git,

shell scrpting

linux,

git -> source mgt tool, version controlling, we use github as remote repo or cetral repo

jenkins ->CONTINUOUS iNTREGRATION [CI] & CONTINUOUS DEPLOYMENT [CD] -IT Is a server-based system ,

interation mgt tool, where we actually intergrate the required plugins, like,git, maven, ansible, k8s, docker , and so on .

maven -> it's build tool for the java applications, (npm for reactjs nodejs applications, gradle for andriod applicaitons,)

ansible - IT IS A powerful OPEN-SOURCE AUTOMATION TOOLS USED FOR MANAGEMENT,APPLICATION DEPLOYMENT,AND ORCHESTRATION IN DEVOPS WORK FLOW ARCHITECTURE.configuaration tool, roles, invetory file and modules , and ansible controller an ansible worker nodes( ip adress of target servers,) ,ssh connection, yaml

files,ansible playbook, [CI|CD].WHAT IS ANSIBLE IN CI | CD?

WHAT IS CI | CD USED FOR?

APPLICATION DEPLOYMENT

CONFIGURATION MANAGEMENT

CLOUD PROVISIONING

UPDATING WORKSTATIONS AND SERVERS

SOFTWARE INSTALLATIONS

INFRASTRUCTURE PROVISIONING

SYSTEM CONFIGURATIONS

ORCHESTRATING ADVANCED WORKFLOWS

docker -> containerization platform, where we create the containers and containerized applications , using the doker images ,

to create the docker images, we need docker file,

there are 2 ways to create the docker images

1.using Dockerfile

2.using commit

3.using docker pull command, from docker registry or dockerhub,

docker compose file ->using docker file we can run the multiple containers as single service

three tier architecture

web server /front end sever -> tomcat , nginx ,c1

applcation server/ backend server -> java applcation or nodejs or react js applcation , c2

database server / -> mysql,

docker swarm -> container mgt tool, meant for docker containers,( pod man)

k8s -> containers orchestration tool,docker containers, podman containers,

stages :

stage 1 -> developers going to pull the source code into the remoterepo or central, github,

stage 2 -> pull the code into ur local repo(git)

stage 3 -> converting the source code into artifacts, using the maven build,mvn build , mvan install,mvn package, mvn clean, mvn test, jar or war

stage 4 -> sonar qube -> testing the sourcecode,

stage 5 -> jfrog or nexus as artifact repos, s3 bucket in aws

stage 6 -> docker -> converting artifacts into docker images, coverting the images into docker containers or containerized applications,

stage 7 -> loading the conainterzed applications into th k8s cluster,helm charts

stage 8 -> prometheus and grafan(dashboard) monitoring the contanierized application, whether the application is running as expected

jenkins

devops

ssh-keygen

forking repo

README file

git api

PAT creation

Braching stragies

linux - opensource operating system, where you can update and upgrade application for free

system is depending on operating system, unix, linux,ios, window

operating system depends on kernel( linux)(it's like a software )

linux os contains shell and kernal

shell - command line interpreter (cli)

linux does not required GUI ,

linux is faster than windows,where you use the cli or commands, cd

maven -> build tool for java applications

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

pom.xml(project object model) -> mvn install -> compile and create or build the artifacts in pom.xml

it contains all the dependencies or plugins which used to build the artifacts, war , jar,

sde -eclipse or vs code or

maven project

target-> war or jar ---> web server , tomcat or nginx

pom.xml--->developers will develop this file ,

<project>

<groupID>com.dbsbank</groupID> --> organization name

<artifactID>maven-java-project</artifactID> ---> project name

<version>1.0.0</version>

<packaging>jar</packaging>

<dependencies>

<dependency>

<groupID>Junit</groupID>

<artifactID>Junit</artifactID>

<version>3.8.1</version>

<scope>test</scope>

</dependency>

<dependencies>

<groupId>org.kie.kogito</groupId>

<artifactId>kogito-springboot-starter</artifactId>

<version>1.22.1.Final</version>

</dependency>

</dependencies>

</project>

scope of dependencies:

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

test

compile

run

provided

Testing Framework:

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

JUnit--->Java

NUnit--->.Net

PyTest--->Python

NodeJs/Angular---> Jasmin

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

Installation of mvn ( build server )

#Execute in Slave Nodes:

#Install epel Package:

amazon-linux-extras install epel -y

#Install Java:

amazon-linux-extras install java-openjdk11 -y

#Install GIT:

yum install git -y

#Install Maven:

sudo wget https://dlcdn.apache.org/maven/maven-3/3.9.4/binaries/apache-maven-3.9.4-bin.tar.gz

sudo tar xf apache-maven-3.9.4-bin.tar.gz -C /opt

#Create a symlink for Apache maven directory to update the maven versions

sudo ln -s /opt/apache-maven-3.9.4 /opt/maven

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

#Set Java Path / Environment Variables:

## Check the java installed version:

#cd /usr/lib/jvm/

#eg: /usr/lib/jvm/java-11-openjdk-11.0.13.0.8-1.amzn2.0.3.x86\_64

#open .bash\_profile & add the following lines:

#java-11-openjdk-11.0.16.0.8-1.amzn2.0.1.x86\_64

#java-11-openjdk-11.0.19.0.7-1.amzn2.0.1.x86\_64

export JAVA\_HOME="/usr/lib/jvm/java-11-openjdk-11.0.20.0.8-1.amzn2.0.1.x86\_64"

export MAVEN\_HOME=/opt/apache-maven-3.9.6

export M2=/opt/apache-maven-3.9.6/bin

PATH=$PATH:$HOME/bin:$JAVA\_HOME/bin:$MAVEN\_HOME:$M2

source ~/.bash\_profile

source ~/.bashrc

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

#Add User :

useradd -m -d /home/devopsadmin devopsadmin

su - devopsadmin

ssh-keygen

ls ~/.ssh

#You should see following two files:

#id\_rsa - private key

#id\_rsa.pub - public

#cat id\_rsa & copy the private key and paste it into jenkins node config. enter private key directly field. Then,

cat id\_rsa.pub > authorized\_keys

chown -R devopsadmin /home/devopsadmin/.ssh

chmod 600 /home/devopsadmin/.ssh/authorized\_keys

chmod 700 /home/devopsadmin/.ssh

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

ssh client -> mobexerm , putty, super putty, gitbash, default terminals in linux os or macos

to connect your local server to remote server

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

tomcat -java is prerequisites

nginx -> c ,not required

devops

developer java, python,

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

aug7

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

proxy vs (nginx)reverse proxy

nginx acts as forwardproxy and reverse proxy

to run any web based applications , you must server that's web server -> nginx

amazon web page- to order a book

client/user server - internet - webserver of the amazon

client/user server - internet - nginx proxy server - web server of the amazon

proxy -> doing something on behalf of someone

proxy server -> it sits between client server and internet

reverse proxy -> it sits between and internet and web servers

hacking

client/user server - nginx proxy server - internet - web server of the amazon

clint 1.1.1.1 \_> proxy -> internet ->

web based applications

static

dynamic

a= rama

echo $a

types of linux distributions

fedora - centos redhat, amz linux -> yum package manager- execute the commands

useradd -> to create a user or to add the user

debian- ubuntu -> apt or apt-get as a package manager

adduser -> to create a user or to add the user

alpine apk

almalinux yum

tomcat-> 8080

jenkins-> 8080

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

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

three tier architecture or typical web application architecture or multi tier architecture

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

0 - 65535

Front-end/web server - User Interface -HTML, CSS, images, and client-side scripts like JavaScript -nginx 80

Back-end/application server/Application - Business Logic -PHP, Java, Python, Ruby,nodejs , reactjs, and etc tomcat 8080

db server - Database -MySQL, PostgreSQL, Oracle, MongoDB

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

What is Web server?

A Web server is a program that uses HTTP or HTTPS (Hypertext Transfer Protocol) protocol to serves web

content (HTML and static content) to users.

Examples

Apache HTTP server (80) HTTPS (443)

Nginx (pronounced engine X) (80)

IBM HTTP server (IHS)

Oracle iplanet web server

Internet Information Server (IIS)

What is Application Server?

An application server is a container upon which you can build and expose business logic and processes to client applications through various protocols including HTTP in a n-tier architecture

(Presentation Layer (Tier):UI, Application Layer (Tier):business logics, Data Layer (Tier): database.

Examples

Apache Tomcat (8080)

JBoss/WildFly - RedHat

WebLogic - Oracle

WebSphere Application Server - IBM

WebSphere Liberty Profile - IBM

Galssfish

What is database Server:

A server is responsible for storing and managing data for the web application. It stores structured data, such as user information, product details, or any other data necessary for the application to function. When the application needs to retrieve or store data, it communicates with the database server using queries.

Examples:

MySQL( 3306), PostgreSQL(5432), Oracle(1521), MongoDB(27017),Elasticsearch(9200)

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

Apache HTTP Server (HTTPD):80 and

Apache HTTP Server, commonly known as Apache, is one of the most popular and widely used web servers in the world

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

Tomcat Server!!!!8080

Tomcat or Apache Tomcat is a light weight, open source web container used to deploy and running the java based web applications,

developed by Apache Software Foundation (ASF).

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

#Install & configure Tomcat server :

#Check out Official WebPage:

#https://tomcat.apache.org/

#Launch AWS EC2 Linux Instance: 8080

sudo -i

yum update -y

#Install JDK

#Install epel Package:

amazon-linux-extras install epel -y

#Install Java:

amazon-linux-extras install java-openjdk11 -y

#Set Java Path / Environment Variables:

#open .bashrc & add the following lines:

#OpenJDK Runtime Environment (Red\_Hat-11.0.20.0.8-1.amzn2.0.1) (build 11.0.20+8-LTS)

OpenJDK 64-Bit Server VM (Red\_Hat-11.0.20.0.8-1.amzn2.0.1) (build 11.0.20+8-LTS, mixed mode, sharing)

export JAVA\_HOME="/usr/lib/jvm/java-11-openjdk-11.0.20.0.8-1.amzn2.0.1.x86\_64"

PATH=$PATH:$HOME/bin:$JAVA\_HOME/bin

#Save the file

#open .bash\_profile & add the following lines:

export JAVA\_HOME="/usr/lib/jvm/java-11-openjdk-11.0.20.0.8-1.amzn2.0.1.x86\_64"

PATH=$PATH:$HOME/bin:$JAVA\_HOME/bin

OpenJDK Runtime Environment (Red\_Hat-11.0.20.0.8-1.amzn2.0.1) (build 11.0.20+8-LTS)

OpenJDK 64-Bit Server VM (Red\_Hat-11.0.20.0.8-1.amzn2.0.1) (build 11.0.20+8-LTS, mixed mode, sharing)

#Save the file

source ~/.bashrc

source ~/.bash\_profile

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#Install tomcat in Amazon Linux Instance:

#https://dlcdn.apache.org/tomcat/tomcat-8/v8.5.85/bin/apache-tomcat-8.5.85.tar.gz

https://dlcdn.apache.org/tomcat/tomcat-10/v10.1.11/bin/apache-tomcat-10.1.11.tar.gz

cd /opt

sudo wget https://dlcdn.apache.org/tomcat/tomcat-10/v10.1.11/bin/apache-tomcat-10.1.11.tar.gz

tar -xvzf /opt/apache-tomcat-10.1.11.tar.gz

mv apache-tomcat-10.1.11 tomcat

#Start Tomcat Server:

#Goto:

cd /opt/tomcat/bin

./startup.sh

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#Add-USer for Tomcat :

useradd -m -d /home/devopsadmin devopsadmin

su - devopsadmin

ssh-keygen

ls ~/.ssh

#You should see following two files:

#id\_rsa - private key

#id\_rsa.pub - public key

cd /home/devopsadmin/.ssh

cat id\_rsa.pub > authorized\_keys

chown -R devopsadmin /home/devopsadmin/.ssh

chmod 600 /home/devopsadmin/.ssh/authorized\_keys

chmod 700 /home/devopsadmin/.ssh

#make devopsadmin user as a owner to tomcat dir :

chown -R devopsadmin /opt/tomcat

/home/devopsadmin

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

How to change the port number in Tomcat?

