

$ apt update

$ apt install -y openjdk-11-jdk

- Type the hostnamectl command :

$ sudo hostnamectl set-hostname jenkinsslave

Jenkins Pipeline

================

(series of tasks done in order on different servers)

\* Pipeline as Code - declarative

\* DSL - Groovy scripts

1. scripted pipeline

2. declarative pipeline

\* Jenkinsfile - Gitlab repository

Advantages:

- collection of multiple freestyle jobs into 1 single pipeline job

- reuse the code

- single job can connect to multiple servers

- ability to call another job within a pipeline

- flexible

Syntax:

------

pipeline {

stages {

stage('stage1'){

agent {}

steps {}

} // end of stage1

stage('stage2'){

agent {}

steps {}

} // end of stage2

} // end of stages

} // end of pipeline

--------------------------

pipeline {

agent any

stages {

stage('Stage1') {

steps {

echo 'First Stage'

}

}

stage('Stage2') {

steps {

echo 'Second Stage'

}

}

}

}

-------------------------------

pipeline {

agent { label 'demo' }

stages {

stage('Stage1') {

steps {

echo 'First Stage'

}

}

stage('Stage2') {

steps {

echo 'Second Stage'

}

}

}

}

-------------------------------

pipeline {

agent none

stages {

stage('Stage1') {

agent { label 'demo' }

steps {

echo 'First Stage'

}

}

stage('Stage2') {

agent any

steps {

echo 'Second Stage'

}

}

}

}

------------------------

pipeline {

agent none

stages {

stage('Stage1') {

agent {

node {

label 'demo'

customWorkspace '/tmp/jenkins'

}

}

steps {

echo 'First Stage'

}

}

stage('Stage2') {

agent any

steps {

echo 'Second Stage'

}

}

}

}

-----------------------------------

pipeline {

agent { label 'demo' }

environment {

MYNAME = 'Adam'

}

stages {

stage('Stage1') {

steps {

sh " echo 'Your name: $MYNAME' "

}

}

stage('Stage2') {

steps {

echo env.MYNAME

}

}

}

}

------------------

pipeline {

agent { label 'demo' }

environment {

MYNAME = 'global'

}

stages {

stage('Stage1') {

environment {

MYNAME = 'local'

}

steps {

sh "echo 'Your name: $MYNAME'"

}

}

stage('Stage2') {

steps {

echo env.MYNAME

}

}

}

}

----------------

pipeline {

agent any

parameters {

string(name: 'PERSON', defaultValue: 'Mr Adam', description: 'Who are you?')

text(name: 'BIOGRAPHY', defaultValue: '', description: 'Enter some information about the person')

booleanParam(name: 'TOGGLE', defaultValue: true, description: 'Toggle this value')

choice(name: 'CHOICE', choices: ['One', 'Two', 'Three'], description: 'Pick something')

password(name: 'PASSWORD', defaultValue: 'SECRET', description: 'Enter a password')

file(name: "file.properties", description: "Choose a file to upload")

}

stages {

stage('Example') {

steps {

echo "Hello ${params.PERSON}"

echo "Biography: ${params.BIOGRAPHY}"

echo "Toggle: ${params.TOGGLE}"

echo "Choice: ${params.CHOICE}"

echo "Password: ${params.PASSWORD}"

}

}

}

}

---------------------

pipeline {

agent { label 'demo' }

options {

buildDiscarder(logRotator(numToKeepStr: '5'))

}

stages {

stage('Stage1') {

steps {

echo 'First Stage'

}

}

}

}

---------------------------

pipeline {

agent { label 'demo' }

options {

retry(3)

}

stages {

stage('Stage1') {

steps {

sh 'exit 1'

}

}

stage('stage2') {

steps {

sh 'echo Stage 2'

}

}

}

}

---------------------------

pipeline {

agent { label 'demo' }

stages {

stage('Stage1') {

options {

retry(3)

}

steps {

sh 'exit 1'

}

}

stage('stage2') {

steps {

sh 'echo Stage 2'

}

}

}

}

------------------

pipeline {

agent { label 'demo' }

options {

timeout(time: 15, unit: 'SECONDS')

timestamps()

}

stages {

stage('Stage1') {

steps {

echo "Stage 1"

sh 'sleep 5'

}

}

stage('Stage2') {

steps {

echo "Stage 2"

sh 'sleep 5'

}

}

}

}

-------------------------------

pipeline {

agent { label 'demo' }

stages {

stage('Clone Repo') {

steps {

echo 'Going to Checkout from Git'

git branch: 'main', changelog: false, credentialsId: 'gitlabCred', poll: false, url: 'https://gitlab.com/wezvaprojects/ansible/ssl-certs.git'

echo 'Completed Checkout from Git'

}

}

}

}

--------------------------------

pipeline {

agent { label 'demo' }

stages {

stage('Stage1') {

steps {

build job: 'secondJ', parameters: [string(name: 'MYNAME', value: 'ADAM')]

}

}

stage('Stage2') {

steps {

echo 'Testing'

}

}

}

}

-----------------------------

pipeline {

agent { label 'demo' }

stages {

stage('Stage1') {

steps {

sh 'touch testfirst'

dir('/tmp/jenkins') {

sh 'touch DUMMY'

}

sh 'touch testlast'

}

}

}

}

-------------------------------

pipeline {

agent any

stages {

stage('Stage1') {

steps {

catchError(buildResult: 'UNSTABLE', message: 'ERROR FOUND') {

sh 'exit 1'

}

}

}

stage('Stage2') {

steps {

echo 'Running Stage2'

}

}

}

}

---------------------------------

pipeline {

agent any

environment { DEPLOY\_TO = 'qa'}

stages {

stage('Stage1') {

when {

environment name: 'DEPLOY\_TO', value: 'qa'

}

steps {

echo 'Running Stage1 for QA'

}

}

stage('Stage2') {

when {

environment name: 'DEPLOY\_TO', value: 'production'

}

steps {

echo 'Running Stage2 for production'

}

}

}

}

--------------------------------

pipeline {

agent any

parameters {

booleanParam(name: 'TOGGLE', defaultValue: true, description: 'Toggle this value')

}

stages {

stage('Stage1') {

when {

expression { return params.TOGGLE }

}

steps {

echo 'Testing'

}

}

}

}

--------------------------------

pipeline {

agent any

parameters {

string(name: 'PERSON', defaultValue: 'Mr Adam', description: 'Who are you?')

