GIT: GLOBAL INFORMATION TRACKER

WHY GIT?

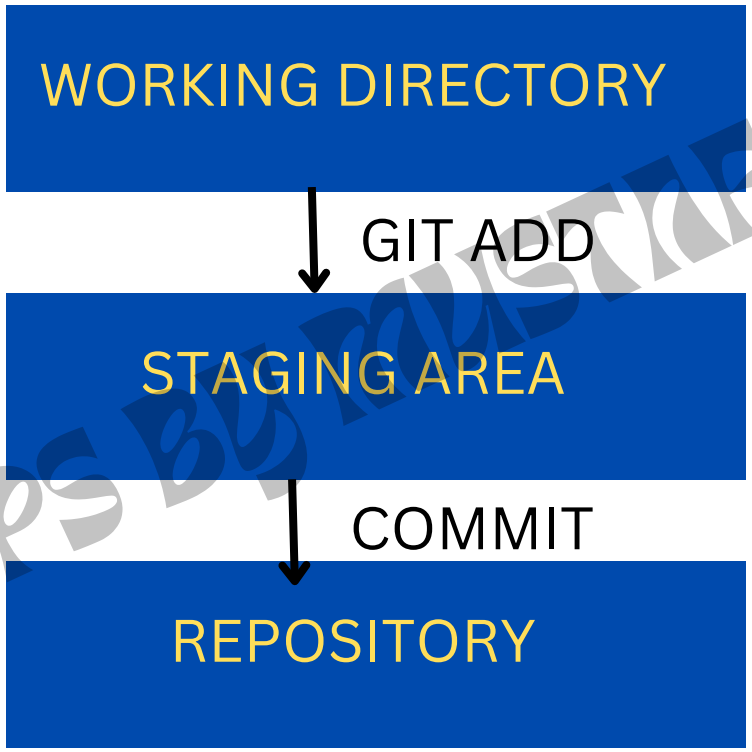
- IT IS USED FOR SOURCE CODE MANAGEMENT.
- GIT IS USED TO MAINTAIN MULTIPLE VERSIONS OF THE SAME FILE.

- GIT IS FREE AND OPEN SOURCE.
- GIT IS PLATFORM INDEPENDENT.

IT SUPPORTS LINUX, MAC OS, AND WINDOWS.

- GIT IS USED TO TRACK THE FILE.
- GIT WAS INTRODUCED IN THE YEAR OF 2005.

GIT STAGES:



WORKING DIRECTORY:

- In this stage git is only aware of having files in the project.
- It will not track these files until we commit those files.

STAGING AREA:

- The staging area is like a rough draft space, it's where you can git add the version of a file or multiple files that you want to save in your next commit.
- In other words, in the next version of your project.

REPOSITORY:

- Repository in Git is considered as your project folder.
- A repository has all the project-related data.
- It contains the collection of the files and also history of changes made to those files.

We have 3 repositories in GIT

- LOCAL REPO
- REMOTE REPO
- CENTRAL REPO

DEVOPS BY MUSTAFA

LOCAL REPO:

The Local Repository is everything in your .git directory. Mainly what you will see in your Local Repository are all of your checkpoints or commits. It is the area that saves everything (so don't delete it).

REMOTE REPO:

The remote repository is a Git repository that is stored on some remote computer.

CENTRAL REPO:

This will be present in our GITHUB

INSTALL GIT:

yum install git -y

yum: yellowdog updater modifier

To check the git version:

git --version

git init . : to get empty repo

To track the file:

git add file_name : single file

git add aws azure gcp: multiple files

git add * : all regular files

git add . : including hidden files

GIT INSTALL: yum install git -y
TO INITILIZE AN EMPTY REPO: git init .

here .(dot) represents the current directory (~). so it will creates .git folder (hidden)

to check the version: git --version
it returns: git version 2.38.1

GIT is used to track the file
to track a file

1. touch file_name : create a new file
2. git add file_name : add git to file
3. to check : git status

green color file represents tracking files
red color files represents untracking file

tracking means: we have a record that what we made the changes on a file.
it will track the each and every change in a file.

to commit a file: git commit -m "message" file_name

once we commit the file, it will goes to repository.
we can see the history of a file by using git log

DEVOPS BY MUSTAFA

to track multiple file: git add f1 f2 f3
to track all files: git add *
to track all files including hidden files: git add .

to commit multiple files: git commit -m "msg" f1 f2
to commit all tracking files: git commit -m "msg" .

.(dot) represents the all tracking files
notes: we can't commit the untracking files.

to untack a file: git rm --cached file_name
once we untrack a file, that files goes to working directory.

DEVOPS BY MUSTAFA

to untrack the files:

git rm --cached file_name

to commit file:

git commit -m "message" file_name

git commit -m "message" aws azure gcp

git commit -m "message" .

to check the history

git log

git log:

git log -2: latest 2 commit

git log --oneline: used to see only commit id's and commit messages

git config: it is used to configure the user with their mail id in git

command: **git config user.name "user_name"**

git config user.email "user@email.com"

git restore: used to get back the deleted files

command: **git restore file-name**

git show: it is used to show the file names in log

command: git show commit_id --name-only

git show <commit> --stat : you'll see the commit summary along with the files that changed and details on how they changed.
git commit --amend -m "New commit message" : to edit the commit message
git commit --amend --no-edit : used to add some files in previous commit. (--no-edit means that the commit message does not change.)
git update-ref -d HEAD : used to delete all the commits in git
git reset commit: used to delete all the commits (upto the commit id)

git reset --hard HEAD~1 : used to delete latest commit along with the changes
git reset --hard HEAD~N : upto nth commit
git revert commit-id : used to undo a middle of the change (file also deleted)

Let's say that you forgot to configure the email and already did your first commit. Amend can be used to change the author of your previous commit as well. The author of the commit can be changed using the following command:

git commit --amend --author "Author Name <Author Email>"

GIT BRANCH:

- A branch represents an independent line of development.
- A branch is a way to isolate development work on a particular aspect of a project.
- The git branch command lets you create, list, rename, and delete branches.
- The default branch name in Git is master.

to see the current branch: git branch
to create a branch: git branch branch_name
to go to a branch: git checkout branch
to delete a branch: git branch -d branch_name
to rename a branch: git branch -m old new
to create and switch at a time: git checkout -b branch_name

GIT MERGE:

It allows us to get the code from one branch to another. This is useful when developers work on the same code and want to integrate their changes before pushing them up in a branch.

command: git merge branch_name

GIT MERGE CONFLICTS

Merge conflicts happen when you merge branches that have competing commits, and Git needs your help to decide which changes to incorporate in the final merge.

GIT makes merging super easy!
CONFLICTS generally araise when two people two people have changed the same lines in a file (or) if one developer deleted a file while another developer is working on the same file!
In this situation git cannot determine what is correct!

Lets understand in a simple way!

cat>file1 : hai all
add & commit
git checkout -b branch1
cat>file1 : 1234
add & commit
git checkout master
cat>>file1 : abcd
add & commit

git merge branch1 : remove it

DEVOPS BY MUSTAFA

identify merge conflicts:

see the file in master branch then you will see both the data in a single file including branch names

resolve:

open file in VIM EDITOR and delete all the conflict dividers and save it! add git to that file and commit it with the command (git commit -m "merged and resolved the conflict issue in abc.txt")

merge:

if you have 5 commits in master branch and only 1 commit in devops branch, to get all the commits from master branch to devops branch we can use merge in git. (command: git merge branch_name)

cherry-pick:

if you have 5 commits in master branch and only 1 commit in devops branch, to get specific commit from master branch to devops branch we can use cherry pick in git. (git cherry-pick commit_id).

GIT STASH:

Using the git stash command, developers can temporarily save changes made in the working directory. It allows them to quickly switch contexts when they are not quite ready to commit changes. And it allows them to more easily switch between branches.
Generally, the stash's meaning is "store something safely in a hidden place."

DEVOPS BY MUSTAFA

- git stash
- git stash apply
- git stash list
- git stash pop
- git stash clear
- git stash drop stash@{2}

DEVOPS BY MUSTAFA

MERGE VS REBASE:

When there are changes on the master branch that you want to incorporate into your java, you can either merge the changes in or rebase your branch from a different point.

merge takes the changes from one branch and merges them into another branch in one merge commit.
rebase adjusts the point at which a branch actually branched off (i.e. moves the branch to a new starting point from the base branch)

to merge: git merge branch name
to rebase: git rebase branch_name

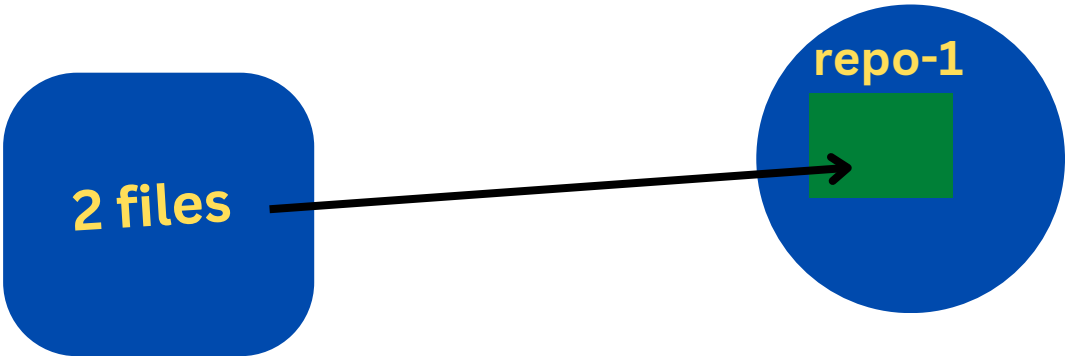
SOME EXTRA COMMITS:

git show <commit> --stat : you'll see the commit summary along with the files that changed and details on how they changed.
git commit --amend -m "New commit message" : to edit the commit message
git commit --amend --no-edit : used to add some files in previous commit. (--no-edit means that the commit message does not change.)
git update-ref -d HEAD : used to delete 1st commit in the git
git reset commit: used to delete all the commits (upto the commit id)
git commit --amend --author "Author Name <Author Email>" : used to change the author of latest commit

DEVOPS BY MUSTAFA

GIT RESET:

GIT HUB: it is a web based centralized platform which is used to share the code and store the code.