Go to the conf directory and open the server.xml and you will find below lines.

cd ..

cd conf/

vim server.xml

check the server.xml port no is 8080 and save

<Connector port="8080" protocol="HTTP/1.1"

connectionTimeout="20000"

redirectPort="8443" />

Replace the 8080 with any port number.

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

As a DevOps engineer, understanding the directory structure in Linux is essential for managing and deploying applications effectively

/bin - Essential user binaries

Shell interpreters (e.g., Bash),services,

Basic utilities like cp, mv, rm, etc.

/etc - Configuration files

Configuration management tools: Ansible, Puppet, Chef, SaltStack

Network configuration tools: ifconfig, ip, iptables

/opt - Optional or third-party softwares

Custom software installations, including various DevOps tools,tomcat or sonarqube,nexus war files

Specific versions of programming languages and runtimes

/var - Variable data

Log management tools: Elasticsearch, Logstash, Kibana, Fluentd, Splunk, jenkins-workspace

Monitoring tools: Prometheus, Grafana, Nagios, Zabbix

/usr - User-related data

Development tools: GCC (GNU Compiler Collection), make, cmake

Source control tools: Git, SVN (Subversion)

Package managers: apt, yum, pip (Python), npm (Node.js)

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

sonarqube -9000

static code analyser tool

code quality mgt tool

code quality ?

once you built the artifact using mvn , you have to check the quality of the source code,

code review -> manually ->

------

tomcat -> to change the port number -> server.xml

sonarqube -> to change the port number -> sonar properties file

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

nexus -8081

nexus is opensource , java based, artifactory repository manager

build artifacts or packages

it's used to store the build artifacts and retrieve the build artifacts whenever required

2 editions

Nexus oss > open source software

nexus pro -> enterprise edition

usually we store

java -> jar , war, ear

docker -> docker images

node js -> .tar

conf

etc/

nexus-default.properties

prerequisites

java 1.8 or java 8 or java 11

nexus version : nexus 3.x

2 gb ram required to run the nexus server

instance type - t2 medium

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

nexus installation

tomcat- apache

sonarqube -apache

sonatype nexus repository mgt

source : https://help.sonatype.com/repomanager3/product-information/download

free -h

java -version

An EC2 instance with a minimum of 2 GB RAM (t2.medium)

sudo yum update -y

amazon-linux-extras list

# Install Java 8 (OpenJDK 8)

sudo yum install java-1.8.0-openjdk-devel -y

This will install OpenJDK 8 and the development tools.

#Set Environmental Variables

Now, you need to set the JAVA\_HOME and update the PATH in both the .bashrc and .bash\_profile files.

#Set Environmental Variables in .bashrc and .bash\_profile

# Open .bashrc for editing

vim ~/.bashrc

# Add the following lines at the end of the file

export JAVA\_HOME="/usr/lib/jvm/jre-1.8.0-openjdk"

export PATH="$JAVA\_HOME/bin:$PATH"

Save and exit the text editor.

source ~/.bashrc

# Open .bash\_profile for editing

vim ~/.bash\_profile

# Add the same lines as in .bash\_profile

export JAVA\_HOME="/usr/lib/jvm/jre-1.8.0-openjdk"

export PATH="$JAVA\_HOME/bin:$PATH"

Save and exit the text editor.

source ~/.bash\_profile

#check the version

java -version or java --version

sudo -i

cd opt

# https://download.sonatype.com/nexus/3/nexus-3.59.0-01-unix.tar.gz

wget https://download.sonatype.com/nexus/3/nexus-3.59.0-01-unix.tar.gz

tar -zxvf nexus-3.59.0-01-unix.tar.gz

mv /opt/nexus-3.59.0-01 /opt/nexus

as a good security practice , nexus is not advised to run nexus service as a root user, so create a new user called nexusadmin and grant sudo access to mange nexus services as follow

useradd nexusadmin

give the sudo access to nexusadmin

visudo

nexusadmin ALL=(ALL) NOPASSWORD:ALL

chown -R nexusadmin:nexusadmin /opt/nexus

chown -R nexusadmin:nexusadmin /opt/sonatype-work

chown -R 775 /opt/nexus

chown -R 775 /opt/sonatype-work

open /opt/nexus/bin/nexus.rc file and uncomment run as user parameter and set as nexusadmin user

vim /opt/nexus/bin/nexus.rc

run\_as\_user="nexusadmin"

create nexus as a service

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

switch as nexusadmin user to start the nexus service

su - nexusadmin

#open the file as root user

vim /etc/systemd/system/nexus.service

------

[Unit]

Description=Nexus Repository Manager

After=network.target

[Service]

Type=forking

LimitNOFILE=65536

User=nexusadmin

ExecStart=/opt/nexus/bin/nexus start

ExecStop=/opt/nexus/bin/nexus stop

Restart=on-abort

[Install]

WantedBy=multi-user.target

------

sudo systemctl daemon-reload

sudo systemctl start nexus

sudo systemctl enable nexus

sudo systemctl status nexus

access the server using publicip:8081

3.15 v

admin

admin123

your admin user password is located in

username : admin

password :

/opt/sonatype-work/nexus3/admin.password on the server

cat /opt/sonatype-work/nexus3/admin.password

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

nexus artifact repo

type of version policies

release and snapshot

whenever the developers deploy the containerized the applications into the productions environment(k8s cluster),live environment

all the production artifacts or versions store in the release repo , v1 ,

snapshot -> it's used to store the ongoing versions or developement patches , v1--> v1.1 , v1.2

what is the release retention ?

mvn clean install deploy

you have the define the release repo and snapshot repo in pom.xml

mvn central repo

/opt/maven/conf/setting.xml

have to modify the nexus server name, username and password in server tag and modify the remote repo in mirror tag(change only the ip address of maven central) in setting.xml file

<settings xmlns="http://maven.apache.org/SETTINGS/1.2.0"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.2.0 https://maven.apache.org/xsd/settings-1.2.0.xsd">

<!-- ... (other settings) ... -->

<!-- servers

Define your server credentials here.

-->

<servers>

<server>

<id>nexus</id>

<username>admin</username>

<password>563856</password>

</server>

</servers>

<!-- ... (other settings) ... -->

<!-- Define mirrors here -->

<mirrors>

<mirror>

<id>maven-default-http-blocker</id>

<mirrorOf>external:http:\*</mirrorOf>

<name>Pseudo repository to mirror external repositories initially using HTTP.</name>

<url>http://52.21.206.31:8081/repository/maven-central/</url>

</mirror>

</mirrors>

</settings>

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

jenkins-> 8080

jenkins - integration mgt tool, which is used to automate your end to end cicd pipeline,

ci - continuous integration ( hudson)

cd - continuous deploy and delivery

master/slave architecture

jenkins master (master node) -> it instructs to jenkins slave to execute the the tasks,projects,items,jobs

jenkins slave (agent or slave node)-> build tool(maven) , slave excutes the tasks which assigned by the jenkins master

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

jenkins installation

#Jenkins Master Slave Configuration:

#Launch New AWS Linux 2 Instances

#Jenkins Master, Node1.

#Launch AWS Linux Instance with port 8080 & Name Tag

#Update the Instance

sudo -i

yum update -y

#Install Jenkins :

source : https://pkg.jenkins.io/redhat-stable/

#to add the jenkins repo in your local machine

sudo wget -O /etc/yum.repos.d/jenkins.repo https://pkg.jenkins.io/redhat-stable/jenkins.repo

sudo rpm --import https://pkg.jenkins.io/redhat-stable/jenkins.io-2023.key

#Install epel Package:

amazon-linux-extras install epel -y

#Install Java:

amazon-linux-extras install java-openjdk11 -y

export JAVA\_HOME="/usr/lib/jvm/java-11-openjdk-11.0.20.0.8-1.amzn2.0.1.x86\_64"

export PATH="$JAVA\_HOME/bin:$PATH"

source ~/.bash\_profile or source ~/.profile

source ~/.bashrc

yum install jenkins

#Start Jenkins:

systemctl start jenkins

# Setup Jenkins to start at boot,

systemctl enable jenkins

# to check the status of jenkins

systemctl status jenkins

# access the jenkins server over browser using public ip of the instance followed by the default port number 8080

publicip:8080

cat /var/lib/jenkins/secrets/initialAdminPassword

admin password :

------

manage jenkins

System Configuration:

system cofigure

plugins -> to install multiple software plugins, which used to create the end to end cicd pipeline

global tools config -> configuring the plugins to jenkins server

Nodes -> to create and configure the slave server

security :

credentials ->

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

The main difference between Jenkins Freestyle projects and Pipeline is the usage of GUI vs scripting.

Free style projects for testing the jobs before implementing into pipeline

basic purpose

Below are some differences in more detail

Freestyle Projects

Use GUI to add different stages and steps

More suitable for less complex scenarios

Good for people who are starting to use Jenkins/ CI solutions

Can become hard to achieve what you want when your scenario become more complex( if you have multiple stages in your projects )

Pipeline Jobs

Use code (Groovy language - this is a Java like language) for giving instructions

Since, everything in one script you can keep that in the source control and have ability to revert back to an earlier version at any time or keep track of changes made to the script

Entire,

pipeline consists of steps (ex: clone stage,build, test, deploy etc..)