}

stages {

stage('Stage1') {

when { equals expected: 'adam' , actual: params.PERSON }

steps {

echo 'Hi Adam !!'

}

}

}

}

-------------------------------

COND1 AND COND2

True True - True

False True - False

True False - False

COND1 OR COND2

True True - True

False True - True

True False - True

False False - False

pipeline {

agent any

parameters {

string(name: 'PERSON', defaultValue: 'Mr Adam', description: 'Who are you?')

booleanParam(name: 'TOGGLE', defaultValue: true, description: 'Toggle this value')

}

stages {

stage('Stage1') {

when {

allOf {

equals expected: 'adam' , actual: params.PERSON

expression { return params.TOGGLE }

}

}

steps {

echo 'Hi Adam !!'

}

}

}

}

---------------------------

pipeline {

agent any

parameters {

string(name: 'PERSON', defaultValue: 'Mr Adam', description: 'Who are you?')

booleanParam(name: 'TOGGLE', defaultValue: true, description: 'Toggle this value')

}

stages {

stage('Stage1') {

when {

anyOf {

equals expected: 'adam' , actual: params.PERSON

expression { return params.TOGGLE }

}

}

steps {

echo 'Hi Adam !!'

}

}

}

}

----------------------

pipeline {

agent any

stages{

stage('stage1'){

steps { echo "stage1"}

}

stage('stage2'){

steps { echo "stage1"}

}

}

post {

always{ echo "Post Stage"}

}

}

---------------------

pipeline {

agent any

stages{

stage('stage1'){

steps { echo "stage1"}

post {

always { echo "Post Stage1" }

}

}

stage('stage2'){

steps { echo "stage2"}

}

}

}

Change Approval Board (CAB):

- Date/time : maintenance window

- Severity

- Rollback mechanism

- Impact to customers

Single Sing-on (SSO) - LDAP/Active Directory

SCA - Static code analysis

Checkov - scan the iac for misconfigurations and coding standard

Ansible HA Project:

- Ensure HA for Ansible pipeline

- We can pass the vault credentials through the Jenkins plugin

- We can also pass the ssh credentials for connecting to the ansible nodes

(no need for passwordless ssh or hardcoding credentials inside hosts file)

===============

DevOps Projects [ Pipeline/Jenkins ]

===============

\* Enabled DevSecops pipelines for Infra provisioning for catching the IAC misconfigurations and to handle approvals automatically

\* Enabled DevSecops pipeline for Configurations of different servers and also to setup HA

\* Developed playbook to setup Terraform core, checkov, java

Code -> Build -> Deploy

Types of Build

--------------

\* Nightly/Full Builds:

- Runs periodically at an regular interval or also on need-basis

- Output of the full build is a full product which is a jar/war/tar/zip deliverable

- the deliverable is consumed by QA for functional/integration/regression/UAT testing

- full builds will be set for Feature branches, Integration Branch & Release Branch

- we take new workspace & do clean build

NOTE: we will take/checkout the last successful CI build commit ID to run the full build

\* Continuous Integration Builds:

# purpose is to help developers identify faulty code as soon as possible (i.e tell whether code is good & whether it can be given to QA for functional testing)

# faster release cycles by helping in continuous testing

- Runs for every commit pushed to central repo

- CI builds will be set on Feature branches only

- Incremental builds, we reuse the workspace

- Output is to notify developers if there is a problem in the code

BUILD PIPELINE

--------------

pipeline {

agent none

stages{

stage('Checkout')

{

agent { label 'demo' }

steps {

git credentialsId: 'GitlabCred', url: 'https://gitlab.com/wezvatechprojects/wezvatech-cicd.git'

}

}

stage('Build')

{

agent { label 'demo' }

steps {

echo "Building Jar Component ..."

dir ("./samplejar") {

sh "export JAVA\_HOME=/usr/lib/jvm/java-8-openjdk-amd64; mvn package"

}

echo "Building War Component ..."

dir ("./samplewar") {

sh "export JAVA\_HOME=/usr/lib/jvm/java-8-openjdk-amd64; mvn package"

}

}

}

stage('Code Coverage')

{

agent { label 'demo' }

steps {

echo "Running Code Coverage ..."

dir ("./samplejar") {

sh "mvn org.jacoco:jacoco-maven-plugin:0.5.5.201112152213:prepare-agent"

}

}

}

stage('Push Artifacts')

{

agent { label 'demo' }

steps {

script {

/\* Define the Artifactory Server details \*/

def server = Artifactory.server 'wezvatechjfrog'

def uploadSpec = """{

"files": [{

"pattern": "samplewar/target/samplewar.war",

"target": "wezvatech"

}]

}"""

/\* Upload the war to Artifactory repo \*/

server.upload(uploadSpec)

}

}

}

} // end of stages

} // end of pipeline

\* Build: mvn package

- compilation

- unit test

- package

\* Code coverage: its a measurement of how much of code has been tested

- we will know whether we need more test cases

- we will know whether there are dead codes

- For Java project we use Jacoco

Install Artifactory using Docker:

--------------------------------------

$ mkdir -p ~/jfrog/artifactory/var/etc

$ chmod -R 777 ~/jfrog

$ touch ~/jfrog/artifactory/var/etc/system.yaml

$ chown -R 1030:1030 ~/jfrog/artifactory/var

$ docker run --name artifactory -v ~/jfrog/artifactory/var:/var/opt/jfrog/artifactory -d -p 8081:8081 -p 8082:8082 releases-docker.jfrog.io/jfrog/artifactory-oss:latest