Git remote -v : used to see the remote repo
Git pull origin branch: used to get changes in GitHub
Git remote rename old -link new-link: change the repo
Git fetch origin master: to fetch from remote
Git merge origin/master : to merge from remote repo
git push origin --delete branch_name : to delete GitHub branch
Git fetch —all : used to fetch all the branches
git push -u origin --all : to push all branches
Git push -u origin branch1 branch2 : to push specific branches
git remote rm origin: removes the remote repo

Bash scripts can be used for various purposes,
such as executing a shell command, running multiple commands together, customizing administrative tasks, performing task automation etc.
So knowledge of bash programming basics is important for every Linux user.

```
#! /bin/bash

touch file1.txt
mkdir cloud
```

==========

```
#!/bin/bash
echo "Printing text with newline"
echo -n "Printing text without newline"
echo -e "\nRemoving \t backslash \t characters\n"
```

==========

```
#!/bin/bash
echo "Enter Your Name"
read name
echo "Welcome $name"
```

==========

MORE: it is used to see the multiple files in a single format

syntax: more file1 file2 file3

vim -o file1 file2 : compares the files in sequential
vim -O file1 file2 : compares the files in parallel

nl file_name: prints data along with numbers

cat -E file_name : prints the lines along with $ symbol
$ symbol represents the end of the line.

SORT: used to print the data in a alphabetical format
sort file_name
sort -r file_name: prints the data in a reverse order

nslookup: used to get the ip address of a website

alias: used to set a shortcuts to out commands
syntax: alias t=touch

MAVEN

- Maven is an automation project management tool.
- It is used to build the code
- once we build the code we will get JAR/WAR/EAR
- Maven is used to add the dependencies to our application.
- Maven is based on POM.xml

POM: Project Object Model

XML: Extensible Markup Language.

- POM.xml contains project related data (metadata, kind of project, kind of output, description, dependencies).
- Maven was developed by Apache software foundations.
- Maven was released in 2004.
- Maven can build any number of projects into desired Output such as .jar, .war and .ear

- .jar = java archive file
- .war = web archive file
- .EAR = enterprise archive

- It is mostly used for java-based projects.
- It was initially released on 13 July 2004.
- Maven is written in java.

MAVEN BUILD LIFE CYCLE:

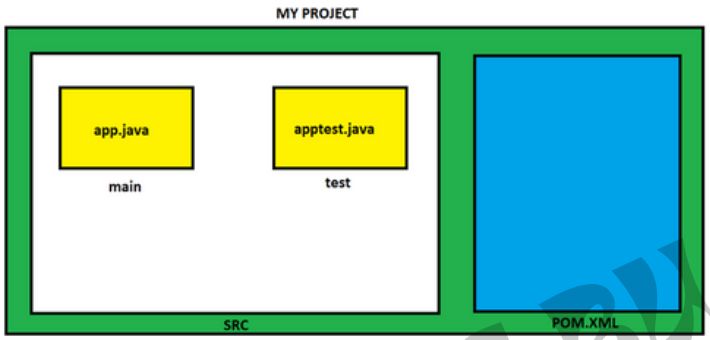
1 Generate Resource
2. Compile Code
3. Unit Test
4. Package (build)
5. install (into Local repo or Artifactory)
6. Deploy (to servers)
7. Clean (to delete all the runtime files)

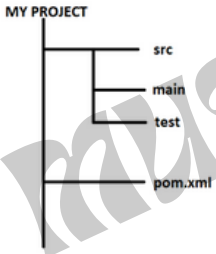
BUILD TOOL:

- it is used to set up everything which is required to run your java code This can be applied to your entire java project.
- It generates source code, compiling code, packaging code to a jar etc.
- POM refers the XML file that have all information regarding project and configuration details
- Main configuration file is in pom.xml.
- It has description of the Project details regarding version and configuration management.
- The XML file is in the Project home directory.

DEVOPS BY MUSTAFA

MAVEN DIRECTORY STRUCTURE:





step-1: download file from dlcdn.apache.org
wget https://dlcdn.apache.org/maven/maven-3/3.8.7/binaries/apache-maven-3.8.7-bin.tar.gz
wget ---> web get : used to download files from browser to server.

step-2: untar the file
tar -zxvf file_name

step-3: install java-1.8.0 version
yum install java-1.8.0-openjdk -y
to check the version: java -version

step-4: install maven
yum install maven -y
to check the version: mvn -v or mvn -version

step-5: generate source code
mvn archetype:generate
Choose a number or apply filter (format: [groupId:]artifactId, case sensitive contains): 1997: (enter)
Choose org.apache.maven.archetypes:maven-archetype-quickstart version: (select any number or keep enter)
Define value for property 'groupId': **DEVOPS**
Define value for property 'artifactId': **BATCH37**
Define value for property 'version' 1.0-SNAPSHOT: : (enter)
Define value for property 'package' DEVOPS: : (enter)

step-6: to compile the code
mvn compile
whenever we compile, target folder will gets created and we can see output on target folder,

step-7: test the source code
mvn test
once the test is done, some test folders will gets created in target folder

step-8: build the code
mvn package
once build is done, jar file will be created. (artifact_id-version-SNAPSHOT.JAR)

step-9: install the code
mvn install

step-10: clean the run time files (executable files)
mvn clean
delets all the target folder

to update java version: update-alternatives --config java

ANT VS MAVEN

Ant	Maven
Ant **doesn't has formal conventions**, so we need to provide information of the project structure in build.xml file.	Maven **has a convention** to place source code, compiled code etc. So we don't need to provide information about the project structure in pom.xml file.
Ant is **procedural**, you need to provide information about what to do and when to do through code. You need to provide order.	Maven is **declarative**, everything you define in the pom.xml file.
There is **no life cycle** in Ant.	There is **life cycle** in Maven.
It is **a tool** box.	It is **a framework**.
It is **mainly a build tool**.	It is **mainly a project management tool**.
The ant scripts are **not reusable**.	The maven plugins are **reusable**.
It is **less preferred** than Maven.	It is **more preferred** than Ant.

JENKINS:

Jenkins is a leading open-source, free automation tool that is used to develop and test software projects.

Reasons why Jenkins is widely used:

1. Used to detect faults in software development and systematize the testing of buildss
2. Used to constantly monitor the code simultaneously and add changes to the build.
3. Jenkins consists of an installer package for most operating systems.
4. Used to keep the team updated and synchronized about the changes incorporated.
5. Used to build CI/CD pipelines since it has plugin capabilities and is easy to use.

FEATURES OF JENKINS

Jenkins features include:

1. It is an open-source automation tool and it is free.
2. Gives pipeline support
3. installation is easy on systems with multiple operations.
4. large number of plugins
5. Jenkins upgrades effortlessly.
6. speedy release cycle
7. Configuration setup is easy.

CONTINUOUS INTEGRATION:

Continuous Integration is the continuous process of checking the code made by developers into a version control system numerous times. The build is automated in the process to inspect and detect bugs in the developed code

DEVOPS BY MUSTAFA

PIPELINE:

Jenkins Pipeline is collection of features of Jenkins. They are installed as plugins that allows delivery of pipeline implementation continuously.

ADVANTAGES OF JENKINS:

The advantages of using Jenkins are:

1. User-friendly, free, and it is an open source
2. Trouble-free Installation
3. Code deployment is convenient and takes very less time. It simultaneously generates reports.
4. Helps in collaboration between the operation and development teams.
5. Free of cost
6. Detection of code errors at an early stage
7. Reduced integration issues due to automation of integration work
8. Rich plugin ecosystem
9. Platform independence

PLUGIN:

It is a small software that we used to add some extra features to our dashboard.

JOB:

A Jenkins job is a sequential set of tasks that a user defines. It is used to run, build, test and deploy the source code.