Created using a Jenkinsfile

Jenkinsfile can be one of two of below types

Declarative Pipeline -> the script start with pipeline -> most widely used scripting language in to build end to end cicd pipeline

Scripted Pipeline -> the script start with node -> this is traditional and complex pipeline

clone project git url < name github project repo >

build the code --> mvn clean package, mvn build ,mvn install, mvn test

test the code

post build step :

deploy the code---> deploy orde depoy container (target servers) -> deploy container plugin

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

jenkins master ---- jenkins slave1 --- build tools java web based application -- maven

jenkins slave2 --- build tools nodejs,reactjs,nextjs web based application -- npm

jenkins slave3 --- build tools andriod application -- gradle

jenkins slave4 --- build tools python web based applications -- pip/python ..

how to integrate the jenkins slave(node) or build server to jenkins master

in slave machine

#Add User :

useradd -m -d /home/devopsadmin devopsadmin

su - devopsadmin

ssh-keygen

ls ~/.ssh

#You should see following two files:

#id\_rsa - private key

#id\_rsa.pub - public

#cat id\_rsa & copy the private key and paste it into jenkins node config. enter private key directly field. Then,

cat id\_rsa.pub > authorized\_keys

chown -R devopsadmin /home/devopsadmin/.ssh

chmod 600 /home/devopsadmin/.ssh/authorized\_keys

chmod 700 /home/devopsadmin/.ssh

----

GIT PARAMETER - TO SELECT THE BRANCH

BUILD EXECUTERS 3

BUILD QUEUE

CLONING STAGE

BUILDING STAGE

TESTING STAGE

POST BUIDING - DEPLOY STAGE

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

scm -checkout - cloning the project

pipeline {

agent any

tools {

// Install the Maven version configured as "M2" and add it to the path.

maven "maven-3.9.4"

}

stages {

stage('cloning the project') {

steps {

// Get some code from a GitHub repository

git url: 'https://github.com/manju65char/insurance-web-application.git'

}

}

stage('Maven Build') {

steps {

// Run Maven on a Unix agent.

sh "mvn -Dmaven.test.failure.ignore=true clean package"

}

}

stage('Maven test') {

steps {

// Run Maven on a Unix agent.

sh "mvn test"

}

}

stage('deploy to tomcat server') {

steps {

script {

ssh publisher : snippet generator

}

}

}

}

}

---

deploy to qa environment or webserver(nginx) or application server(tomcat)

deploying your source code into tomcat -> publish over ssh --> the plugin which we use to deploy your code from source server to target server .

deploying from build server to tomcat to serve the application

source server -> build server -> target/<artifactname>.war or jar || /home/devopsadmin/workspace/firstpipelineproject

target server(remote server) -> qa server -> /opt/tomcat/webapps

jenkins master is intermediary between source server and target server

use snippet generator to get pipeline code to add in deploy stage

-> whenever you get any unstable error -> make sure with the target server is up and running and check with user credentials

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

integration of sonar and nexus to jenkins

create sonar admin token

sonar-token - 68ad9e25dfd5621c15551d9c84963ae91b9cac2e

username: admin

password: 563856

sonar scanner for jenkin- sonarqube integration -> quality gate, code smell,

jacoco plugin - java code coverage - it's used to get the code coverage of your project ( source code)

in nexus repo

create a repo

plugins required

sonatype nexus

display name : nexus-server

server id : nexus server

server url : publicip:8081

credential : nexus user name and password

username: admin

password: 563856

nexus platform plugin - nexus platform plugin

---

mvn clean pkg or mvn clean install -> remove the existing artifacts in your pom.xml and target (jar or war file) ,

pipeline {

agent any

tools {

// Install the Maven version configured as "M2" and add it to the path.

maven "maven-3.9.4"

}

stages {

stage('Cloning the project') {

steps {

// Get some code from a GitHub repository

git url: 'https://github.com/manju65char/insurance-web-application.git'

}

}

stage('Maven Build') {

steps {

// Run Maven on a Unix agent.

sh "mvn -Dmaven.test.failure.ignore=true clean package"

}

}

stage('Test & Jacoco Static Analysis') {

steps {

script {

junit 'target/surefire-reports/\*\*/\*.xml'

jacoco()

}

}

}

stage('Sonar Scanner Coverage') {

steps {

script {

sh """export MAVEN\_OPTS="-Xmx1024m"

mvn sonar:sonar -Dsonar.login=68ad9e25dfd5621c15551d9c84963ae91b9cac2e -Dsonar.host.url=http://35.169.20.187:9000"""

}

}

}

stage('Publish to Nexus') {

steps {

nexusPublisher nexusInstanceId: 'nexus-server',

nexusRepositoryId: 'amazon-releases',

packages: [

[$class: 'MavenPackage',

mavenAssetList: [

[classifier: '', extension: '', filePath: './target/insure-me-1.0.jar']

],

mavenCoordinate: [

artifactId: 'insure-me',

groupId: 'com.project.manjuDOE',

packaging: 'jar',

version: '1.0'

]

]

]

}

}

stage('deploy into tomcat') {

steps {

script {

ssh publisher : snippet generator

}

}

}

}

}

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

as devops engineer , need to hide the project related information and credentials in the pipeline

shared library

in jenkins

create jenkins credentials for shared libraries to call the fuctions from share library repository through jenkins pipepline to execute the code

to hide the code where you use the varaibles , and functions ,

in github

share library -> to store the variables , you are going to create the functions and store the variable (codes) in that function

you are going the call the function through pipeline

github or gitlab ---> create a repo in github, then clone the repo into your local machine , then ,create var dir and create fucntions with groovy ,

crate a repo -> shared-libraries

(folder) vars

.groovy files ( creating functions of the stages )

how to integrate shared library to jenkins ,

pipeline {

agent any

tools {

// Install the Maven version configured as "M2" and add it to the path.

maven "maven-3.9.4"

}

environment {

REPOSITORY\_NAME = "insurance-web-application"

REPOSITORY\_URL = "https://github.com/manju65char/${env.REPOSITORY\_NAME}.git"

}

stages {

stage('Cloning the Project') {

steps {

catchError(buildResult: 'FAILURE', stageResult: 'FAILURE') {

script {

withCredentials([

string(credentialsId: 'github-credentials', variable: 'github-credentials')

]) {

library identifier: 'jsl@master', retriever: modernSCM([

$class: 'GitSCMSource',

remote: "https://github.com/manju65char/shared-libraries.git",

credentialsId: 'github-credentials'

])

}

cloneProjectStage(REPOSITORY\_URL, params.BRANCH\_NAME)

}

}

}

}

stage('Maven Build') {

steps {

script {

try {

buildCodeStage()

} catch (Exception e) {

currentBuild.result = 'FAILURE'

error('Failed to build code')

}

}

}

}

stage('Test & Jacoco Static Analysis') {

steps {

script {

junit 'target/surefire-reports/\*\*/\*.xml'

jacoco()

}

}

}

stage('Sonar Scanner Coverage') {

steps {

script {

sh """export MAVEN\_OPTS="-Xmx1024m"

mvn sonar:sonar -Dsonar.login=68ad9e25dfd5621c15551d9c84963ae91b9cac2e -Dsonar.host.url=http://35.169.20.187:9000"""

}

}

}

stage('Publish to Nexus') {

steps {

nexusPublisher nexusInstanceId: 'nexus-server',

nexusRepositoryId: 'amazon-releases',

packages: [

[$class: 'MavenPackage',

mavenAssetList: [

[classifier: '', extension: '', filePath: './target/insure-me-1.0.jar']

],

mavenCoordinate: [

artifactId: 'insure-me',

groupId: 'com.project.manjuDOE',

packaging: 'jar',

version: '1.0'

]

]

]

}

}

}

}

stage('deploy into tomcat') {

steps {

script {

ssh publisher : snippet generator

}

}

}

}

}

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

def cloneProjectStage(repositoryUrl, branchName) {

git branch: branchName, credentialsId: 'latest-github-credentials', url: repositoryUrl

}

pipeline {

agent any

parameters {

choice(

choices: ['master', 'develop', 'feature/branch-1', 'feature/branch-2'],

description: 'Select the branch to build',

name: 'BRANCH\_NAME'

)

}

environment {

REPOSITORY\_NAME = "insurance-web-application"

REPOSITORY\_URL = "https://github.com/manju65char/${env.REPOSITORY\_NAME}.git"

}

stages {

stage('Cloning the Project') {

steps {

catchError(buildResult: 'FAILURE', stageResult: 'FAILURE') {

script {

withCredentials([

string(credentialsId: 'latest-github-credentials', variable: 'latest-github-credentials')

]) {

library identifier: 'jsl@master', retriever: modernSCM([

$class: 'GitSCMSource',

remote: "https://github.com/manju65char/shared-libraries.git",

credentialsId: 'latest-github-credentials'

])

}

cloneProjectStage(REPOSITORY\_URL, params.BRANCH\_NAME)

}

}

}

}

stage('buildCodeStage') {

steps {

script {

try {

buildCodeStage()

} catch (Exception e) {

currentBuild.result = 'FAILURE'

error('Failed to build Docker container')

}

}

}

}

stage('mvn test') {

steps {

script {

try {

testCodeStage()

} catch (Exception e) {

currentBuild.result = 'FAILURE'

error('Failed to build Docker container')

}

}

}

}

stage('pubilish to nexus') {

steps {

script {

try {

artifactStage()

} catch (Exception e) {

currentBuild.result = 'FAILURE'

error('Failed to build Docker container')

}

}

}

}

stage('dockerbuild') {

steps {

script {

try {

dockerBuildStage()

} catch (Exception e) {

currentBuild.result = 'FAILURE'

error('Failed to build Docker container')

}

}

}

}

stage('Deploy to k8s') {

steps {

script {

try {

deploymentStage()

} catch (Exception e) {

currentBuild.result = 'FAILURE'

error('Failed to build Docker container')

}

}

}

}

}

}

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

buildCodeStage.groovy

def call() {

sh 'echo "maven code build and created artifacts"'

}

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

testCodeStage.groovy

def call() {

sh 'echo "test the code"'

}

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

artifactStage.groovy

def call() {

sh 'echo "deploying the code into nexus server"'

}

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

dockerBuildStage.groovy

def call() {

sh 'echo "building the image "'

}

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

deploymentStage.groovy

def call() {

sh 'echo "deployed the containerized app into k8s cluster"'

}

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

slack

collaboration tool

slackwithjenkins

integration of slack with jenkins

collabarative tool

requirements

slack account

create a channel in slack

install jenkins server -- and install slack notification plugin

source : https://slack.com/intl/en-in/

workspace name : ManjuSoftwareSolutions

team working on : Devops Methodology

add coworkers or Who else is on the ManjuSoftwareSolutions team? or skip

What’s your team working on right now? devops

channel name ; # build notifications

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

def cloneProjectStage(repositoryUrl, branchName) {

git branch: branchName, credentialsId: 'latest-github-credentials', url: repositoryUrl

}

pipeline {

agent any

parameters {

choice(

choices: ['master', 'develop', 'feature/branch-1', 'feature/branch-2'],

description: 'Select the branch to build',

name: 'BRANCH\_NAME'

)

}

environment {

REPOSITORY\_NAME = "insurance-web-application"

REPOSITORY\_URL = "https://github.com/manju65char/${env.REPOSITORY\_NAME}.git"