--\*\*NOTE: If OSS Image is not available use the below Community Edition Image \*\* --

$ docker run --name artifactory -v ~/jfrog/artifactory/var/:/var/opt/jfrog/artifactory -d -p 8081:8081 -p 8082:8082 releases-docker.jfrog.io/jfrog/artifactory-cpp-ce:latest

- for jfrog artifactory default cred: admin/password

GENERIC BUILD PIPELINE - FULL/CI BUILD

---------------------------------------

pipeline {

agent none

stages{

stage('Checkout')

{

agent { label 'demo' }

steps {

git credentialsId: 'GitlabCred', url: 'https://gitlab.com/wezvatechprojects/wezvatech-cicd.git'

}

}

stage('GitLabWebHookCause') {

when {

beforeAgent true

triggeredBy 'GitLabWebHookCause'

}

steps {

echo "I am only executed when triggered by SCM push"

script {

env.BUILDTYPE = "CI" // Set env variable to enable further Build Stages

}

}

}

stage('ManualTimed') {

when {

beforeAgent true

anyOf {

triggeredBy 'TimerTrigger'

triggeredBy cause: 'UserIdCause'

}

}

steps {

echo "I am only executed when triggered manually or timed"

script {

env.BUILDTYPE = "FULL" // Set env variable to enable further Build Stages

}

}

}

stage('Validate')

{

agent { label 'demo' }

when {

environment name: 'BUILDTYPE', value: 'CI'

anyOf {

changeset "samplejar/\*\*"

changeset "samplewar/\*\*"

}

}

steps {

script {

env.BUILDME = "yes" // Set env variable to enable further Build Stages

}

}

}

stage('Build')

{

when {

anyOf {

environment name: 'BUILDME', value: 'yes'

environment name: 'BUILDTYPE', value: 'FULL'

}

}

agent { label 'demo' }

steps {

echo "Building Jar Component ..."

dir ("./samplejar") {

sh "export JAVA\_HOME=/usr/lib/jvm/java-8-openjdk-amd64; mvn package"

}

echo "Building War Component ..."

dir ("./samplewar") {

sh "export JAVA\_HOME=/usr/lib/jvm/java-8-openjdk-amd64; mvn package"

}

}

}

stage('Code Coverage')

{

when {

anyOf {

environment name: 'BUILDME', value: 'yes'

environment name: 'BUILDTYPE', value: 'FULL'

}

}

agent { label 'demo' }

steps {

echo "Running Code Coverage ..."

dir ("./samplejar") {

sh "mvn org.jacoco:jacoco-maven-plugin:0.5.5.201112152213:prepare-agent"

}

}

}

stage('Stage Artifacts')

{

when {

anyOf {

environment name: 'BUILDME', value: 'yes'

environment name: 'BUILDTYPE', value: 'FULL'

}

}

agent { label 'demo' }

steps {

script {

/\* Define the Artifactory Server details \*/

def server = Artifactory.server 'wezvatechjfrog'

def uploadSpec = """{

"files": [{

"pattern": "samplewar/target/samplewar.war",

"target": "wezvatech"

}]

}"""

/\* Upload the war to Artifactory repo \*/

server.upload(uploadSpec)

}

}

}

} // end of stages

} // end of pipeline

Containerize Java web application

---------------------------------

\* Application server - Jboss, tomcat

\* JRE - Java runtime environment

\* JDK - Java development kit

\* JVM - java virtual machine

\* Heap memory - min & max memory needed to run the java app inside the jvm

-xmx -xms

\* Config file - db details, log details (app log, server log, user log), heap memory details, dependent service details

\* Certificates

\* Scripts - startup, stop, healthcheck

---Setup ECR registry----

provider "aws" {

region = "ap-south-1"

}

# Create indivual private repository per project

resource "aws\_ecr\_repository" "example" {

name = "wezvabaseimage" # name of the repo/project

image\_tag\_mutability = "MUTABLE"

image\_scanning\_configuration {

scan\_on\_push = true

# ECR uses Clair

}

}

https://437030480074.dkr.ecr.ap-south-1.amazonaws.com/wezvabaseimage

--------- Jenkins plugin needed for Docker Image Pipeline: ------

1. Docker Pipeline

2. Amazon ECR Plugin

3. Pipeline: AWS steps

Docker Image DevSecOps Pipeline

================================

\* SCA/Lint - Code analysis for Dockerfile standards - Hadolint

\* Image Efficiency Verification - Analyze the layers of Image & give the space efficiency - Dive

\* Image scanner - Scan Image for vulnerabilities - Clair

\* Quality Gates - Rules to decide whether the Image is good for consumption

Full Build Pipeline (DevSecOps Enabled) - Take a large build server i.e t2.medium with 15GB HDD

---------------------------------------

pipeline {

agent none

parameters {

booleanParam(name: 'CLEANBUILD', defaultValue: true, description: 'Enable Clean Build ?')

string(name: 'ECRURL', defaultValue: '437030480074.dkr.ecr.ap-south-1.amazonaws.com', description: 'Please Enter your Docker ECR REGISTRY URL without https?')

string(name: 'BASEREPO', defaultValue: 'wezvabaseimage', description: 'Please Enter your Docker Base Repo Name?')

string(name: 'APPREPO', defaultValue: 'wezvaappimage', description: 'Please Enter your Docker App Repo Name?')

string(name: 'REGION', defaultValue: 'ap-south-1', description: 'Please Enter your AWS Region?')