Types:

- Freestyle
- Pipeline
- Multi-configuration Project
- folder
- MultiBranch pipeline
- organisation folder

DEVOPS BY MUSTAFA

JENKINS SETUP:

STEP-1: get the links from jenkins.io
STEP-2: Install jenkins dependencies
amazon-linux-extras install java-openjdk11 -y
STEP-3: install jenkins
yum install jenkins -y

to check the jenkins status: systemctl status jenkins
STEP-4: start the jenkins
systemctl start jenkins

STEP-5: to connect with the jenkins
public_ip:8080

DEVOPS BY MUSTAFA

WEBHOOKS VS BUILD PERIODICALLY VS POLL SCM:

WEBHOOK:

whenever developer commit the code, it will automatically trigger the job. It doesn't need any specific time.
developer commit whenever he want, then automatically it will trigger.

POLL SCM:

When we change the code **in a particular time**, it will automatically trigger the job.

BUILD PERIODICALLY:

IT will gets trigger the job, even if the changes are made or not.

DEVOPS BY MUSTAFA

JENKINS PROJECT:

DEPLOY A WAR FILE IN TOMCAT SERVER

CLIENT REQUIREMENT: TO DEVLOPE A STATIC WEBSITE

PLAN:

1. CODE: HTML, CSS TO DEVELOPE THE APP
2. WE WILL USE AWS TO DEPLOY
3. TO AUTOMATE THE PROJECT WE NEED TO USE JENKINS
4. 2 DEVELOPERS
5. 24 HOURS
6. COST

CODE:

1. HTML & CSS IS USED TO DEVELOP A BASIC WEBSITE

BUILD:

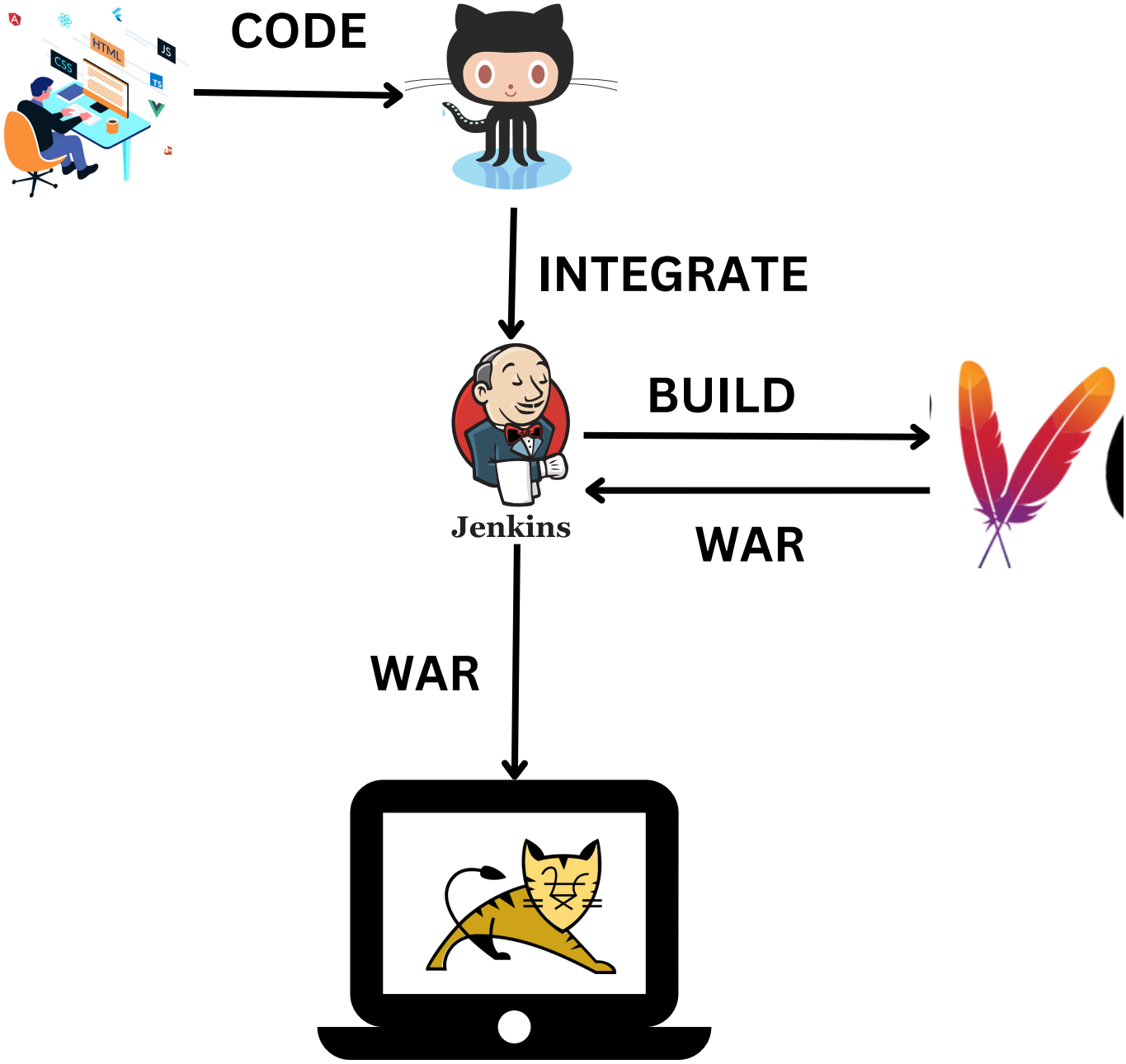
1. USE MAVEN TO BUILD THE SOURCE.
2. ONCE WE BUILD WE WILL GET A JAR/WAR
3. THESE FILES WE WILL USE TO TEST AND DEPLOY

TEST:

1. TESTING ENGINEERS WILL DO THE TESTING
2. SO WE WILL SKIP THIS PART.

DEPLOY:

1. TO DEPLOY THE WAR FILE WE NEED HAVE A CLIENT SERVER.
2. SETUP APACHE TOMCAT IN SERVER AND DEPLOY THE CODE IN SERVER.



REQUIREMENTS:

- LAUNCH 2 SERVER (jenkins, prod)
- GET THE CODE FROM THE DEVELOPERS
- SETUP JENKINS IN JENKINS SERVER
- SETUP TOMCAT IN PROD SERVER

PROCEDURE:

STEP-1: LAUNCH 2 INSTANCES WITH 8080 PORT

STEP-2: SETUP JENKINS IN SERVER

- 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
- amazon-linux-extras install java-openjdk11 -y
- yum install jenkins -y && systemctl restart jenkins

STEP-3: FORK THE GITHUB (https://github.com/devops0014/one.git)

STEP-4: INSTALL GIT IN OUR SERVER
yum install git -y

STEP-5: CREATE JOB AND INTEGRATE GIT TO JENKINS AND BUILD IT.
ONCE WE BUILD THE JOB, FILES PRESENT IN MASTER BRANCH WILL COMES INTO CI SERVER

STEP-6: NEXT STEP IS BUILD THE SOURCE CODE WHICH ARE PRESENT IN CI SERVER. TO BUILD THE WE NEED TO USE MAVEN

INSTALL JAVA-1.8.0 & MAVEN IN OUR SERVER

yum install java-1.8.0-openjdk -y
yum install maven -y

STEP-7: **CONFIGURE** THE SAME JOB AND CLICK ON **BUILD STEP** AND SELECT **ADD BUILD STEP** SELECT **invoke top level maven targer.**

in the goal : **clean package**

SAVE THE JOB AND BUILD.
SO WE WILL GET A WAR FILE IN TARGET FOLDER.

STEP-8: SETUP THE TOMCAT SERVER IN PROD SERVER.

DEVOPS BY MUSTAFA

- download tomcat file from dlcdn: wget https://dlcdn.apache.org/tomcat/tomcat-9/v9.0.70/bin/apache-tomcat-9.0.70.tar.gz
- untar the file: tar -zxvf apache-tomcat-9.0.70.tar.gz
- go to the folder: cd apache-tomcat-9.0.70//webapps/manager/META-INF
- open the context.xml in vim editor and make some change (delete 2 lines (21 and 22 lines))
- go to three steps back: cd ../../../
- and go to conf folder and open tomcat-user.xml file in vim editor

```
-->
<!--
  <role rolename="tomcat"/>
  <role rolename="role1"/>
  <user username="tomcat" password="<must-be-changed>" roles="tomcat"/>
  <user username="both" password="<must-be-changed>" roles="tomcat,role1"/>
  <user username="role1" password="<must-be-changed>" roles="role1"/>
-->
  <role rolename="manager-gui"/>
  <role rolename="manager-script"/>
  <user username="tomcat" password="123456" roles="manager-gui, manager-script"/>
</tomcat-users>
```