}

stages {

stage('Cloning the Project') {

steps {

catchError(buildResult: 'FAILURE', stageResult: 'FAILURE') {

script {

withCredentials([

string(credentialsId: 'latest-github-credentials', variable: 'latest-github-credentials')

]) {

library identifier: 'jsl@master', retriever: modernSCM([

$class: 'GitSCMSource',

remote: "https://github.com/manju65char/shared-libraries.git",

credentialsId: 'latest-github-credentials'

])

}

cloneProjectStage(REPOSITORY\_URL, params.BRANCH\_NAME)

}

}

}

}

stage('buildCodeStage') {

steps {

script {

try {

buildCodeStage()

} catch (Exception e) {

currentBuild.result = 'FAILURE'

error('Failed to build Docker container')

}

}

}

}

stage('mvn test') {

steps {

script {

try {

testCodeStage()

} catch (Exception e) {

currentBuild.result = 'FAILURE'

error('Failed to build Docker container')

}

}

}

}

stage('pubilish to nexus') {

steps {

script {

try {

artifactStage()

} catch (Exception e) {

currentBuild.result = 'FAILURE'

error('Failed to build Docker container')

}

}

}

}

stage('dockerbuild') {

steps {

script {

try {

dockerBuildStage()

} catch (Exception e) {

currentBuild.result = 'FAILURE'

error('Failed to build Docker container')

}

}

}

}

stage('Deploy to k8s') {

steps {

script {

try {

deploymentStage()

} catch (Exception e) {

currentBuild.result = 'FAILURE'

error('Failed to build Docker container')

}

}

}

}

stage('Send Slack Notification') {

steps {

script {

slackSend (

color: currentBuild.resultIsBetterOrEqualTo('SUCCESS') ? 'good' : 'danger',

message: "Pipeline for ${env.JOB\_NAME} has completed (${currentBuild.result}).",

channel: '#build-notifications', // Replace with your Slack channel name

teamDomain: 'manjusoftware-v6b3769', // Replace with your Slack team name

tokenCredentialId: 'slack-credential-id' // Create a secret text credential for your Slack token

)

}

}

}

}

}

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

def cloneProjectStage(repositoryUrl, branchName) {

git branch: branchName, credentialsId: 'latest-github-credentials', url: repositoryUrl

}

pipeline {

agent any

parameters {

choice(

choices: ['master', 'develop', 'feature/branch-1', 'feature/branch-2'],

description: 'Select the branch to build',

name: 'BRANCH\_NAME'

)

}

environment {

REPOSITORY\_NAME = "insurance-web-application"

REPOSITORY\_URL = "https://github.com/manju65char/${env.REPOSITORY\_NAME}.git"

}

stages {

stage('Cloning the Project') {

steps {

catchError(buildResult: 'FAILURE', stageResult: 'FAILURE') {

script {

withCredentials([

string(credentialsId: 'latest-github-credentials', variable: 'latest-github-credentials')

]) {

library identifier: 'jsl@master', retriever: modernSCM([

$class: 'GitSCMSource',

remote: "https://github.com/manju65char/shared-libraries.git",

credentialsId: 'latest-github-credentials'

])

}

cloneProjectStage(REPOSITORY\_URL, params.BRANCH\_NAME)

}

}

slackSend (

color: currentBuild.resultIsBetterOrEqualTo('SUCCESS') ? 'good' : 'danger',

message: "Stage 'Cloning the Project' has completed (${currentBuild.result}).",

channel: '#build-notifications', // Replace with your Slack channel name

teamDomain: 'manjusoftware-v6b3769', // Replace with your Slack team name

tokenCredentialId: 'slack-credential-id' // Create a secret text credential for your Slack token

)

}

}

stage('buildCodeStage') {

steps {

script {

try {

buildCodeStage()

} catch (Exception e) {

currentBuild.result = 'FAILURE'

error('Failed to build Docker container')

}

slackSend (

color: currentBuild.resultIsBetterOrEqualTo('SUCCESS') ? 'good' : 'danger',

message: "Stage 'buildCodeStage' has completed (${currentBuild.result}).",

channel: '#build-notifications', // Replace with your Slack channel name

teamDomain: 'manjusoftware-v6b3769', // Replace with your Slack team name

tokenCredentialId: 'slack-credential-id' // Create a secret text credential for your Slack token

)

}

}

}

// Repeat the same pattern for other stages...

stage('Send Slack Notification') {

steps {

script {

slackSend (

color: currentBuild.resultIsBetterOrEqualTo('SUCCESS') ? 'good' : 'danger',

message: "Pipeline for ${env.JOB\_NAME} has completed (${currentBuild.result}).",

channel: '#build-notifications', // Replace with your Slack channel name

teamDomain: 'manjusoftware-v6b3769', // Replace with your Slack team name

tokenCredentialId: 'slack-credential-id' // Create a secret text credential for your Slack token

)

}

}

}

}

}

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

DOCKER

it's containerization platform

before Docker

single server model :

app1 app2

server 8gb server 8gb

vm model :

virtual machine model

virtual box

vm ware

docker model

artifacts --> docker image --> docker container -- pushing the docker images into the docker hub

docker commit

dockerfile very imp dockerfile components - dockerfile commands or docker layers or docker file instructions

docker pull

------

DOCKER

it's an open source containerized platform .

it's containerization platform , where you are going to create docker images ,container , run the container delete containers

it's advanced than virtualization

dockerswarm for docker containers 1000 -- k8s pod --it smallest unit of k8s --it's collections of containers

container -> it's like a vm and it does not have any OS, it's a lightweight os level virtualization

virtualization -> process that allow for more efficient utilization of physical computer hardware components , cloud computing

containerization --> it's a process packing the applications and its dependencies .

-===================================================

project

team members ---> team leaders ----> team manager

docker client --> it's intial way that many docker user interact with docker daemon or docker engine

docker host --> docker host is the machine where you are installed the docker images

docker daemon --> main component of docker architecture , it runs on the host os, it 's responsible for creating , running,deleting the containers and images , to manage the containers and images

dockerhub ( docker registry) -> it's like remote repo , where we stored all the images ,

build server --> maven --> mvn clean package ---> jar or war --> docker --> docker images -> docker file or pull image from docker hub

sudo yum install docker -y

systemctl start docker --> to start docker service

systemctl enable docker --> to start the server at boot

systemctl status docker

ways to create docker images

docker pull

docker dockerfile

docker pull <image name> --> to pull the images from docker hub

docker images --. to list the docker images

docker build -t <image name >:tag --> to build or create docker image

docker run <image id or image name > --> to convert the image into container ( docker build + docker run)

docker run -d --name <customized continer name > <image name> -->

docker run -d -p hostport:conainerport --name <customized container name> <image name>

docker ps --> to get the running or active containers

docker ps -a --> to get the active and exited containers

port mapping or binding or attaching

attaching your hostport to container port,( -p hostport:containerport) if you want expose your containerised application over browser , where you supposed to containerized port number or externalip:hostport

docker start <container id>

docker stop < container id>

docker rename <existing container name> < new container name> --> to change the container name

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

docker file : it's a text file , which containers a set of instructions, which are used to build the application along with its dependencies .

it's a text file , which containers a set of instructions that automate the docker image creation

dockerfile is used to build the docker image

docker layers or docker instructions or dockerfile components

D --> is the letter while creating the Dockerfile --> FROM, COPY, ADD, ARG, ENTRYPOINT, CMD,WORKDIR,RUN,

start components also be the capital like FROM, COPY, ADD, ARG, ENTRYPOINT, CMD,WORKDIR,RUN,

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

to containerise the entire application --> webserver , applcation , database -- os

FROM -- is used to define the base image ,this component must be on top of the file

ex : FROM ubuntu,alpine -->

RUN --> it defines the executable commands , installing , updating deleting softwares , ls ps pwd echo mkdir ,

what are the commands you want to executtte in the container, where you are going to use RUN component with executable commands

RUN echo"this is my image" >> file1.txt

COPY --> This component is used to copy the files(jar or war) from local machine to container, you have to mention the source and destination point

source -- local machine

destination ---> container

COPY target/\*.war target

ADD -- >

ADD is same as COPY ,it's useful copy the files from your local machine to container , it has 2 two more extra capabailities

ADD can download the file directly from the internet to the container

ADD can untar/unzip the file directly in the container

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

EXPOSE : it's used to expose the default port of container

EXPOSE 80

it's used to expose the ports such as 8080, 80,

ex; if you want to run jenkins in the containers, default port for jenkins is 8080, so you need to specify the jenkins public ip and port in EXPOSE component

EXPOSE instruction is useful to tell the users of the image about the ports and protocols image/container is opening. It will not have any functionality it used only to tell the information.

FROM almalinux

EXPOSE 8080/

WORKDIR /home/manjunathachar/dockerdemo/tem

Set the working directory in the container

like, you want to get into specific dir by default inside the container , in that case, you can specify the dir name , so that you can directly get into it.

ex: if you want to get into tem dir inside the container,

so you can specify the tem in workdir component so that as soon as i open the container , by default you would be there in tem directory

ENV : enviornament variables

ENV is the instruction to provide environment variables to image and container. You can override env variables at runtime

docker run -d <name of image> < environment variable-this is optionl>

VOLUME : it'used to create the volume in the container --> why volume ,to attach or mount the container's volume data to host machine(host volume), so that if the container exits , the data of container is still available in host volume .

VOLUME /mydir

LABEL --> It's used to give some meta data information , just like tags , and it's key value pair,

LABEL instruction to add metadata to your Docker image. Metadata in the form of key-value pairs can provide information about the image, such as version, maintainer, description, and more.

ex:

ex: jio and some other covers put together , you can easily filter based on labeling

vim Dockerfile

# Use an official Alpine Linux base image

FROM alpine:3.14

# Set the maintainer and description labels

LABEL maintainer="manjunath"

LABEL description="This is a sample Docker image with labels."

# Run any commands to configure your image

RUN apk add --update some-package

# Set the default command to run when the container starts

CMD ["sh"]

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

CMD VS ENTRYPOINT

MULTISTAGE DOCKERFILE OR DISTROLESS IMAGES -- SCRACH Images :

FROM openjdk:11

ARG JAR\_FILE=target/\*.jar

COPY ${JAR\_FILE} app.jar

ENTRYPOINT ["java","-jar","/app.jar"] 8081

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

FROM ubuntu

COPY requirement.txt /app

COPY devops /app

RUN install python3 # need to be edited

ENTRYPOINT ["python3","manage.py","runserver","0.0.0.0:8080"] - --qa testing team

----------

FROM ubuntu

COPY requirement.txt /app

COPY devops /app

RUN install python3 # need to be edited

CMD ["python3","manage.py","runserver","0.0.0.0:8080"] -

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

FROM ubuntu

COPY requirement.txt /app

COPY devops /app

RUN install python3 # need to be edited

ENTRYPOINT ["python3"] --> non changeable

CMD ["manage.py","runserver","0.0.0.0:8080"] ---> changeable

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

FROM ubuntu

COPY requirement.txt /app

COPY devops /app

RUN install python3 # need to be edited

CMD ["python3"] --> changeable

ENTRYPOINT ["manage.py","runserver","0.0.0.0:8080"] ---> non changeable

=======

FROM ubuntu

COPY requirement.txt /app

COPY devops /app

RUN install python3 # need to be edited

ENTRYPOINT ["python3"] --> non changeable

CMD ["manage.py","runserver","0.0.0.0:8080"] ---> changeable

CMD ["python3","manage.py","runserver","0.0.0.0:8080"] -

--

CMD ["java","-jar","/app.jar"]

--

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

CMD :

CMD component that runs the container

CMD specifies what command to run within the container.