}

stages{

stage('Checkout')

{

agent { label 'demo' }

steps {

git credentialsId: 'GitlabCred', url: 'https://gitlab.com/wezvatechprojects/wezvatech-cicd.git'

}

}

stage('GitLabWebHookCause') {

when {

beforeAgent true

triggeredBy 'GitLabWebHookCause'

}

steps {

echo "I am only executed when triggered by SCM push"

script {

env.BUILDTYPE = "CI" // Set env variable to enable further Build Stages

}

}

}

stage('ManualTimed') {

when {

beforeAgent true

anyOf {

triggeredBy 'TimerTrigger'

triggeredBy cause: 'UserIdCause'

}

}

steps {

echo "I am only executed when triggered manually or timed"

script {

env.BUILDTYPE = "FULL" // Set env variable to enable further Build Stages

}

}

}

stage('Validate')

{

agent { label 'demo' }

when {

environment name: 'BUILDTYPE', value: 'CI'

anyOf {

changeset "samplejar/\*\*"

changeset "samplewar/\*\*"

}

}

steps {

script {

env.BUILDME = "yes" // Set env variable to enable further Build Stages

}

}

}

stage('Build')

{

when {

anyOf {

environment name: 'BUILDME', value: 'yes'

environment name: 'BUILDTYPE', value: 'FULL'

}

}

agent { label 'demo' }

steps {

echo "Building Jar Component ..."

dir ("./samplejar") {

sh "export JAVA\_HOME=/usr/lib/jvm/java-8-openjdk-amd64; mvn package"

}

echo "Building War Component ..."

dir ("./samplewar") {

sh "export JAVA\_HOME=/usr/lib/jvm/java-8-openjdk-amd64; mvn package"

}

}

}

stage('Code Coverage')

{

when {

anyOf {

environment name: 'BUILDME', value: 'yes'

environment name: 'BUILDTYPE', value: 'FULL'

}

}

agent { label 'demo' }

steps {

echo "Running Code Coverage ..."

dir ("./samplejar") {

sh "mvn org.jacoco:jacoco-maven-plugin:0.5.5.201112152213:prepare-agent"

}

}

}

stage('SCA')

{

agent { label 'demo' }

when {

environment name: 'BUILDTYPE', value: 'FULL'

}

steps {

echo "Running Software Composition Analysis using OWASP Dependency-Check ..."

dir ("./samplejar") {

sh "mvn org.owasp:dependency-check-maven:check"

}

}

}

stage('SAST')

{

agent { label 'demo' }

when {

environment name: 'BUILDTYPE', value: 'FULL'

}

steps{

echo "Running Static application security testing using SonarQube Scanner ..."

withSonarQubeEnv('mysonarqube') {

dir ("./samplejar") {

sh 'mvn sonar:sonar -Dsonar.dependencyCheck.jsonReportPath=target/dependency-check-report.json -Dsonar.dependencyCheck.htmlReportPath=target/dependency-check-report.html'

}

}

}

}

stage("Quality Gate"){

agent { label 'demo' }

when {

environment name: 'BUILDTYPE', value: 'FULL'

}

steps{

script {

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

def qg = waitForQualityGate()

if (qg.status != 'OK') {

error "Pipeline aborted due to quality gate failure: ${qg.status}"

}

}

}

}

}

stage('Store Artifacts')

{

when {

anyOf {

environment name: 'BUILDME', value: 'yes'

environment name: 'BUILDTYPE', value: 'FULL'

}

}

agent { label 'demo' }

steps {

script {

/\* Define the Artifactory Server details \*/

def server = Artifactory.server 'wezvatechjfrog'

def uploadSpec = """{

"files": [{

"pattern": "samplewar/target/samplewar.war",

"target": "wezvatech"

}]

}"""

/\* Upload the war to Artifactory repo \*/

server.upload(uploadSpec)

}

}

}

stage('Build Image')

{

agent { label 'demo' }

when {

environment name: 'BUILDTYPE', value: 'FULL'

}

steps{

script {

// Prepare the Tag name for the Image

AppTag = params.APPREPO + ":" + env.BUILD\_ID

BaseTag = params.ECRURL + "/" + params.BASEREPO

// Docker login needs https appended

ECR = "https://" + params.ECRURL

docker.withRegistry( ECR, 'ecr:ap-south-1:AWSCred' ) {

/\* Build Docker Image locally \*/

myImage = docker.build(AppTag, "--build-arg BASEIMAGE=${BaseTag} .")

/\* Push the Image to the Registry \*/

myImage.push()

}

}

}

}

stage ('Smoke Deploy'){

agent {label 'demo'}

when {

environment name: 'BUILDTYPE', value: 'FULL'

}

steps {

script {

env.DEPLOYIMAGE = params.APPREPO + ":" + env.BUILD\_ID

// Create Containers using the recent Build Image

sh ("export DEPLOYIMAGE=${DEPLOYIMAGE}; docker-compose up -d")

}

}

}

stage ('DAST'){

agent {label 'demo'}

when {

environment name: 'BUILDTYPE', value: 'FULL'

}

steps {

echo "Running Dynamic application security testing using OWASP-ZAP ..."

sh "docker run --rm -t owasp/zap2docker-stable zap-baseline.py -t http://`hostname -i|awk '{print \$1}'`:8080/samplewar -I"

}

}

stage ('Smoke Test'){

agent {label 'demo'}

when {

environment name: 'BUILDTYPE', value: 'FULL'

}

steps {

catchError(buildResult: 'SUCCESS', message: 'TEST-CASES FAILED', stageResult: 'UNSTABLE')

{

sh "sleep 10; chmod +x runsmokes.sh; ./runsmokes.sh"

}

}

post {

always {

script {

env.DEPLOYIMAGE = params.APPREPO + ":" + env.BUILD\_ID

// Create Containers using the recent Build Image

sh ("export DEPLOYIMAGE=${DEPLOYIMAGE}; docker-compose down")

sh ("docker rmi ${params.APPREPO}:${env.BUILD\_ID}")

}

}

}

}

} // end of stages

} // end of pipeline

http://ec2-43-205-112-115.ap-south-1.compute.amazonaws.com:8080/sonarqube-webhook/

USING SCRIPTS

-------------

- startup/stop scripts

- health scripts

- quality gate script

- ansible upgrade

- run smoke test

DevSecOps Build Pipeline

========================

\* SCA - Static code analysis - OWASP Dependency check

- Coding standards, 3rd party dependencies; false positives and false negative

\* SAST (Static Application Security Testing) - Sonarscanner, Sonarqube

- Memory leaks, endless loops, falling into unknown states, unhandled errors and other exceptions, and so many more potential serious security vulnerabilities are easily found with SAST products.