- go to one step back: cd ..
- go to bin folder and execute startup.sh file

./startup.sh

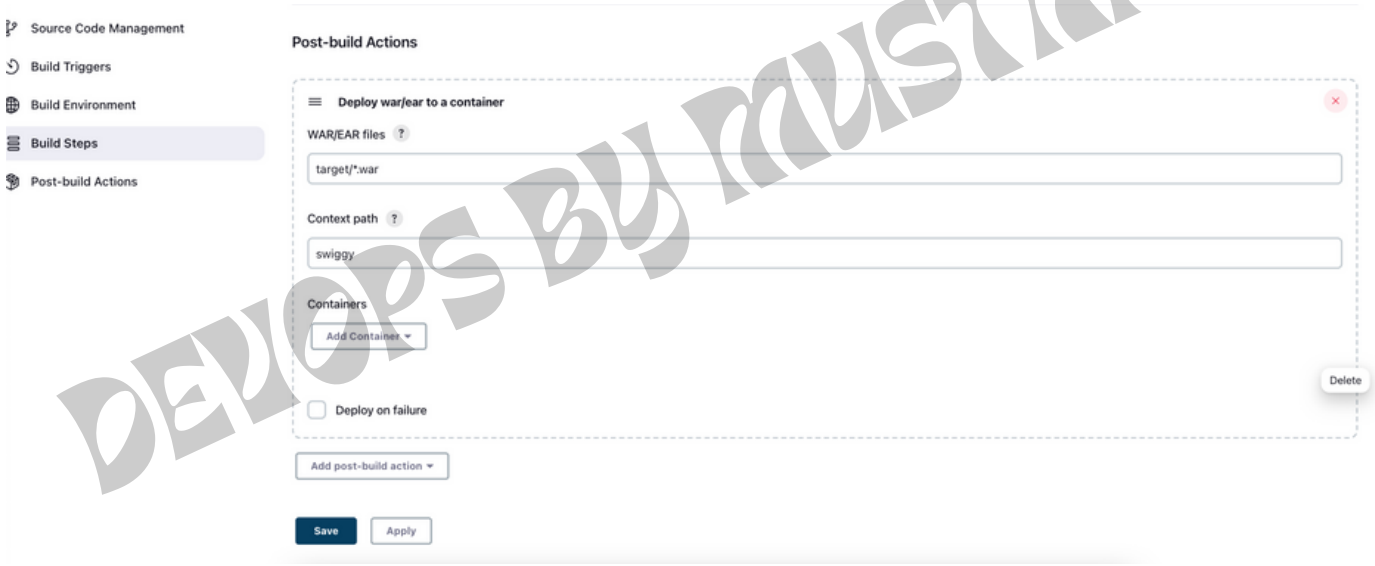
STEP-9:

GO to manager apps and it will ask the user name and password enter it

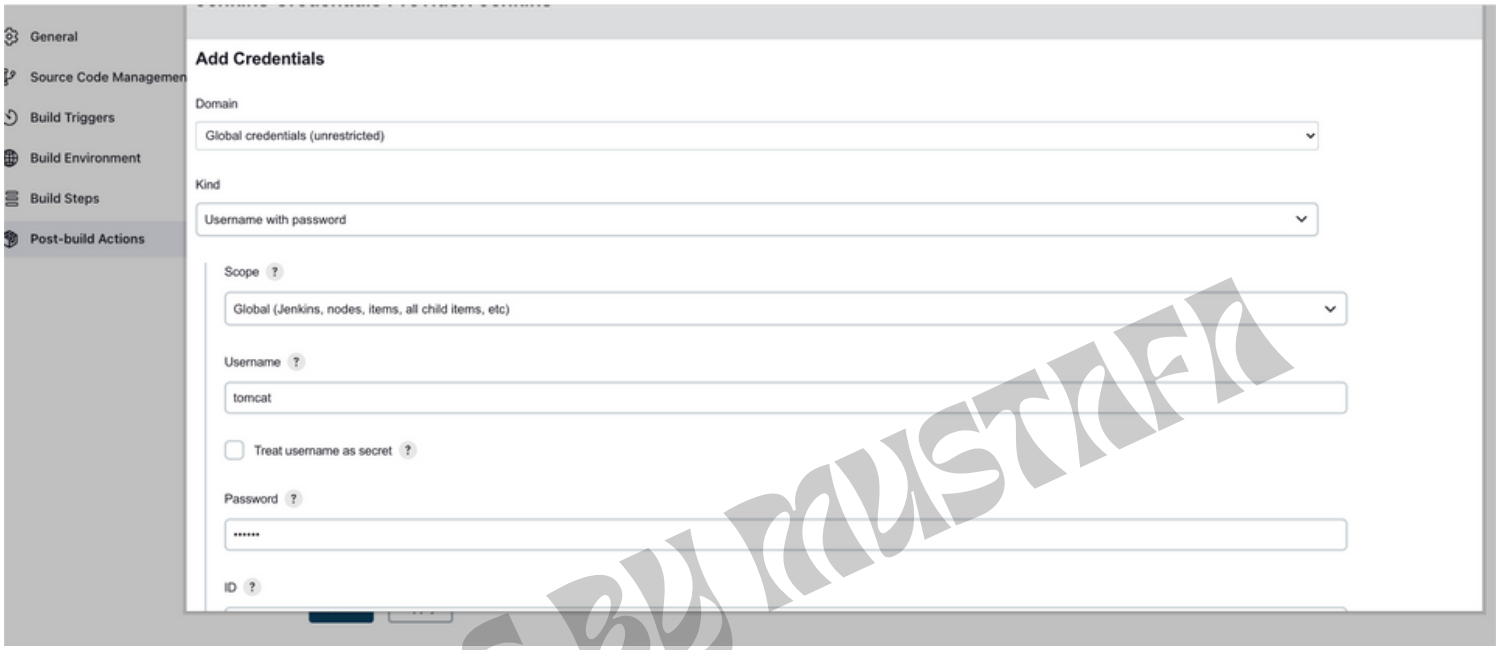
STEP-10: DEPLOY THE WAR FILE IN TOMCAT

- go to jenkins dashboard
- install plugin (manage jenkins --> manage plugin --> available plugin --> deploy to container
- after installing the plugin go to our job and select post build actions --> add post build actions

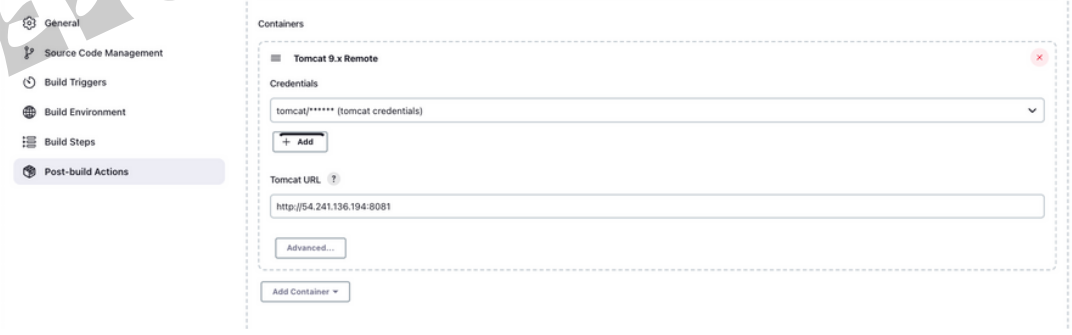
- select deploy war/ear to container



- click on add container(9th version). and add credentials (username & password of tomcat)



- add tomcat url



- save and build the job and go to tomcat

you will see swiggy folder. click on the folder you can access the client application

TASKS:

1. IMPLEMENT CI/CD IN ABOVE PROJECT
2. SETUP TOMCAT IN ANOTHER SERVER
3. PRACTICE BRANCH AND MERGE CONFLICTS IN GIT
4. UNDERSTAND THE PROJECT FOLDER STRUCTURE IN MAVEN
5. PRACTICE GIT WITH BITBUCKET
6. CREATE A PIPELINE WITH 4 JOBS

(JOB1 NEEDS TO CREATE FILE USING PARAMETER
JOB2 NEEDS TO CREATE FOLDER USING PARAMETER
JOB3 NEEDS TO GET THE SOURCE CODE FROM GITHUB
JOB4 NEEDS TO CREATE A FILE AND INSERT SOME DATA INTO FILE)