CMD :The entrypoint to execute a command @ the running container.

important point : your docker file must contains atleast one CMD to run your container for infinite time

CMD: Sets default parameters that can be overridden from the Docker command line interface (CLI) while running a docker container.

ex:

FROM ubuntu

CMD echo s1

docker build -t manjunathachar/sampleap:latest .

docker run -d manjunathachar/sampleap:latest

s1 -> which print s1

docker run -d manjunathachar/sampleap:latest echo s11

s11 -> which displays s11 ,it overridden or overwritten , it' going to override if u execute any arguments during runtime

RUN vs CMD

RUN command executes at the time of image creation

CMD command executes at the time of running container

FROM almalinux

RUN yum install nginx -y

CMD ["nginx", "-g", "daemon-off"]

CMD ["sleep", "20"]

# to get docker id -> docker ps -a -q

ENTRYPOINT: Sets default parameters that cannot be overridden while executing Docker containers with CLI parameters.

ENTRYPOINT is also used to run the container just like CMD. But there are few differences.

We cant override ENTRYPOINT, but we can override CMD.

We can't override ENTRYPOINT, if you try to do so it will go and append to the ENTRYPOINT command.

If you use CMD and ENTRYPOINT and dont give any command from terminal, CMD acts as argument provider to ENTRYPOINT.

CMD will supply default arguments to ENTRYPOINT.

You can always override CMD arguments from runtime.

You can stop misusing your image with other commands.

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

cmd vs entry point

FROM ubuntu

COPY requirement.txt /app

COPY devops /app

RUN install python3 # need to be edited

ENTRYPOINT ["python3"] --> non changeable

CMD ["manage.py","runserver","0.0.0.0:8080"] ---> changeable

CMD ["python3","manage.py","runserver","0.0.0.0:8080"]

both of them , just like starting commands for the containers

entry point : it's a something that can't change

cmd : it's configurable command , we can change at run time

always have main executable in the entrypoint [python3] --> i dont want to change my pyhon3 to nodejs application, coz this is not a nodejs,

entrypoint have always non overiddable value ,we can't override

if you dont want your users to change anything in this ["python3","manage.py","runserver","0.0.0.0:8080"] , you can use same parameter in entrypoint ENTRYPOINT ["python3","manage.py","runserver","0.0.0.0:8080"],

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

creation of docker images using docker files :

FROM ubuntu

RUN apt-get update -y

RUN mkdir myfirstdir

RUN echo "my firest dockeer file content" >> file1.txt

docker build -t <dockerhub username>/<imagename>:v1 .

to push the image from local machine to docker hub .

docker login -u <dockerhub username>

password : < dockerhub token id>

docker push <created image>:tag

ex : ENV

# Use the base image (AlmaLinux in this case)

FROM almalinux

# Set environment variables

ENV AUTHOR="manjunath" \

DESCRIPTION="25HR"

# Add metadata to the image

LABEL maintainer="${AUTHOR}" \

description="${DESCRIPTION}"

# Specify the command to run when the container starts

CMD ["/bin/bash"]

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

FROM almalinux

RUN yum install nginx -y

CMD ["nginx", "-g", "daemon-off"]

CMD ["sleep", "20"]

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

FROM ubuntu:latest

# Update the package list and install Nginx

RUN apt-get update && apt-get install -y nginx

# Create a folder to store some files

RUN mkdir -p /var/www/html

# Create an index.html file with content

RUN echo 'Hi, I am in your container' > /var/www/html/index.html

# Expose port 80 for incoming connections

EXPOSE 80

# Start Nginx when the container starts

CMD ["nginx", "-g", "daemon off;"]

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

FROM openjdk:11

ARG JAR\_FILE=target/\*.jar

COPY ${JAR\_FILE} app.jar

ENTRYPOINT ["java","-jar","/app.jar"]

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

FROM openjdk:11

ARG JAR\_FILE=target/\*.jar

COPY ${JAR\_FILE} app.jar

# Set an environment variable to specify the port

ENV PORT 8082

# Expose the port for the container

EXPOSE ${PORT}

# CMD instruction to run the application with the specified port

CMD ["java", "-jar", "/app.jar", "--server.port=${PORT}"]

---

FROM ubuntu

WORKDIR /app

ADD https://download.sonatype.com/nexus/3/nexus-3.59.0-01-unix.tar.gz

RUN tar -zxvf nexus-3.59.0-01-unix.tar.gz

RUN mv /opt/nexus-3.59.0-01 /opt/nexus

VOLUME /opt/nexus

CMD ["/opt/nexus","start"]

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

# Use an official Ubuntu as a base image

FROM ubuntu

# Set the working directory to /app

WORKDIR /app

# Download Nexus and extract it

RUN apt-get update && apt-get install -y wget && \

wget https://download.sonatype.com/nexus/3/nexus-3.59.0-01-unix.tar.gz && \

tar -zxvf nexus-3.59.0-01-unix.tar.gz -C /opt/ && \

rm nexus-3.59.0-01-unix.tar.gz

# Rename the Nexus directory

RUN mv /opt/nexus-3.59.0-01 /opt/nexus

# Define a volume for Nexus data

VOLUME /opt/nexus

# Start Nexus

CMD ["/opt/nexus/bin/nexus", "run"]

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

TASKS

task1

clone the insureme project

modify the docker file , replace the entrypoint with cmd instruction and specify the different port number in order to run the application .

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

task2

size

ex : original image is ---> mynewimage latest cdcdf0e349e9 2 days ago 695MB

reduce the image size using multi build stages like build stage and run stages in docker file or use any disto images .

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

Docker Compose

===========

pre-requisites

docker

sudo yum update -y

sudo amazon-linux-extras install docker -y

sudo systemctl start docker

sudo systemctl enable docker

sudo systemctl status docker

sudo usermod -a -G docker ec2-user

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

How Can We Easily and Visually Explain Docker-Compose?

source : https://medium.com/clarusway/how-can-we-easily-and-visually-explain-the-docker-compose-53df77e9f046

Docker Compose is a tool that facilitates the composition of multiple images into a single container or service

docker compose is a tool that allows you to run multiple containerized applications

docker compose uses a yaml files to define the services , networks and volumes that make up your applications and then uses that files to create and manage the containers needed to run your applcations

to build the entire app --> front end -- html --> backend-businesslogics - java,python,nodejs--> dbbase-mysql

or

Docker Compose uses YAML files to describe what your applications need, like the different parts they consist of (services), how they connect (networks), and where they store data (volumes). It then takes these descriptions and sets up and oversees the containers needed to make your applications work.

-> it's used for running multiple containers as single service

-> here containers run in isolation but can interact with each other

-> in docker compose , a user can start all services(containers) using a single command

ex:

if you want to run web application, where dependencies like user interface ,application server(tomcat), database(mysql) are required, so instead running all containers individually, we can create docker-compose file or yaml file, which can run all the containers as a service

commands

docker-compose --help -> to get all commands which help to use docker-compose commands efficiently

docker-compose up -d -> to create and start the containers

docker-compose stop -> to stop the services

docker-compose ps -a -> list of all docker compose containers

docker-compose down -> to stop and remove the containers, networks

installation of docker-compose

docker-compose is not a part of docker package, so we have to install separately

sudo curl -L "https://github.com/docker/compose/releases/download/1.29.2/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose

# Apply executable permissions to the binary.

sudo chmod +x /usr/local/bin/docker-compose

sudo yum install docker-compose

vim .bash\_profile

export PATH=$PATH:$HOME/bin:$JAVA\_HOME/bin:$MAVEN\_HOME:$M2:/usr/local/bin

vim .bashrc

export PATH=$PATH:$HOME/bin:$JAVA\_HOME/bin:$MAVEN\_HOME:$M2:/usr/local/bin

source ~/.bash\_profile

source ~/.bashrc

docker-compose --version or docker-compose -version

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

version: '3'

services:

webserver:

image: nginx:latest

container\_image: name: nginx

ports:

- "80:80"

volumes:

- ./nginx.conf:/etc/nginx/nginx.conf # You can customize nginx configuration by providing a custom nginx.conf file

git:

image: alpine/git:latest

container\_name: git

command: ["tail", "-f", "/dev/null"] # Keeps the container running without exiting

dockerfile

FROM mysql:5.7

ENV MYSQL\_ROOT\_PASSWORD: manju65char

ENV MYSQL\_DATABASE: wordpress

EXPOSE 3306

version: '3'

services:

webserver:

build:

context: .

dockerfile: Dockerfile

ports:

- "9999:8081"

image: "manjunathchar/myapp"

docker-compose up -d

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

vim docker-compose.yaml

version: '3'

services:

db:

image: mysql:5.7

volumes:

- ./mysql-data:/var/lib/mysql

restart: always

environment:

MYSQL\_ROOT\_PASSWORD: manju65char

MYSQL\_DATABASE: wordpress

MYSQL\_USER: wordpress

MYSQL\_PASSWORD: wordpress

wordpress:

depends\_on: :

- db

image: wordpress:latest

ports:

- "80:80"

restart: always

environment:

WORDPRESS\_DB\_HOST: db:3306 -->

WORDPRESS\_DB\_USER: wordpress

WORDPRESS\_DB\_PASSWORD: wordpress

WORDPRESS\_DB\_NAME: wordpress

volumes:

- ./wp-content:/var/www/html/wp-content

docker-compose up -d -> to create and start the containers

docker-compose stop -> to stop the services

docker-compose ps -a -> list of all docker compose containers

docker-compose down -> to stop and remove the containers, networks

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

docker volume :

volume --> storage blocks

drives --> root volumes

b -->

c -->

----

what's nature of container : ephemeral in nature db containers , web server containers ,

docker volume --> it's persistant or permanent volume , host volume ,--> located in docker host

Volume ???