\* DAST (Dynamic Application Security Testing) - OWASP ZAP

\* Ensure dependency check plugin is added in pom.xml

\* Add properties in pom.xml to refer to the location of the dependency check report file

\* Installing the SonarQube Dependency-Check plugin

- Goto Sonarqube - Administration and click on Marketplace - Click understand the risk

- In the plugins search box put "Dependency-Check" & install the plugin

- restart sonarqube'

\* Enable webhook in Sonarqube for the QualityGate plugin

- http:<JenkinsURL>:8080/sonarqube-webhook/

\* Take a bigger build server (t2.medium)

$ sync; echo 1 > /proc/sys/vm/drop\_caches

$ sync; echo 2 > /proc/sys/vm/drop\_caches

$ sync; echo 3 > /proc/sys/vm/drop\_caches

$ docker run -d --name sonarqube -p 9000:9000 sonarqube

- Default credentials: admin/admin

Jenkins Plugins Used

------------------------

\* Git plugin, JDK plugin

\* Parameterized trigger plugin

\* Ansible plugin

\* Gitlab plugin

\* Artifactory plugin

\* Docker Pipeline

\* Amazon ECR Plugin

\* Pipeline: AWS steps

\* SonarQube Scanner

\* Quality Gates

\* Prometheus metrics

HOW TO RUN AN APPLICATION IN KUBERNETES

=======================================

\* What namespace to use

\* How to create a pod

- how many containers are needed (Init, Main, Sidecar)

- how much of resources needed to run the containers

- what healthcheck to run (liveness, readiness)

- what config files are needed

- what secrets are needed

- what ports to be exposed

- what environment variables are cmds

- what volumes to use (logs, persistent volume, claim)

- what is the startup cmd or arguments for the startup cmd

\* How to group the objects

\* How to run a pod

- replication/scale (stateless/stateful)

- number of replicas

- do we need autoscaling

\* How to access the pod

- how to access the port (service)

\* What files are needed to start the application/container

Kubernetes Objects for our Java App

------------------------------------

\* Namespace

\* Deployment object

\*\* POD

- replicas

- Image

- resources

- liveness & readiness

- port map

- env

- label

- ImagePullSecret

- Volumes

\* Service

\* CongifMap

\* Secret

\* PV & PVC

\* HPA & Metric server

$ minikube addons enable registry-creds

$ minikube addons configure registry-creds

$ kubectl get secrets

NAME TYPE DATA AGE

awsecr-cred kubernetes.io/dockerconfigjson 1 13s

GITOPS

=======

\* Deployment will be secure

\* Trigger deployments for scenario:

- when new build is triggered, new image will be deployed

- deploy a old image

- modify configmap/secret & deploy

$ kubectl create namespace argocd

$ kubectl apply -n argocd -f https://raw.githubusercontent.com/argoproj/argo-cd/stable/manifests/install.yaml

$ kubectl patch svc argocd-server -n argocd -p '{"spec": {"type": "NodePort"}}'

For version 1.9 or later:

$ kubectl -n argocd get secret argocd-initial-admin-secret -o jsonpath="{.data.password}" | base64 -d && echo

CICD

====

- Add this parameter in the begining of your pipeline

password(name: 'PASSWD', defaultValue: '', description: 'Please Enter your Gitlab password')

stage ('Trigger CD'){

agent {label 'demo'}

when {

environment name: 'BUILDTYPE', value: 'FULL'

}

steps {

script {

TAG = '\\/' + params.APPREPO + ":" + env.BUILD\_ID

build job: 'Deployment\_Pipeline', parameters: [string(name: 'IMAGE', value: TAG), password(name: 'PASSWD', value: params.PASSWD)]

}

}

}

CAPSTONE PROJECT

================

\* Enable continous delivery (CICD) - anytime build/deployment

- Continuous Development

- Continuous Build

- Continuous Deployment

- Continuous Testing

1. Plan

2. Development

Build Automation (Feature branch, integration branch, release branch)

3. Testing

- Functional testing (QA env)

- Integration testing (QA env)

- Regression testing (QA env)

- User acceptance testing (Stage env)

4. Release

- Production

Sub Projects

------------

\* Build Pipeline (CI & Full Build)

\* Containerize the Java application

- Develop platform base image

- Develop application image

\* Enable Devsecops for Build pipeline, Base/App Image Pipeline

\* Deployment Pipeline (CD)

- Develop manifest file

- Enable Devsecops or developed deployment pipeline using Gitops approach

\* Combine CI & CD

Jenkins Plugins Used

------------------------

\* Git plugin, JDK plugin

\* Parameterized trigger plugin

\* Ansible plugin

\* Gitlab plugin

\* Artifactory plugin

\* Docker Pipeline

\* Amazon ECR Plugin

\* Pipeline: AWS steps

\* SonarQube Scanner

\* Quality Gates

\* Prometheus metrics

Prometheus

===========

\* 3 parts of monitoring solution

- Collection of data (prometheus server)

- visualization of data (grafana)

- alerting/notification (alert manager)

Create a user for Prometheus on your system

$ useradd -rs /bin/false prometheus

Create a new folder and a new configuration file for Prometheus

$ mkdir /etc/prometheus

$ touch /etc/prometheus/prometheus.yml

Create a data folder for Prometheus

$ mkdir -p /data/prometheus

$ chown prometheus:prometheus /data/prometheus /etc/prometheus/\*

$ vi /etc/prometheus/prometheus.yml

global:

scrape\_interval: 5s

evaluation\_interval: 1m

# A scrape configuration scraping a Node Exporter and the Prometheus server itself

scrape\_configs:

# Scrape Prometheus itself every 10 seconds.

- job\_name: 'prometheus'

scrape\_interval: 10s

static\_configs:

- targets: ['localhost:9090']

Get the userid for running Prometheus

$ cat /etc/passwd | grep prometheus

Create the Prometheus container

$ docker run --name myprom -d -p 9090:9090 --user 997:997 --net=host -v /etc/prometheus:/etc/prometheus -v /data/prometheus:/data/prometheus prom/prometheus --config.file="/etc/prometheus/prometheus.yml" --storage.tsdb.path="/data/prometheus"

Create a Grafana container

$ docker run --name grafana -d -p 3000:3000 --net=host grafana/grafana

---------------Monitor A Node -------

Create a user for Node Exporter

$ useradd -rs /bin/false node\_exporter

$ cat /etc/passwd | grep node\_exporter

Creating Node exporter container

$ docker run --name exporter -d -p 9100:9100 --user 997:997 -v "/:/hostfs" --net="host" prom/node-exporter --path.rootfs=/hostfs

$ vi /etc/prometheus/prometheus.yml

- job\_name: 'BuildMachine01'

static\_configs:

- targets: ['172.31.6.222:9100']

Restart Prometheus, get the PID & send SIGHUP signal

$ ps aux | grep prometheus

$ kill -HUP <PID>

To test Increase the memory

$ apt update

$ apt install -y stress

$ stress --vm 1 --vm-bytes 100M

\* User 1860 grafana dashboard

----------------------Monitor Jenkins------------

- job\_name: 'Jenkins'

metrics\_path: /prometheus

static\_configs:

- targets: ['172.31.5.213:8080']

\* Use "9964" Dashboard Id in Grafana

---------------------Monitor Containers---------

Run Cadvisor container to collect container metrics

$ docker run --name cadvisor -d -p 8080:8080 -v /:/rootfs:ro -v /var/run:/var/run:rw -v /sys:/sys:ro -v /var/lib/docker/:/var/lib/docker:ro google/cadvisor

Edit /etc/prometheus/prometheus.yml & add a job for Cadvisor:

- job\_name: 'cadvisor'

static\_configs:

- targets: ['localhost:8080']

\* Use 893 grafana dashboard

----------------------Monitor an Application: TOMCAT -----------

Download Java JMX Exporter jar

$ wget https://repo1.maven.org/maven2/io/prometheus/jmx/jmx\_prometheus\_javaagent/0.12.0/jmx\_prometheus\_javaagent-0.12.0.jar

Create a config file prometheus-jmx-config.yaml to expose all metrics

---

startDelaySeconds: 0

ssl: false

lowercaseOutputName: false

lowercaseOutputLabelNames: false

From tomcat:latest Docker Image, copy /usr/local/tomcat/bin/catalina.sh locally and add JVM Parameter to your application

$ docker run --name test --rm -d tomcat

$ docker cp test:/usr/local/tomcat/bin/catalina.sh .

$ vi catalina.sh

JMX\_OPTS="-javaagent:/data/jmx\_prometheus\_javaagent-0.12.0.jar=8081:/data/prometheus-jmx-config.yaml"

JAVA\_OPTS="$JMX\_OPTS $JAVA\_OPTS $JSSE\_OPTS"

- Create a Dockerfile

FROM tomcat

RUN mkdir /data

COPY catalina.sh /usr/local/tomcat/bin/catalina.sh

ADD jmx\_prometheus\_javaagent-0.12.0.jar /data/jmx\_prometheus\_javaagent-0.12.0.jar

ADD prometheus-jmx-config.yaml /data/prometheus-jmx-config.yaml

EXPOSE 8081

Create a Tomcat container

$ docker build -t jmxtomcat .

$ docker run --name jmxtomcat --rm -d -p 8080:8080 -p 8081:8081 jmxtomcat

Edit /etc/prometheus/prometheus.yml & add a job for JMX, restart Prometheus

- job\_name: 'JMX'

static\_configs:

- targets: ['172.31.47.62:8081']

\* Use 3066 grafana dashboard

====== Alert Manager ==============

$ wget https://github.com/prometheus/alertmanager/releases/download/v0.20.0/alertmanager-0.20.0.linux-amd64.tar.gz

$ tar xzvf alertmanager-0.20.0.linux-amd64.tar.gz

\* Setup Gmail alerts:

- Make sure 2-Step Verification is enabled

- Account Settings -> Security -> Signing in to Google -> App password

\* Edit alertmanager.yml & add:

global:

resolve\_timeout: 1m

route:

receiver: 'gmail-notifications'

receivers:

- name: 'gmail-notifications'

email\_configs:

- to: scmlearningcentre@gmail.com

from: scmlearningcentre@gmail.com

smarthost: smtp.gmail.com:587

auth\_username: scmlearningcentre@gmail.com

auth\_identity: scmlearningcentre@gmail.com

auth\_password: <your gmail App password>

send\_resolved: true

$ nohup ./alertmanager --config.file=alertmanager.yml 2>&1 &

\* Edit /etc/prometheus/prometheus.yml & add:

rule\_files:

- /etc/prometheus/rules.yml

alerting:

alertmanagers:

- static\_configs:

- targets:

- 'localhost:9093'

- Edit /etc/prometheus/rules.yml & add sample rule:

groups:

- name: default

rules:

- alert: InstanceDown

expr: up == 0

for: 1m

labels:

severity: critical

annotations:

summary: "Instance is Down"

description: "Instance is down since last 1 min"

--------------------------------Monitor K8s Cluster------------

$ curl https://get.helm.sh/helm-v3.2.3-linux-amd64.tar.gz > helm.tar.gz

$ tar xzvf helm.tar.gz

$ mv linux-amd64/helm /usr/local/bin

\* Kubernetes Master/Controlplane

- etcd/dns/api-server/controller

\* Kubernetes Nodes/Dataplane - worker/master

\* Namespace/Pod

$ helm repo add prometheus-community https://prometheus-community.github.io/helm-charts

$ helm repo update

$ helm pull --untar prometheus-community/kube-prometheus-stack

1. Edit kube-prometheus-stack/charts/grafana/values.yaml, set the values under Service key:

type: NodePort

port: 3000

2. Edit kube-prometheus-stack/values.yaml, search for "Configuration for Prometheus service" under this set the value of service(s)

type: NodePort

3. Remove charts/kube-state-metrics dir & entry from Charts.yaml

$ helm install myprom .