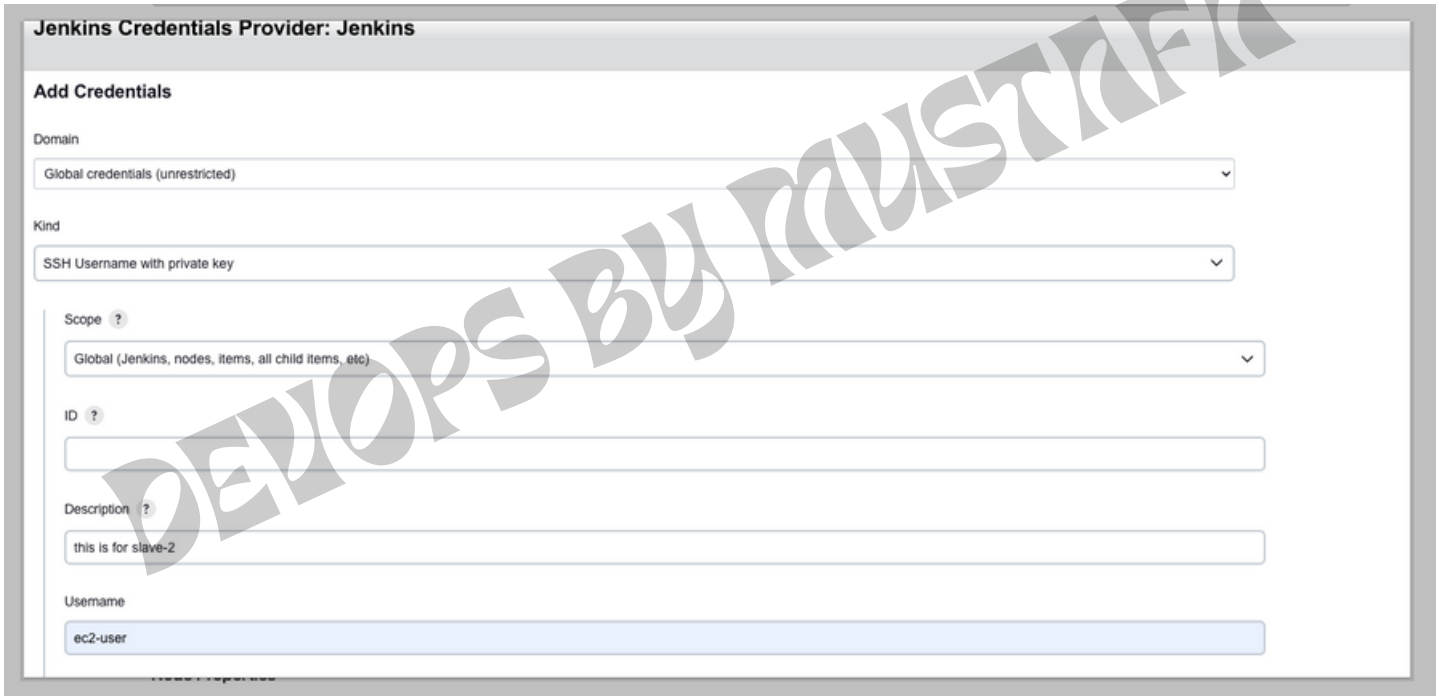
- CHANGE TOMCAT PORT NUMBER:
- to stop the tomcat: ./shutdown.sh
- go to conf folder and open server.xml file in vim editor
- change the port number in 69 line from 8080 to 8081
- and save & exit from the file
- go to one step back cd ..
- go to bin folder again and execute startup.sh file

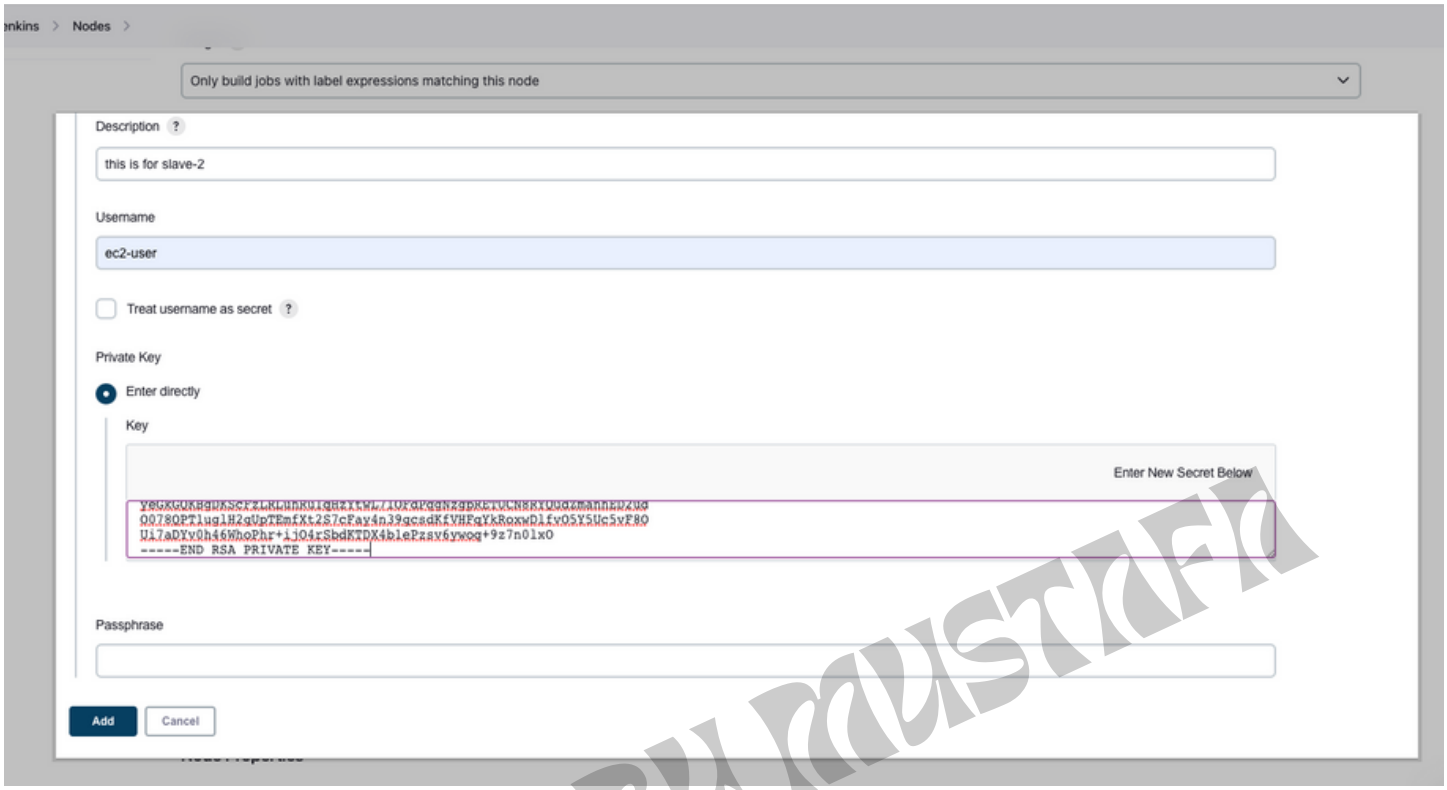
PUBLICIP:8081 (CONNECT IN THE BROWSER)

JENKINS MASTER - SLAVE

1. LAUNCH **3 INSTANCES** WITH **KEYPAIR** (PEM FILE)
    a. MASTER
    b. SLAVE-1
    c. SLAVE-3
2. SETUP **JENKINS IN MASTER**
3. install **JAVA11** on slaves (amazon-linux-extras install java-openjdk11 -y)
4. go to manage jenkins >> manage nodes and clouds >> new node --> give node name & select permanent agent