Storage blocks/units.

dir1/file1.txt

cloud / on-prem ===>

s3 buckets

SSD

HDD

Cluster volume

Gluster Volume

ex: consider i have an application, this app get deployed using container (scope of app with in a container) or without container ( on host machine )

there are 2 types of application in volume perspective to deploy applications

1. Stateless Application

2. Statefull Application

stateless app :

whenever running any app , if app deosn't require any volume in target environment on local machine

ex : google drive -> once we close g-drive we dont have trace of execution of it .

statefull application :

app1 -->volumes app2

some applications require volume, whenever run any app , which entire target environment and execute some tasks and leave some trace , the trace can be , in the form of logs and reports

ex : if you are created webapp , and trying to monitor cpu of machine and produce some results in the form of reports

-> earlier ( 10 years ago ) , containers were used only for stateless applications

it's used to run the tasks

when the microservices architecture is introduced , the deveolopers started to creating stateful applications

UI --WEB PAGE micro1

APP TIER -- BUSINESS LOGICS --> LOGIN TAB -- micro2

DATABASE --username and password micro3

Micro-service1 Micro-service2

(User-Login\_Screen) (Application Landing Page)

container1 username,password container2 (username)

(Container1 Volume) <--------------> (Container2 Volume)

copy

containers are dont have its filesystem default, it uses the host file system . to see where the containers data are stored

docker inspect <containerid> --> get dir path

/var/snap/docker/common/var-lib-docker/overlay2/

var/lib/docker//overlay2/

----

docker volume

docker volume create volume1 -->

docker volume create nginx

cd /var/lib/docker/volumes/volume1/-data -> as soon as you create volume , on docker host , the docker host will create volume folder , once you are get into the container , and create some file and those file

docker run -d -p 9999:80 --name ng nginx:latest

docker rm -f container name

docker run -d -v nginx:/usr/share/nginx/html -p 80:80 --name ng nginx:latest

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

containers are dont have its filesystem default, it uses the host file system . to see where the containers data are stored

docker inspect <containerid> --> get dir path

/var/snap/docker/common/var-lib-docker/overlay2/

var/lib/docker//overlay2/

----

docker volume

docker volume create volume1 -->

docker volume create nginx

cd /var/lib/docker/volumes/volume1/-data -> as soon as you create volume , on docker host , the docker host will create volume folder , once you are get into the container , and create some file and those file

docker run -d -p 9999:80 --name ng nginx:latest

docker rm -f container name

docker run -d -v nginx:/usr/share/nginx/html -p 80:80 --name ng nginx:latest

docker run -d -v nginx1:/usr/share/nginx/html -p 80:80 --name ng nginx:latest

alias dvl='docker volume ls' (You can add this line to your shell's configuration file (e.g., .bashrc ($ dvl) or .bash\_profile) to make the alias available every time you open a new terminal session.)

docker volume ls

docker volume inspect volume1 to get complete info of volume or location of volume

cd /var/lib/docker/volumes/volume1/\_data --> create a html file --> http://34.203.77.138/hi.html --> this in host dir

docker run -d -v nginx:/usr/share/nginx/html -p 80:80 --name ng nginx:latest

-------

create a dir , and host it

mkdir test-data

docker run -d -v /root/test-data:/usr/share/nginx/html -p 8989:80 --name ng2 nginx:latest

cd test-data

echo " hello guys" >> me.html

http://34.203.77.138:8989/me.html

docker volume ls --> you wont see the test-data volume , coz it's anonymous volume - it's not managed by docker , for the best practice you are supposed to use docker volume command to create a volume , it's recommended .

how to attach volume to container

docker run -d -v volume1

-p host-port:container-port

-v host-path:container path - inside container where you want mount the volume or data

docker run -d -v volume:/u

docker run -d -v volume1:/usr/share/volume1/

inside the host you will have directory for volume

that might be /var/lib/docker/volumes/

this directory can be mounted with any path inside the container

mounted with

docker host ----------------------------------------------------------------------- container

/var/lib/docker/volumes/volume1/nginx\_data usr/share/nginx/html

FROM ubuntu

RUN mkdir /myvol

RUN echo "hello world" > /myvol/greeting

VOLUME /myvol

----------

lets take a mysql image

docker volume create mysql-data

docker run -d -v mysql-data:/var/lib/mysql mysql ( this is mandatory path /var/lib/mysql for mysql)

9a46d342c8a1 mysql "docker-entrypoint.s…" 25 seconds ago Exited (1) 19 second

once you run the above command , you mysql container will be in exit status , so check with logs

docker logs 9a46d342c8a1

2023-10-14 10:04:33+00:00 [Note] [Entrypoint]: Entrypoint script for MySQL Server 8.1.0-1.el8 started.

2023-10-14 10:04:33+00:00 [Note] [Entrypoint]: Switching to dedicated user 'mysql'

2023-10-14 10:04:33+00:00 [Note] [Entrypoint]: Entrypoint script for MySQL Server 8.1.0-1.el8 started.

2023-10-14 10:04:33+00:00 [ERROR] [Entrypoint]: Database is uninitialized and password option is not specified

You need to specify one of the following as an environment variable:

- MYSQL\_ROOT\_PASSWORD

- MYSQL\_ALLOW\_EMPTY\_PASSWORD

- MYSQL\_RANDOM\_ROOT\_PASSWORD

you need to specify the password in evnvironment variable so , you need to give password when you are initializing the database

docker run -d -v mysql-data:/var/lib/mysql -e MYSQL\_ROOT\_PASSWORD=12345 mysql

docker exec -it <container id updated mysql> /bin/bash

mysql -u root -p12345 or

mysql -u root -p

enterpassword : 12345

create database devopsIIHT;

show databases;

mysql> show databases;

+--------------------+

| Database |

+--------------------+

| devopsIIHT |

| information\_schema |

| mysql |

| performance\_schema |

| sys |

+--------------------+

5 rows in set (0.01 sec)

exit;

next : remove the mysql container -->

cd /var/lib/docker/volumes/mysql-data/\_data

docker run -d -v mysql-data:/var/lib/mysql -e MYSQL\_ROOT\_PASSWORD=12345 mysql ( once run this one , you are data will be visible in the new container )

-----

important :how to be shown same data to another container or how to host/attach/mount same volume to multiple containers

docker volumes ls

[root@ip-172-31-62-19 \_data]# docker volume ls

DRIVER VOLUME NAME

local mysql-data

local nginx

lemme mount the same volume( nginx) to multiple containers

docker run -d -v nginx:/usr/share/nginx/html -p 80:80 --name ng1 nginx:latest

docker run -d -v nginx:/usr/share/nginx/html -p 8089:80 --name ng2 nginx:latest

docker run -d -v nginx:/usr/share/nginx/html -p 8099:80 --name ng3 nginx:latest

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

(Host-Volume)

(/opt/dir1/username.json)

Volume Mapping/Binding/attaching/mounting

why we need volume ?

whenever you run any microservices if it relies on another micro service , it's able to transfer data from one server to another server

here micro service1 output is input for microservice 2 , so micro sevice2 depends on microservice1, so to store data of each micro services we need volume

we can't copy the data from container1 to container2 directly, so whereas we use HOST VOLUME, which helps us to transfer data from one container to another container

volume mapping is used to binding the host volume to container volume

because containers are not a persistance one

as soon as the container task is over, it will exit or stop automatically , ( what are the data which contains in container are no permanent , they are temporary , so to overcome this where we use host voloume,

---

docker volume commands :

docker volume create <volume name> -->

docker volume ls

docker volume inspect -> Inspect a volume - to get complete information of the volume and mountpoint or volume location

docker volume rm my-vol

docker volume prune --> to remove the volumes which are unused or if you delete the container , eventhough the attached volume it not deleted, in that case you can this command to delete all rarely used or unused volumes

mounting a volume using -v or --mount

cd /var/lib/docker/volumes/volume1/-data -> host volume location

as soon as you create volume , on docker host , the docker host will create volume folder , once you are get into the container , and create some file and those file are reflected in the host volume and once you delete container , those data still available in host volume inside the data .

docker volume testvol1 -> to create a volume ,

docker volume inspect testvol1 -

"Mountpoint": "/var/snap/docker/common/var-lib-docker/volumes/testvol1/\_data", --> volume path , it's not mounted/mapped to any container lets mount it

docker run -it --name myvolcontainer --mount source=testvol1,destination=/volume1 ubuntu --> using this command , you are mounting the host volume(testvol1) to container's volume(volume1) using ubuntu image

now you are inside the container

ls

where you can volume1 dir --> create some files

docker run -it --mount source=volume1,destination=/volume1 ubuntu bash

--

if you want transfer data from host volume to container2 volume

docker run -it --volumes-from < source or existing containername,container1 > --name <new continer name,container2> ubuntu /bin/bash

-----

containers are dont have its filesystem default, it uses the host file system . to see where the containers data are stored

docker inspect <containerid> --> get dir path

/var/snap/docker/common/var-lib-docker/overlay2/

var/lib/docker//overlay2/

----

docker volume

docker volume create volume1 -->

docker volume create nginx

cd /var/lib/docker/volumes/volume1/-data -> as soon as you create volume , on docker host , the docker host will create volume folder , once you are get into the container , and create some file and those file

docker run -d -p 9999:80 --name ng nginx:latest

docker rm -f container name

docker run -d -v nginx:/usr/share/nginx/html -p 80:80 --name ng nginx:latest

alias dvl='docker volume ls' (You can add this line to your shell's configuration file (e.g., .bashrc ($ dvl) or .bash\_profile) to make the alias available every time you open a new terminal session.)

docker volume ls

docker volume inspect volume1 to get complete info of volume or location of volume

cd /var/lib/docker/volumes/volume1/\_data --> create a html file --> http://34.203.77.138/hi.html --> this in host dir

now just remove the container --> docker rm -f con id --> the data is still exist in volume , then refresh the http://34.203.77.138/hi.html , execute the same below command to add the volume to container ( u can use same host volume path to another container)

docker run -d -v nginx:/usr/share/nginx/html -p 80:80 --name ng nginx:latest

-------

create a dir , and host it

mkdir test-data

docker run -d -v /root/test-data:/usr/share/nginx/html -p 8989:80 --name ng2 nginx:latest

cd test-data

echo " hello guys" >> me.html

http://34.203.77.138:8989/me.html

docker volume ls --> you wont see the test-data volume , coz it's anonymous volume - it's not managed by docker , for the best practice you are supposed to use docker volume command to create a volume , it's recommended .

how to attach volume to container

docker run -d -v volume1

-p host-port:container-port

-v host-path:container path - inside container where you want mount the volume or data

docker run -d -v volume:/u

docker run -d -v volume1:/usr/share/volume1/

inside the host you will have directory for volume

that might be /var/lib/docker/volumes/

this directory can be mounted with any path inside the container

mounted with

docker host ----------------------------------------------------------------------- container

/var/lib/docker/volumes/volume1/nginx\_data usr/share/nginx/html

FROM ubuntu

RUN mkdir /myvol

RUN echo "hello world" > /myvol/greeting

VOLUME /myvol

----------

lets take a mysql image

docker volume create mysql-data

docker run -d -v mysql-data:/var/lib/mysql mysql ( this is mandatory path /var/lib/mysql for mysql)

9a46d342c8a1 mysql "docker-entrypoint.s…" 25 seconds ago Exited (1) 19 second

once you run the above command , you mysql container will be in exit status , so check with logs

docker logs 9a46d342c8a1

2023-10-14 10:04:33+00:00 [Note] [Entrypoint]: Entrypoint script for MySQL Server 8.1.0-1.el8 started.