To get password for user "admin"

$ kubectl get secret | grep grafana

$ kubectl get secret myprom-grafana -o jsonpath='{.data.admin-password}' | base64 --decode

( prom-operator )

Centralized Log Management (server/app/access)

==========================

\* Log management using EFK

\* Fluentd - is a log agrregrator for pods, which reads logs from pods and redirects

\* Elasticsearch - is a nosql db to store all the logs & also a search engine

\* Kibana - is a data visualization dashboard

Fluentd - daemonset - ClusterIP

Elasticsearch - Statefulset - ClusterIP

Kibana - Deployment - NodePort

$ git clone https://github.com/cdwv/efk-stack-helm

$ cd efk-stack-helm

- Edit values.yaml, set the below values for rbac & kibana service type:

rbac:

enabled: true

service: # this is for kibana configuration

type: NodePort

- Edit Chart.yaml & add below line

version: 0.0.1

- Edit templates/kibana-deployment.yaml & change the apiversion

apiVersion: apps/v1

$ helm install demoefk .

kind: Pod

apiVersion: v1

metadata:

name: testpod

spec:

containers:

- name: c00

image: ubuntu

command: ["/bin/bash", "-c", "while true; do echo Hello-Adam-`date`; sleep 5 ; done"]

restartPolicy: Never

Clear Cache in Ubuntu:

$ sync; echo 1 > /proc/sys/vm/drop\_caches

$ sync; echo 2 > /proc/sys/vm/drop\_caches

$ sync; echo 3 > /proc/sys/vm/drop\_caches

Setup a Kubernetes Cluster - Self Managed

=========================================

Setup Master on AWS EC2 – Ubuntu (2 cpu):

Install Docker CE

$ apt update

$ apt install -y docker.io

Add the repo for Kubernetes

$ curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | apt-key add -

$ cat << EOF > /etc/apt/sources.list.d/kubernetes.list

deb http://apt.kubernetes.io/ kubernetes-xenial main

EOF

Install Kubernetes components

$ apt update

$ apt install -y kubeadm kubelet

Initialize the cluster using the IP range for Flannel.

$ kubeadm init --pod-network-cidr=10.244.0.0/16

-- Copy the kubeadmin join command that is in the output. We will need this later.

-- Add port obtained from above cmd (6443) in the Inbound rules

Exit sudo and copy the admin.conf to your home directory and take ownership as normal user

$ mkdir -p $HOME/.kube

$ sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config

$ sudo chown $(id -u):$(id -g) $HOME/.kube/config

You should now deploy a pod network to the cluster

$ kubectl apply -f https://raw.githubusercontent.com/coreos/flannel/master/Documentation/kube-flannel.yml

====== Worker ========

Setup node on AWS EC2 – Ubuntu: (t2.micro)

Install Docker CE

$ apt update

$ apt install -y docker.io

Add the repo for Kubernetes

$ curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | apt-key add -

$ cat << EOF > /etc/apt/sources.list.d/kubernetes.list

deb http://apt.kubernetes.io/ kubernetes-xenial main

EOF

Install Kubernetes components

$ apt update

$ apt install -y kubeadm kubelet

Join the cluster by running the output cmd obtained from ‘kubeadm init’ on master as root:

CAPSTONE PROJECT

================

\* Enable continous delivery (CICD) - anytime build/deployment

- Continuous Development

- Continuous Build

- Continuous Deployment

- Continuous Testing

1. Plan

2. Development

Build Automation (Feature branch, integration branch, release branch)

3. Testing

- Functional testing (QA env)

- Integration testing (QA env)

- Regression testing (QA env)

- User acceptance testing (Stage env)

4. Release

- Production

Sub Projects

------------

\* Build Pipeline (CI & Full Build)

\* Containerize the Java application

- Develop platform base image

- Develop application image

\* Enable Devsecops for Build pipeline, Base/App Image Pipeline

\* Deployment Pipeline (CD)

- Develop manifest file

- Enable Devsecops or developed deployment pipeline using Gitops approach

\* Combine CI & CD

Jenkins Plugins Used

------------------------

\* Git plugin, JDK plugin

\* Parameterized trigger plugin

\* Ansible plugin

\* Gitlab plugin

\* Artifactory plugin

\* Docker Pipeline

\* Amazon ECR Plugin

\* Pipeline: AWS steps

\* SonarQube Scanner

\* Quality Gates

\* Prometheus metrics

\* IAC - Terraform

\* CAC - Ansible

\* DVCS - Gitlab, Git

\* Pipeline - Jenkins

\* Containerization - Docker

\* Orchestration of Containers - Kubernetes

\* Continuous Monitoring - Prometheus, Grafana, EFK

\* Continuous Delivery (Gitops) - ArgoCD

\* DevSecops - Checkov, Hashicorp vault, haldo, dive, clair, Sonarqube, OSWASP Zap

\* Cloud - AWS

===============

DevOps Projects - IAC/Terraform

===============

\* Developed modules for network to support public(frontend) & private(backend) infra

\* Developed modules for Auto-scaling, load balancers(ALB) to support high availability