ADD CREDENTIALS

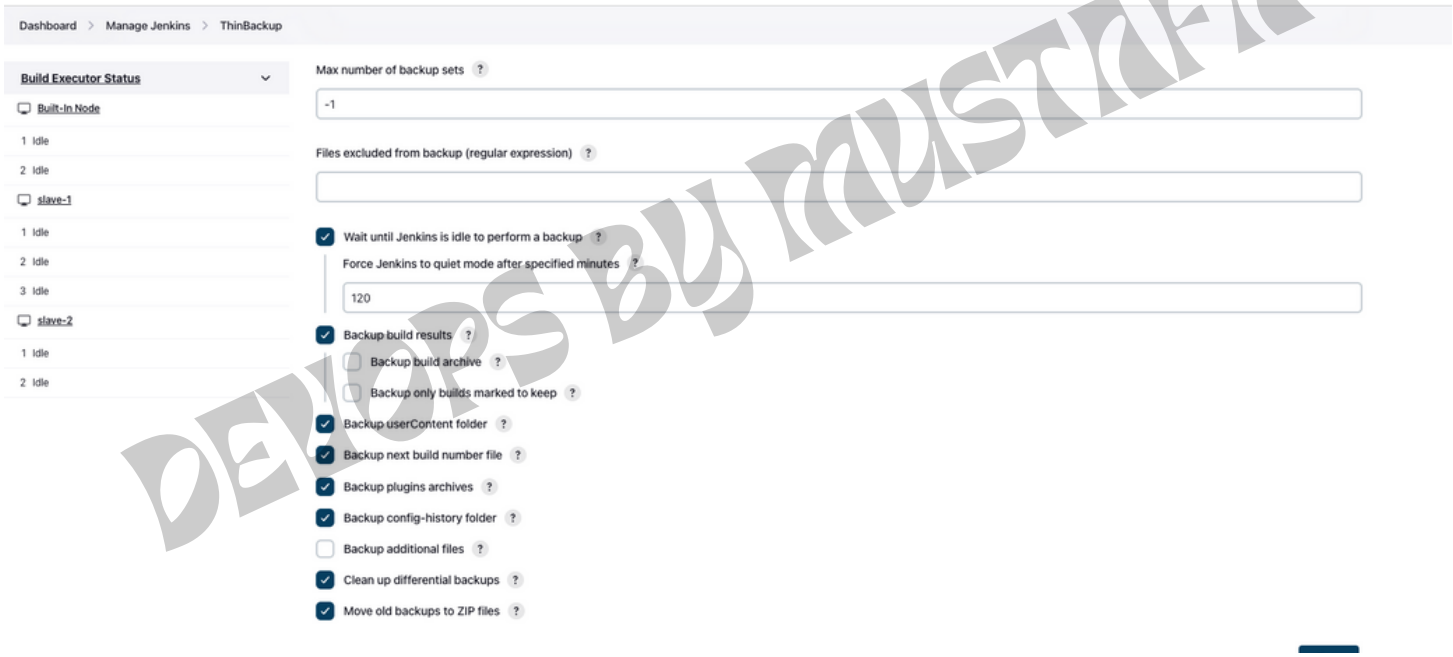




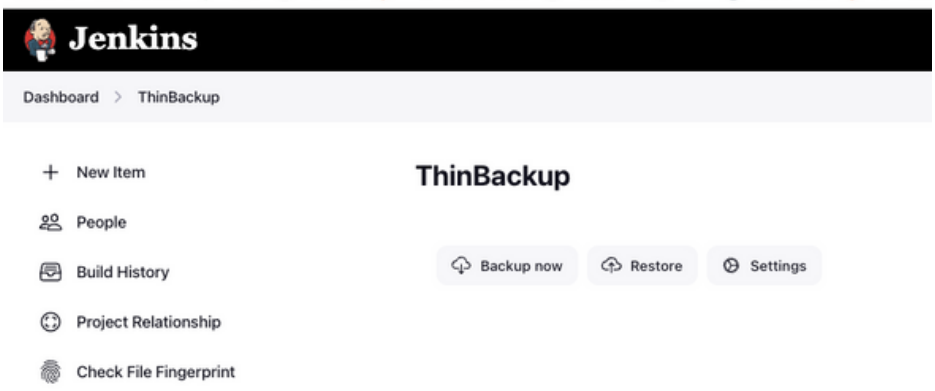
JENKINS BACKUP

Whatever we execute via Jenkins or perform any setting in Jenkins, it stores in the form of backup in Jenkins home directory. This backup includes data and configuration-related settings like **job configurations, plugins**, plugin configurations, build logs, etc. So, it's essential to take the backup of this prominent information to use this backup in the future if required.

1. INSTALL PLUGIN (THINBACKUP).
2. create a folder in opt directory: mkdir /opt/mybackup
3. change owners of the folder: chown jenkins:jenkins /opt/mybackup
4. give full permissions to the folder: chmod 777 /opt/mybackup
5. go to manage jenkins and select thinbackup and click on settings.

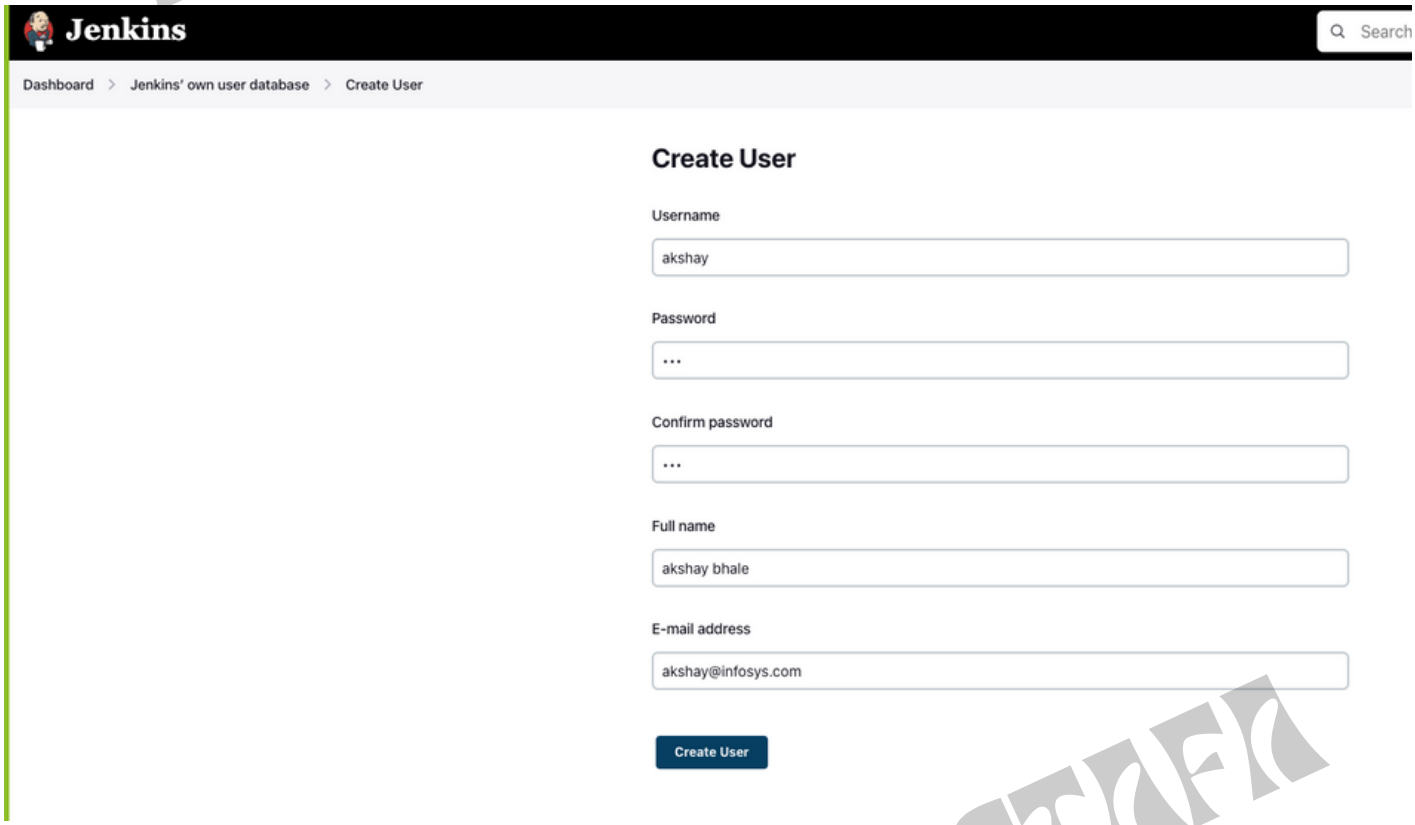


click on **backup now**
go and check in **/opt/mybackup folder**, you will get all the config files of jenkins in our folder.
if you want to restore those files, click on restore option



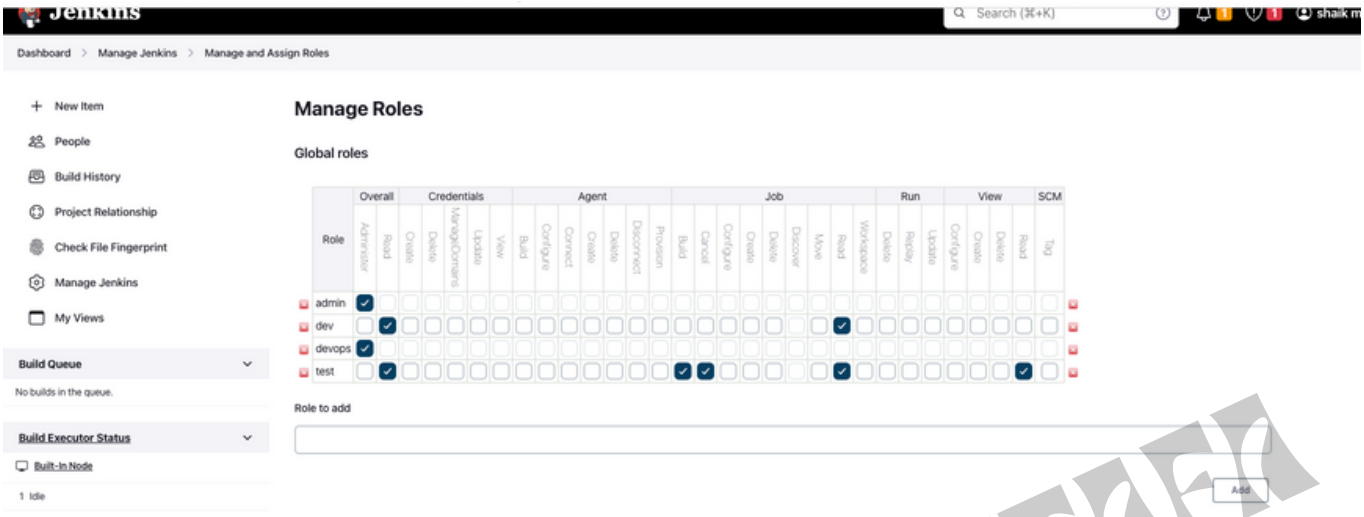
jenkins user management:

1. INSTALL PLUGIN (role-based authorization strategy). INSTALL WITHOUT RESTART & RESTART JENKINS
2. go to manage jenkins and select manage users.
3. create user