2023-10-14 10:04:33+00:00 [Note] [Entrypoint]: Switching to dedicated user 'mysql'

2023-10-14 10:04:33+00:00 [Note] [Entrypoint]: Entrypoint script for MySQL Server 8.1.0-1.el8 started.

2023-10-14 10:04:33+00:00 [ERROR] [Entrypoint]: Database is uninitialized and password option is not specified

You need to specify one of the following as an environment variable:

- MYSQL\_ROOT\_PASSWORD

- MYSQL\_ALLOW\_EMPTY\_PASSWORD

- MYSQL\_RANDOM\_ROOT\_PASSWORD

you need to specify the password in evnvironment variable so , you need to give password when you are initializing the database

docker run -d -v mysql-data:/var/lib/mysql -e MYSQL\_ROOT\_PASSWORD=12345 mysql

docker exec -it <container id updated mysql> /bin/bash

mysql -u root -p12345 or

mysql -u root -p

enterpassword : 12345

create database devopsIIHT;

show databases;

mysql> show databases;

+--------------------+

| Database |

+--------------------+

| devopsIIHT |

| information\_schema |

| mysql |

| performance\_schema |

| sys |

+--------------------+

5 rows in set (0.01 sec)

exit;

next : remove the mysql container -->

cd /var/lib/docker/volumes/mysql-data/\_data

docker run -d -v mysql-data:/var/lib/mysql -e MYSQL\_ROOT\_PASSWORD=12345 mysql ( once run this one , you are data will be visible in the new container )

-----

important :how to be shown same data to another container or how to host/attach/mount same volume to multiple containers

docker volumes ls

[root@ip-172-31-62-19 \_data]# docker volume ls

DRIVER VOLUME NAME

local mysql-data

local nginx

lemme mount the same volume( nginx) to multiple containers

docker run -d -v nginx:/usr/share/nginx/html -p 80:80 --name ng1 nginx:latest

docker run -d -v nginx:/usr/share/nginx/html -p 8089:80 --name ng2 nginx:latest

docker run -d -v nginx:/usr/share/nginx/html -p 8099:80 --name ng3 nginx:latest

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

docker networks

nginx run -d -p 8989:80 --name mynginx nginx:latest --> default network - bridge network

3 tier ->

roboshop- ecommerce web app

namespace -> roboshop

roboshop\_nw

hr marketing team ,sale team it team

web tier public subnet

app tier private subnet

db tier private subnet

bridge network

docker network create <network\_name> --> Creates a new user-defined bridge network.

host network

none network

---

there are 3 types

-------

bridge

whenever you create a container by default it's created using bridge network

if you want expose your application where we use dockerhost ip with host port and container port

for every container we use different host port number to expose the app on browser ( 8080:80, 8081:80, 8888:80 etc...)

docker network inspect bridge --> to see complete inform of bridge network, you will get what are the containers attached to this network .

most widely using network

EX :

docker run -d -p 8989:80 --name mynginx nginx:latest

--------

host

if you want create container with host network , where you can use host network --

docker run -d --name http01 --net host http

docker run -d --name mynginx2 --net host nginx:latest

now we can run the application over internet with hostip or ec2 publicip without using port number 80

in bridge we defined the -p 8090:80

in host --net host --> so that it can run with 80 , but if you want run one more application with same port number , then it's not possible ,

coz , nginx is attached to host so , only single port can run

use only in limited situation

------

none

it's disable the network while creating containers

docker run -d --name nginx3 --net none nginx:latest

if you create container with non network,so it's not communicate internally and externally, so it's used for testing purpose , where you can't run the application internally and over the browser

usually we dont use none network

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

docker netwoking,

docker volumes

best practice of docker file and docker composefile

docker images must be lightweigt images, more secured --

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

using the docker engine --> create, delete , update the images and containers

docker swarm ->

k8s --> kubernetes ( k ubernete s )

1000 micro services --> 1000 containers

container orchestration tool :

k8s --> objects

k8s architecture :

kubectl

k8s cluster :

1. master node

kube api server

kube scheduler

etcd

controller manager

2. worker nodes

kubelet

kubeproxy

cri-docker engine

pods -> collection of containers

cAdvisor -> monitoring tool for the containers

==

k8s manifestfile/ k8s yaml files

apiVersion : apps/v1

kind : pod

metadata

spec

apiVersion : v1

kind

metadata

spec

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

k8s cluster setup :

oss or cloud

aws -> eks -> elastic k8s service

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

k8s :

it'a opensource tool,

it's container orchestration tool, which perform for automating deployment,scaling and mage of containerized work loads(deployments, replicasets,stateful sets,HPA and so on )

why we need k8s :

managing workloads and services

high availability of containers

orchestrate containers ( deployed a service in PROD env)

application must be up and running 24/7

===

k8s architecture

kubectl

yaml file

masternode/control plane --> which control the actions

workernode/data plane ---> which execute the actions ( each worker node contains kubelet,kube proxy, cri-docker engine)

master node :

api server

scheduler

etcd

controller manager

worker node :

kubelet

kubeproxy

cri- docker

pods...

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

k8s-manifest file-- yaml file

container : it's package of application and its dependencies

Pods -->it's collection of containers and Smallest deployable units in Kubernetes that can hold one or

multiple containers.

Nodes: Worker machines in Kubernetes.and collections of pods,

cluster : collection of nodes , including master node and worker node that run containerized applications

Kubectl: Command-line tool for interacting with a Kubernetes cluster

it's a command line interpreter, that allows us to interact with k8s cluster

api server ?

it's front end interface for k8s k8s masternode

he API Server is a critical part of the Kubernetes control plane, and it serves as a front-end interface through which users, administrators, and various Kubernetes components interact with the cluster's control plane.

whenever you submit a file to k8s cluster, the intially the request is received by api server, so the role of api server

is to receive incoming request and verify the user that , who is submitted and whether he is authorised or not, and what are the things he performs in the server

ETCD : : Consistent and highly-available key-value store used as Kubernetes' backing store for all cluster data.

API server updates all the information(of containers and nodes) to etcd, it's a etc database

scheduler :

Once it's updated ,etcd gives controller to scheduler, then scheduler will interact to api server through etcd and collect the latest infomation, and identify the healthy nodes to deploy the containers.

NEXT STEP : with this information api server will interact with its correspondence worker nodes

kubelet : it's interface between workernode and masternode

docker engine

---

kube proxy --> it enable and manage the network connection between conatiners , nodes in the k8s cluster .

controller manager : it's responsible for implementing strategies , ( high availability, replicas ,scale up/scaledown )

it's monitoring the running containers and it always ensure that,the containers are running as expected ...

---

k8s distributions /technologies/Kubernetes Development Tools:

k3s

Minikube: Runs a single-node Kubernetes cluster inside a VM on your laptop for users looking to try out Kubernetes or develop

with it day-to-day.

Skaffold: Command line tool that facilitates continuous development for Kubernetes applications.

Kompose: Conversion tool for all Docker Compose users to help them

move to Kubernetes.

Kubeadm: Tool for bootstrapping a best-practice Kubernetes

cluster.

-----

Services --> it's used to expose your containerized application outside k8s cluster or over internet ,

A way to expose an application running in Pods as a network service

types

cluster ip :

NodePort :

LoadBalancer :

-----

Advanced Kubernetes Networking:

Network Plugins: Extend Kubernetes networking.

CNI (Container Network Interface): Standard for writing plugins to configure network interfaces in Linux containers.

Flannel: Overlay network provider.

Calico: Provides secure network connectivity.

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

#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

#Kube-Master(Controller) - VM

# Kube-Worker1

# Kube-Worker2

#https://kubernetes.io/docs/setup/

#Add Port: 0 - 65535 /default cidr block is class b /16 --> ipv4 - 32 bits/ 4 octets /8 bits 2^8=256 -> 16 bits -> 256\*256=65536

#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

On Both Master and Worker Nodes:

#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

sudo -i

yum update -y

swapoff -a

#The Kubernetes scheduler determines the best available node on which to deploy newly created pods. If memory swapping is allowed to occur on a host system, this can lead to performance and stability issues within Kubernetes.

setenforce 0

#Disabling the SElinux makes all containers can easily access host filesystem.

yum install docker -y

systemctl start docker

systemctl enable docker

systemctl status docker

cat <<EOF > /etc/yum.repos.d/kubernetes.repo

[kubernetes]

name=Kubernetes

baseurl=https://packages.cloud.google.com/yum/repos/kubernetes-el7-x86\_64

enabled=1

gpgcheck=1

repo\_gpgcheck=0

gpgkey=https://packages.cloud.google.com/yum/doc/yum-key.gpg https://packages.cloud.google.com/yum/doc/rpm-package-key.gpg

exclude=kube\*

EOF

cat <<EOF > /etc/sysctl.d/k8s.conf

net.bridge.bridge-nf-call-ip6tables = 1

net.bridge.bridge-nf-call-iptables = 1

EOF

sysctl --system

yum install -y kubeadm-1.21.3 kubelet-1.21.3 kubectl-1.21.3 --disableexcludes=kubernetes

systemctl start kubelet

systemctl enable kubelet

systemctl status docker

systemctl status kubelet

#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

#Only on Master Node:

#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

#provide the private ip of masternode ( ec2 instance )

sudo kubeadm init --apiserver-advertise-address=172.31.87.77 --pod-network-cidr=192.168.0.0/16 --ignore-preflight-errors=NumCPU --ignore-preflight-errors=Mem

mkdir -p $HOME/.kube

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

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

#export KUBECONFIG=/etc/kubernetes/kubelet.conf

#We need to install a flannel network plugin to run coredns to start pod network communication.

sudo kubectl apply -f https://docs.projectcalico.org/v3.3/getting-started/kubernetes/installation/hosted/rbac-kdd.yaml

sudo kubectl apply -f https://docs.projectcalico.org/v3.8/getting-started/kubernetes/installation/hosted/kubernetes-datastore/calico-networking/1.7/calico.yaml

#Test the configuration ::

kubectl get pods --all-namespaces

#Generate NEW Token :

kubeadm token create --print-join-command

kubectl get nodes

kubectl describe nodes

#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

#Execute the below commmand in Worker Nodes, to join all the worker nodes with Master :

#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

kubeadm join 172.31.87.77:6443 --token gzqr8z.j7tkf7qgnu5cw65s --discovery-token-ca-cert-hash sha256:a92b395abece36603c826627aacc587d693a272491580c6fc676ceca241583ca

#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

integration with jenkins

#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

#Add-USer for k8s :

useradd -m -d /home/devopsadmin devopsadmin

su - devopsadmin

ssh-keygen

ls ~/.ssh

#You should see following two files:

#id\_rsa - private key

#id\_rsa.pub - public

cd /home/devopsadmin/.ssh

cat id\_rsa.pub > authorized\_keys

sudo chown -R devopsadmin /home/devopsadmin/.ssh # run in root ,add devopsadmin to visudo ,root and no password

sudo chmod 600 /home/devopsadmin/.ssh/authorized\_keys

sudo chmod 700 /home/devopsadmin/.ssh

#make devopsadmin user as a owner to home dir of devopsadmin dir :

chown -R devopsadmin /home/devopsadmin

#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

k8s objects/k8s resources / workloads

Pod:

A Pod is the smallest and simplest unit in the Kubernetes object model. It represents a single instance of a running process in a cluster. A Pod can contain one or more tightly related containers, such as an application container and a sidecar container( supporting container to main application continer .