\* Developed modules for creating user profile in AWS & assign dev/qa/ops policy

\* Developed module for EC2 provisioning

\* Developed modules for Ansible controller & Ansible target node setup

===============

DevOps Projects [ CAC/Ansible ]

===============

\* Developed Roles/Playbooks to Setup & verify LAMP stack on Dev servers to help web developers with their development/building web applications activities

\* Developed Roles/Playbooks to Enable automatic compression, rotation, deletion of application & system logs on required servers using logrotate

\* Developed Roles/Playbooks to Generate self signed Root & Client SSL Certificates for microservices and to Verify the certificate expiration & to renew it

\* Combining IAC(Terraform-Provision) with CAC(Ansible-Configure)

\* Developed playbook to setup Terraform core, checkov, java

\* Developed Roles/Playbooks to configure Build servers for Java/Node based web applications

\* Developed Roles/Playbooks to configure Jenkins Master and Jenkins Slaves servers

\* Developed Roles to setup self managed Kubernetes Master & Dataplane

===============

DevOps Projects [ Pipeline/Jenkins/DevSecOps ]

===============

\* Developed Jenkinsfile (Pipeline as Code) for Build Pipeline

\* Setup Deployment pipelines for Java/Node microservices to support Release activities cross different environments QA/Stage/Production

\* Setup DevSecops pipeline for Building Platform Base Image

\* Enabled DevSecops pipelines for Infra provisioning for catching the IAC misconfigurations and to handle approvals automatically

\* Enabled DevSecops pipeline for Configurations of different servers and also to setup HA

Setup Build Pipelines for Java/Node microservices to support Release activities cross different branches like Feature/Integration/Release branches

\* Setup DevSecops based CICD pipeline for SCA, SAST, DAST, Vulnerabilities, GitOps

===============

DevOps Projects [ Containerization/Docker ]

===============

\* Containerization of Java/Node microservices - generating OS-Base Image, platform Base Image & Application Image

\* Developed deployment manifest files for Java/Node microservices

\* Enabled Gitops based deployments on Kubernetes

===============

DevOps Projects [ Montioring ]

===============

\* Enabled Continuous Monitoring solution i.e collect, visualize, alerts

\* Enabled Centralized Log management for all applications running in QA & Prod

More Projects on our Youtube Channel:

====================================

- K8s deployment strategy

- K8s debugging

- Golden Images using packer/terrform

- Docker Image optimization

- Managing Dynamic secrets using Hashicorp Vault for Infra provisioning

- Kubernetes secrets using Hashicorp Vault

Day-Day Tasks

=============

\* Standup Meeting (Scrum/Kanban), Release Planning, CAB meeting

\* Customer tickets or Dev/QA tickets or Internal project Jira Tickets/MS Teams/Slack/ServiceNow

\* Monitor all the services, servers, applications on UAT/Prod environments

\* Monitor all the build, deployment, other pipelines are running fine

\* Manage Gitlab groups, Manage gitlab projects, manage branches, approval, merges, webhook

\* Develop Terraform configuration files, modules to support provisioning Dev/QA/stage/prod environments

\* Develop Ansible Playbook/Roles for automating configuration of servers - Build/LAMP

\* Develop Jenkins Pipeline for automating various builds like CI, Nightly, BaseImage, Continious Delivery and IAC pipelines, Configuration Management Pipelines

\* Setup, Configure & Maintain Jenkins Master, Jenkins Slaves, plugin management, backup, permissions

\* Maintain Jenkins jobs for various build pipeline automation & IAC/CM

\* Develop Dockerfile for BaseImage(OS, JDK, JBOSS, JMX exporter) & AppImage (Jar/War, start/stop scripts, healthcheck scripts, monitoring agent)

\* Setup Gitops operator using ArgoCD for Dev/QA/UAT/Prod environments

\* Develop Kubernetes Manifest for Application deployment, log management

\* Develop Helm packages for Microservice deployment, DevOps applications

\* Develop Monitoring solution and Alerting rules for notifiation using Prometheus, Grafana

- configure exporters, configure targets to scarpe frequently, setup rules, setup dashboards

\* Implement DevSecOps at every level of DevOps

\* Collaborate with stakeholders

Roles & Responsibilities

========================

\* Support Continuous Development for projects - Manage Gitlab, namespaces, groups, projects, branching strategies, developer workflow

\* Automate Infrastructure provisioning using IAC - Terraform

\* Automate Configuration Management of different servers like Dev, QA, Build, Stage or Production using Ansible

\* Automate Formal Builds like Continuous Integration, Full/Nightly Build, Release, Integration pipelines

\* Automate Continuous Delivery & Deployment pipelines

\* Containerization of Products - Develop BaseImage, App Image

\* Automated Kubernetes Deployments using Gitops & Helm

\* Support Centralized Log management - EFK

\* Continuous Monitoring of Build server, Deployment server, Kubernetes Cluster, Application, Services - setup a monitoring solution

\* Ensure the security of the DevOps process & pipeline

\* Cloudops - EKS |IAM-RBAC

k8s workstation - kube/config

Resume

======

Technical Summary:

- X years of exp

- problems you solved or concepts you were responsible to solve

Technical Skills:

- expert in setting up CICD pipelines using Jenkins, Jenkinsfile

- proficient in developing terraform scripts

- hands-on developing ansible playbooks, roles

- strong working experince in kubernetes and developing docker images using dockerfile

- experience in setting up monitoring solution using prometheus and grafana

- proficient in managing projects in gitlab

- knowledgable on AWS cloud infra

- proficient in implementing devsecops

- proficient in automating day-day tasks using shell/python scripts

Achievements:

- KPI

Work Experience:

- Company: Roles/Responsibilities

Preparing for Interviews

========================

\* Module - problem statement & the solution

\* Best practise

\* 2-3 projects

\* Architecture / Diagrams

\* prepare syntax

\* Topic wise notess