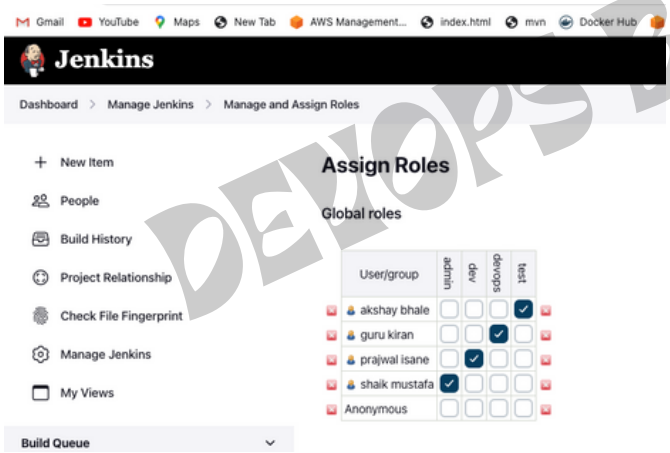


4. Go to manage jenkins >> config global security and change the authorization to role based strategy and save it

4. Go to manage jenkins >> manage and assign roles and select manage roles.



5. Go to assign roles



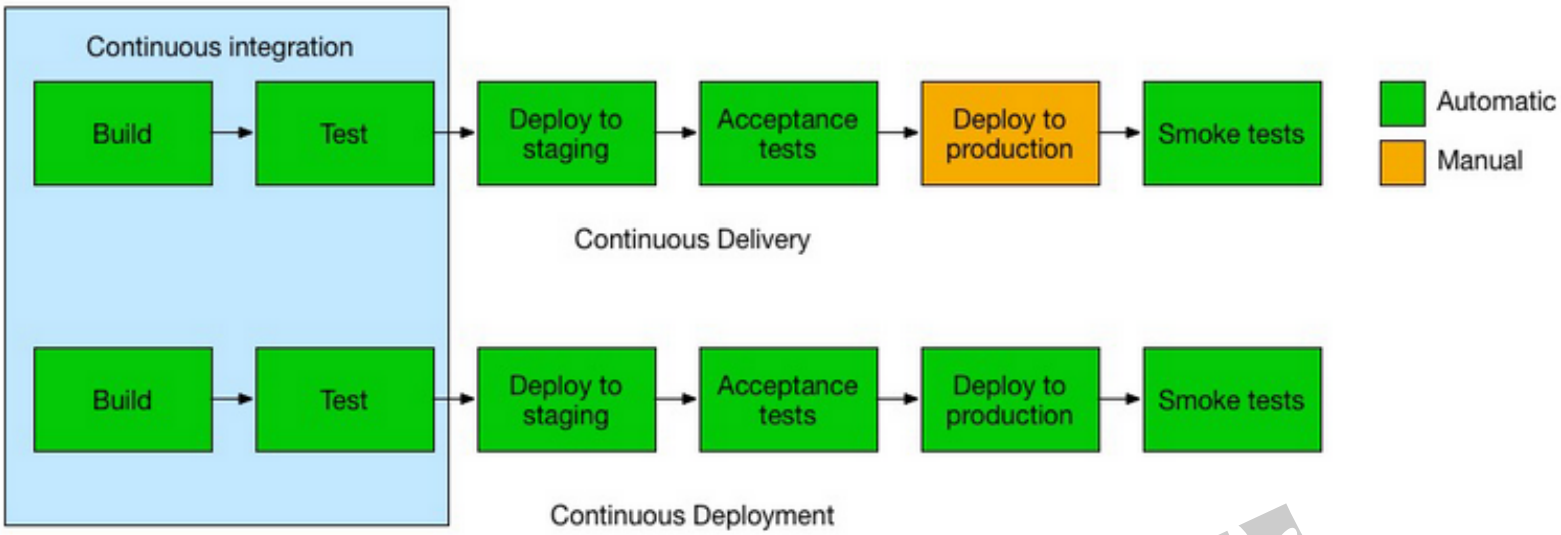
WHAT IS CI?

Continuous Integration is the combination of continuous build and continuous test.

WHAT IS CD?

Continuous delivery: we have to do the manual deployment in the production server.

Continuous deployment: here, automation deployment will be done on the production server.



WHAT IS JENKINS PIPELINE?

Jenkins pipeline is a group of jobs that are interlinked in a particular sequence. It has set of plugins that is used to implement CI /CD in jenkins.

Every job on jenkins will have one or more events like code, build, Test and Deploy etc..

WHAT IS JENKINS FILE?

If we are writing the entire jenkins pipeline in a text format. It contains the steps that are required for running the jenkins pipeline.

It is used to create a pipeline for **build and deploy** the code. This Jenkins file uses **groovy syntax**.

These jenkins file will written in 2 ways

1. Declarative pipeline
2. Scripted Pipeline

Declarative pipeline: It is a **recent feature** of the jenkins pipeline which helps us to write the pipeline in a easier way. It will starts with the word **pipeline**

Scripted pipeline: It is a **traditional way** of writing a jenkins pipeline as a code. It starts with the word **node**

DEVOPS BY MUSTAFA

SINGLE STAGE PIPELINE:

```
pipeline {
    agent any
    stages {
        stage ("stage-1") {
            steps {
                echo "hai this is my first stage"
            }
        }
    }
}
```

pipeline : A Pipeline is a user-defined model of a CD pipeline. A Pipeline's code defines your entire build process, which typically includes stages for building an application, testing it and then delivering it.

agent : it defines that, in which server the pipeline will gets executes.

stage : stage in jenkins file contains of unique tasks such as build, test, Deploy etc..

step : It tells the jenkins what exactly needs to be done. ex: executing a build command or linux command etc..

DEVOPS BY MUSTAFA

MULTI-STAGE PIPELINE:

```
pipeline {
    agent any
    stages {
        stage ("ashokit") {
            steps {
                echo "hello ashok it"
            }
        }
        stage ("devops") {
            steps {
                echo "we are learning devops"
            }
        }
        stage ("aws") {
            steps {
                echo "we are learning aws also"
            }
        }
    }
}
```


DEVOPS BY MUSTAFA

PIPELINE AS A CODE:

YOU CAN RUN COMMANDS HERE

```
pipeline {
  agent any

  stages {
    stage('CMD') {
      steps {
        sh 'touch file1'
        sh 'pwd'
      }
    }
  }
}
```

MULTIPLE COMMANDS OVER SINGLE LINE:

```
pipeline {
  agent any

  stages {
    stage('CMD') {
      steps {
        sh '''
        touch file2
        pwd
        date
        whoami
        '''
      }
    }
  }
}
```


DEVOPS BY MUSTAFA

ENVIRONMENT VARIABLES:

```
  pipeline {
    agent any
    environment {
       name = 'raham'
    }
    stages {
       stage('ENV') {
          steps {
                echo "hai my name is $name"
          }
       }
    }
  }
```

```
 pipeline {
   agent any
   environment {
      name = 'raham'
   }
   stages {
      stage('ENV1') {
         steps {

            sh 'echo "${name}"'
         }
      }
      stage('ENV2') {
         environment {
            name = 'shaik'
         }
         steps {

            sh 'echo "${name}"'
         }
      }
 }
}
```


DEVOPS BY MUSTAFA

PIPELINE TO GET A SOURCE CODE TO DEV SERVER:

```
pipeline {
    agent {
        node {
            label 'dev'
        }
    }
    stages {

        stage ("git") {
            steps {
                git "https://github.com/devops0014/devoprepoforpractice.git"
            }
        }
    }
}
```

CREATE A PIPELINE:

1 - CREATE A FILE
2 - INSERT SOME DATA INTO A FILE
3 - CREATE A FOLDER
4 - COPY THE FILE INTO A FOLDER
5 - PRINT YOUR NAME USING VARIABLE

DEVOPS BY MUSTAFA

PIPELINE WITH PARAMETERS:

STRING

```
pipeline {
    agent any
    parameters {
        string (name: "aws", defaultValue: "EC2", description: "i am unsing aws cloud")
    }
    stages {
        stage ("stage-1") {
            steps {
                echo "hai i am using parameters"
            }
        }
    }
}
```

BOOLEAN:

```
pipeline {
    agent any
    parameters {
        booleanParam (name: "jenkins", defaultValue: true, description: "")
    }

    stages {
        stage('Hello') {
            steps {
                echo ' i am using boolean parameter'
            }
        }
    }
}
```


DEVOPS BY MUSTAFA

CHOICE:

```
pipeline {
    agent any
    parameters {
        choice (name: "branch", choices : ["one", "two", "three", "four"], description: "this
is cp")
    }
    stages {
        stage('Hello') {
            steps {
                echo 'Hello World'
            }
        }
    }
}
```

PIPELINE WITH POST BUILD ACTIONS:

ALWAYS:

```
pipeline {
    agent any

    stages {
        stage('Hello') {
            steps {
                echo 'Hello World
            }
        }
    }
    post{
        always {
                echo 'THIS WILL BE PRINTED ANYWAY'
            }
        }
}
```


DEVOPS BY MUSTAFA

SUCCESS:

```
pipeline {
  agent any

  stages {
    stage('Hello') {
      steps {
        echo 'Hello World
      }
    }
  }
  post{
    success {
        echo 'THIS WILL BE PRINTED IF THE BUILD GETS SUCCESS'
      }
    }
}
```

FAILURE:

```
pipeline {
  agent any

  stages {
    stage('Hello') {
      steps {
        echo 'Hello World
      }
    }
  }
  post{
    failure {
        echo 'THIS WILL BE PRINTED EVEN IF THE BUILD GETS FAILURE'
      }
    }
}
```


DEVOPS BY MUSTAFA

SUCCESS, ALWAYS, FAILURE:

```
pipeline {
   agent any

   stages {
      stage('Hello') {
         steps {
            ech 'Hello World'
         }
      }
   }
   post {
      always {
         echo "this will gets printed anyway"
      }
      success {
         echo "this will be printed if the build gets success"
      }
      failure {
         echo "this will be printed even if the build gets failure"
      }
   }
}
```