Example: Imagine a Pod called "web-app" that runs a web server (e.g., Nginx) and a sidecar container responsible for logging.

Deployment/deployment set :

it's default deployment strategy( rolling update ),create replica set, pods , auto scaling , auto healing feature

A Deployment manages the desired state of ReplicaSets, ensuring a specified number of Pod replicas are running and updating them when needed. It allows easy scaling and rolling updates for your application.

replicas : 3

replicaSet: No of copies(replicas) of the containerized applications

Example: A Deployment named "web-deployment" that ensures there are always three replicas of the "web-app" Pod running.

Service:

types

cluster ip

NodePort

LoadBalancer

A Service provides a stable endpoint (IP address:port) to connect to a group of Pods. It enables load balancing and automatic routing to available Pods behind the Service.

Example: A Service called "web-service" that exposes the "web-app" Pods, allowing external users to access the web server.

Volume:

A Volume is used to persist data beyond the lifetime of a Pod. It enables sharing data between containers in the same Pod and provides a way to store data that survives Pod restarts.

Example: A Volume named "data-volume" attached to the "web-app" Pod to store user uploads and preserve them across Pod restarts.

ConfigMap:

A ConfigMap holds configuration data as key-value pairs, which can be used as environment variables or configuration files within Pods.

Example: A ConfigMap named "app-config" that stores database connection settings or other application-specific configuration.

Secret:

Similar to ConfigMaps, Secrets store sensitive data like passwords, tokens, or API keys, providing a more secure way to handle confidential information.

Example: A Secret called "db-credentials" storing the database password for the application to access the database.

Namespace:

A Namespace is a virtual cluster within a physical Kubernetes cluster, allowing you to partition resources and create isolated environments.

Example: Creating a Namespace called "development" to segregate development-related resources from production.

DaemonSet:

A DaemonSet ensures that a copy of a Pod runs on each node in the cluster. It's typically used for system-level tasks or agents that should be deployed on every node.

Example: A DaemonSet that deploys a monitoring agent on every node to collect metrics.

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

5. StatefulSets:

StatefulSets are used for stateful applications that require unique network identities and persistent storage for each Pod.

Example: A StatefulSet for a database application, where each Pod has a stable hostname and corresponding storage.

9. PersistentVolumes (PV):

PersistentVolumes are storage resources in the cluster that exist independently of Pods.

Example: A PersistentVolume representing a network-attached storage device like a cloud disk.

10. PersistentVolumeClaims (PVC):

PersistentVolumeClaims request specific storage resources from available PersistentVolumes.

Example: A PersistentVolumeClaim to request 10GB of storage for a database.

---

12. Jobs:

Jobs create Pods to run tasks until completion, ensuring the task is completed successfully.

Example: A Job that runs a backup script once and terminates after the backup is done.

13. CronJobs:

CronJobs are used to schedule Jobs to run periodically at specified intervals using cron syntax.

Example: A CronJob to perform a database cleanup every night at midnight.

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

types of deployment strategies in k8s

Rolling Update:

Rolling Update is the default deployment strategy in Kubernetes. It updates the Pods in a Deployment or ReplicaSet gradually, one at a time, while ensuring the desired number of replicas is maintained throughout the process. This strategy ensures minimal downtime during updates.

Recreate:

The Recreate strategy terminates all existing Pods before creating new ones with the updated configuration. This results in a brief period of downtime during the deployment process.

Blue-Green Deployment:

In a Blue-Green Deployment, you have two identical environments (blue and green). You switch traffic from the old environment (blue) to the new one (green) after the update is complete. This strategy allows for instant rollback in case of issues.

Canary Deployment:

Canary Deployment gradually introduces the new version of an application to a subset of users while still serving the old version to the majority. It allows you to test the new version's performance and stability before rolling it out to everyone.

A/B Testing:

A/B Testing is similar to Canary Deployment, but instead of testing versions, it allows you to test different features or configurations for a subset of users.

Shadow Deployment:

Shadow Deployment sends a copy of production traffic to the new version (shadow deployment) without serving it to end-users. This allows you to monitor the behavior of the new version before making it fully active.

Rollback:

Rollback is not a deployment strategy itself, but it's a crucial feature in Kubernetes that allows you to revert to a previous stable version if an update causes issues.

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

scheduling and maintainance in k8s

In Kubernetes, scheduling and maintenance are crucial aspects of managing and maintaining the desired state of applications and resources within a cluster. These processes ensure that Pods are deployed and distributed efficiently across the available nodes and that the cluster remains healthy and operational. Let's delve into scheduling and maintenance in Kubernetes and explore their main components:

Scheduling:

---------

Scheduling in Kubernetes is the process of assigning Pods to nodes in the cluster based on resource requirements, constraints, and other policies. The Kubernetes scheduler handles this responsibility, aiming to optimize resource utilization and distribute workloads evenly across the cluster. The scheduler makes decisions based on the current state of the cluster, such as node capacity, affinity/anti-affinity rules, resource constraints, and Pod priorities.

Components of Scheduling:

Kube-scheduler:

The kube-scheduler is the core component responsible for making scheduling decisions. It watches for new or pending Pods and assigns them to suitable nodes based on configurable scheduling policies. The default scheduler in Kubernetes is the "DefaultScheduler."

Node Selector and Affinity/Anti-Affinity:

Node Selector allows you to specify constraints for which nodes Pods can be scheduled on based on node labels. Affinity and Anti-Affinity rules provide more advanced ways to influence Pod placement based on node characteristics or other Pods' locations.

Resource Requests and Limits:

Resource requests and limits help the scheduler understand the resource requirements of Pods and determine the best nodes to place them based on available resources.

Taints and Tolerations:

Taints are attributes applied to nodes to repel Pods from being scheduled on them. Tolerations, on the other hand, allow specific Pods to tolerate the taints and be scheduled on tainted nodes.

Node Readiness:

The scheduler considers node readiness when making scheduling decisions to avoid placing Pods on nodes that are not ready.

Priority and Preemption:

Pod priorities enable the scheduler to prioritize Pods based on importance. Preemption allows lower-priority Pods to be evicted to make room for higher-priority Pods if necessary.

Maintenance:

-----------

Maintenance in Kubernetes refers to the process of keeping the cluster healthy and operational. It involves monitoring the cluster, ensuring its components are running correctly, and taking necessary actions to address failures or prevent disruptions.

Components of Maintenance:

kubelet:

The kubelet is a node-level agent that runs on each node and ensures that Pods are running as expected. It communicates with the Kubernetes master node and manages the containers associated with each Pod.

Cluster AutoScaler:

The Cluster AutoScaler automatically adjusts the number of nodes in the cluster based on resource demands. It scales the cluster up or down to meet the workload requirements efficiently.

Node Problem Detector:

Node Problem Detector detects node-level problems, such as out-of-memory errors or kernel panics, and reports them to the kubelet for further action.

HorizontalPodAutoscaler (HPA):

While primarily a scaling component, the HPA also plays a role in maintenance by dynamically adjusting the number of replicas to meet resource demands and maintain optimal perform

#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

k8s cmds :

nodes commands -> no

kubectl get nodes --> > to display the nodes

kubectl get node -->To list down all worker nodes.

kubectl delete node <node\_name> --> Delete the given node in cluster.

kubectl top node -----> Show metrics for a given node.

kubectl describe nodes | grep ALLOCATED -A 5 --> Describe all the nodes in verbose.

kubectl get pods -o wide | grep <node\_name> ---> List all pods in the current namespace,with more details.

kubectl get no -o wide --> List all the nodes with mode details.

kubectl describe no ---> Describe the given node in verbose.

kubectl annotate node <node\_name> ---> Add an annotation for the given node.

kubectl uncordon node <node\_name> --> Mark my-node as schedulable.

kubectl label node --> Add a label to given node

pod commands : po

kubectl get pods ---> to display the pods

kubectl get pods --all-namespaces --> to get all pods from all namespaces

kubectl get po --> To list the available pods in the default namespace.

kubectl describe pod <pod\_name> --> To list the detailed description of pod.

kubectl delete pod <pod\_name> --> To delete a pod with the name.

kubectl create pod <pod\_name> --> To create a pod with the name.

Kubectl get pod -n <name\_space> --> To list all the pods in a namespace.

Kubectl create pod <pod\_name> -n <name\_space> --> To create a pod with the name in a namespace.

kubectl run myweb --image=nginx ---> create a pod from command line(this will create a pod with the name myweb)

kubectl delete pod myweb ---> to destroy the pod(this will delete a pod with the name myweb)

kubectl apply -f yaml\_file\_name -> > to apply yaml/config file to create a resource

kubectl delete -f yaml\_file\_name ---- > to destroy yaml/config file to delete a resource

kubectl create namespace name\_for\_namespace --> > create a namespace example: kubectl create namespace devops

either use "-n name\_of\_namespace" or "--namespace name\_of\_namespace" ---> to deploy to specific namespace

example: kubectl apply -f pod.yml -n devops or kubectl apply -f replica.yml --namespace devops

kubectl describe -n devops pod pod\_name ---> > to describe any resource

ex : kubectl describe -n devops deployment deployment\_name

kubectl descripbe deploy <name of deploy> --> to get complete inform of deploy

deployment strategies commands

kubectl get deploy --> to get all default deployments

kubectl describe deploy insurance-deploy ---> to get complete information of deployment

======

3. Namespaces : ns

#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

1. Create and display Pods

# Create and display PODs

kubectl create -f nginx-pod.yaml -->

kubectl get pods

kubectl get pod -o wide

kubectl get pod nginx-pod -o yaml

kubectl describe pod nginx-pod

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

pod creation

# nginx-pod.yaml or yml

vim firstpod.yaml

apiVersion: v1

kind: Pod

metadata:

name: nginx-pod

labels:

app: nginx

spec:

containers:

- name: nginx-container

image: nginx

ports:

- containerPort: 80

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

kubectl create -f firstpod.yaml docker build -t

kubectl apply -f firstpod.yaml docker run -d --> docker-compose up -d

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

creation of multi-containers pod

---

apiVersion: v1

kind: Pod

metadata:

name: multi

spec:

containers:

- name: nginx-container

image: nginx

ports:

- containerPort: 80

- name: almalinux-container

image: almalinux