INPUT IN JENKINS FILE:

```
pipeline {
   agent any

   stages {
      stage('deploy') {
         input {
            message "can i deploy"
            ok "yes you can"
         }
         steps {
            echo 'our code is deployed'
         }
      }
   }
}
```

JENKINS FILE TO DEPLOY THE CODE IN TOMCAT SERVER:

```
pipeline {
    agent any
    stages {
        stage ("code") {
            steps {
                git "https://github.com/devops0014/myweb-01.git"
            }
        }
        stage ("build") {
            steps {
                sh 'mvn clean package'
            }
        }
        stage ("deploy") {
            steps {
                sh "cp /var/lib/jenkins/workspace/pipeline-1/target/*.war
/opt/apache-tomcat-9.0.71/webapps"
            }
        }
    }
}
```


STEPS:

1. LAUNCH AN **INSTANCE** WITH **8080 PORT** AND **PEM FILE.**
2. SETUP **JENKINS**
3. SETUP **TOMCAT IN /opt FOLDER**. (change the **tomcat port number**).
4. CHANGE THE **OWNER OF THE TOMCAT FOLDER** (**chown -R jenkins:jenkins /opt/apache-tomcat-9.0.71/webapps/**)
5. INSTALL GIT, JAVA-1.8.0, MAVEN IN OUR SERVER
6. CREATE A PIPELINE JOB AND ADD THE ABOVE CODE.

DEVOPS BY MUSTAFA

TOMCAT SETUP:

- download tomcat file from dlcdn: wget https://dlcdn.apache.org/tomcat/tomcat-9/v9.0.70/bin/apache-tomcat-9.0.70.tar.gz
- untar the file: tar -zxvf apache-tomcat-9.0.70.tar.gz
- go to the folder: cd apache-tomcat-9.0.70//webapps/manager/META-INF
- open the context.xml in vim editor and make some change (delete 2 lines (21 and 22 lines))
- go to three steps back: cd ../../../
- and go to conf folder and open tomcat-user.xml file in vim editor

```
-->
<!--
  <role rolename="tomcat"/>
  <role rolename="role1"/>
  <user username="tomcat" password="<must-be-changed>" roles="tomcat"/>
  <user username="both" password="<must-be-changed>" roles="tomcat,role1"/>
  <user username="role1" password="<must-be-changed>" roles="role1"/>
-->
  <role rolename="manager-gui"/>
  <role rolename="manager-script"/>
  <user username="tomcat" password="123456" roles="manager-gui, manager-script"/>
</tomcat-users>
```

- go to one step back: cd ..
- go to bin folder and execute startup.sh file

./startup.sh

note: tomcat will not run in 8080 port now. because jenkins is already running in 8080 port

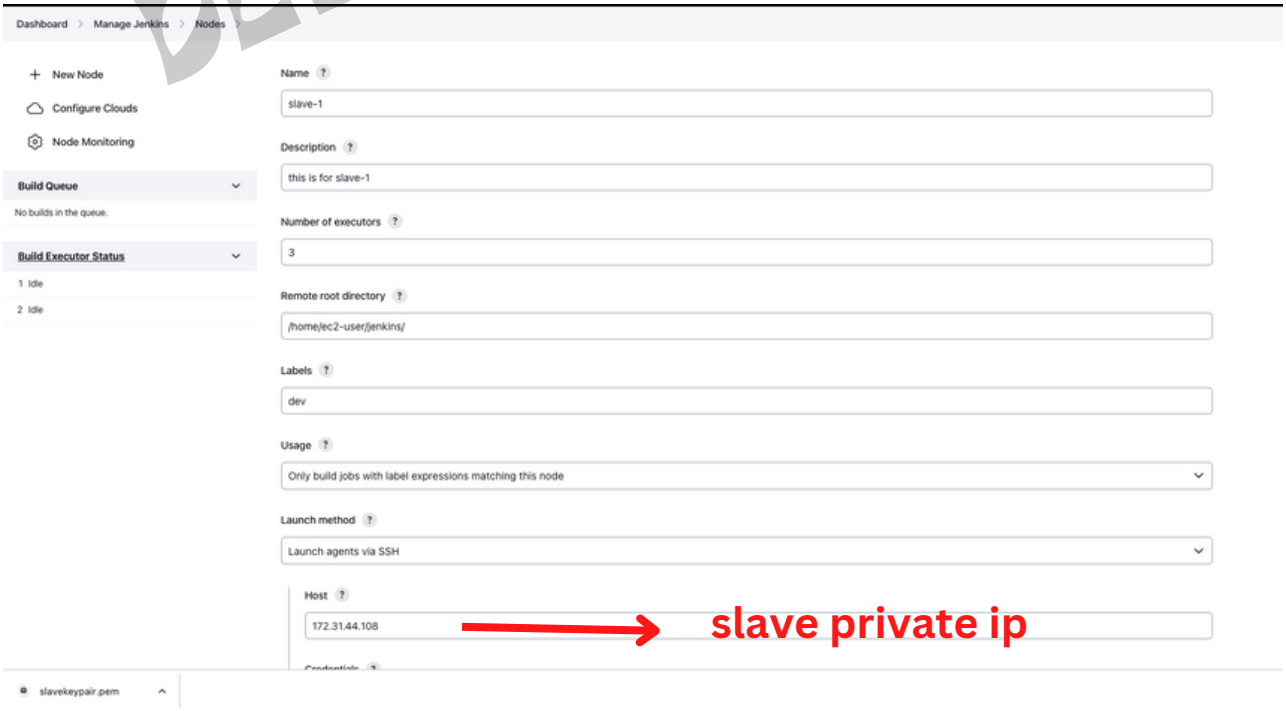
- CHANGE TOMCAT PORT NUMBER:
- to stop the tomcat: ./shutdown.sh
- go to conf folder and open server.xml file in vim editor
- change the port number in 69 line from 8080 to 8081
- and save & exit from the file
- go to one step back cd ..
- go to bin folder again and execute startup.sh file

PUBLICIP:8081 (CONNECT IN THE BROWSER)

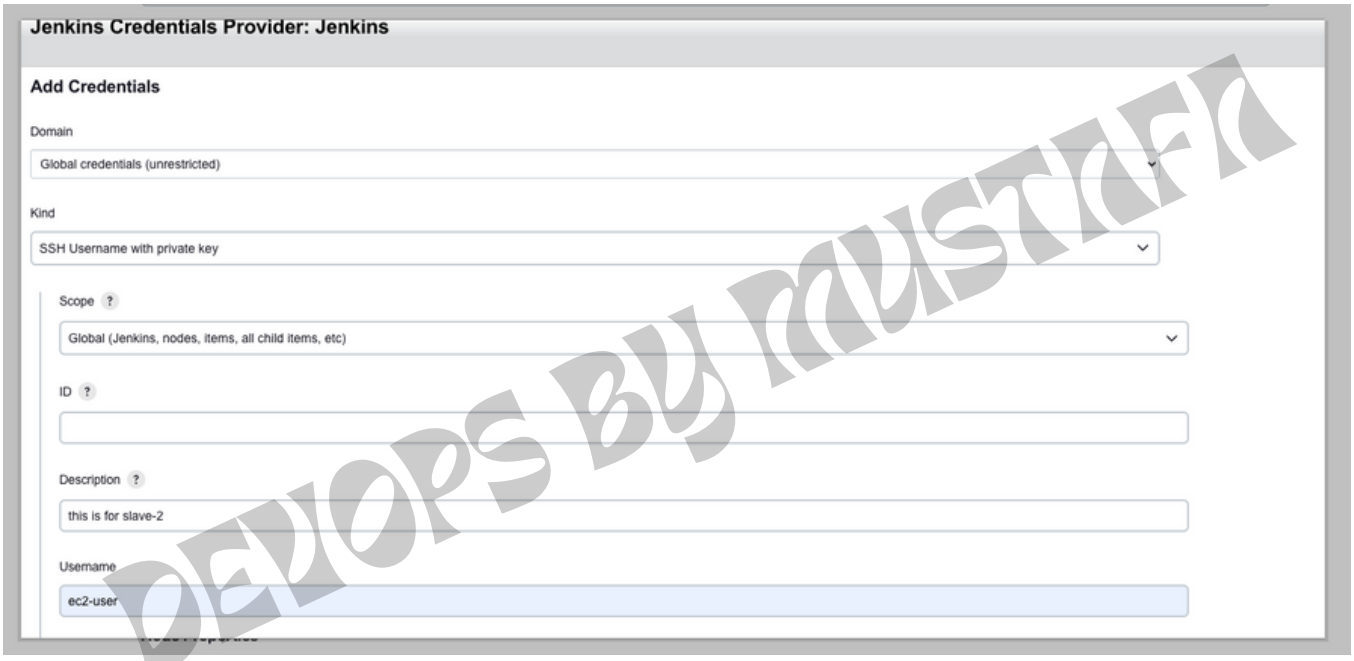
DEVOPS BY MUSTAFA

JENKINS JOB TO DEPLOY A WEBAPP IN CLIENT SERVER:

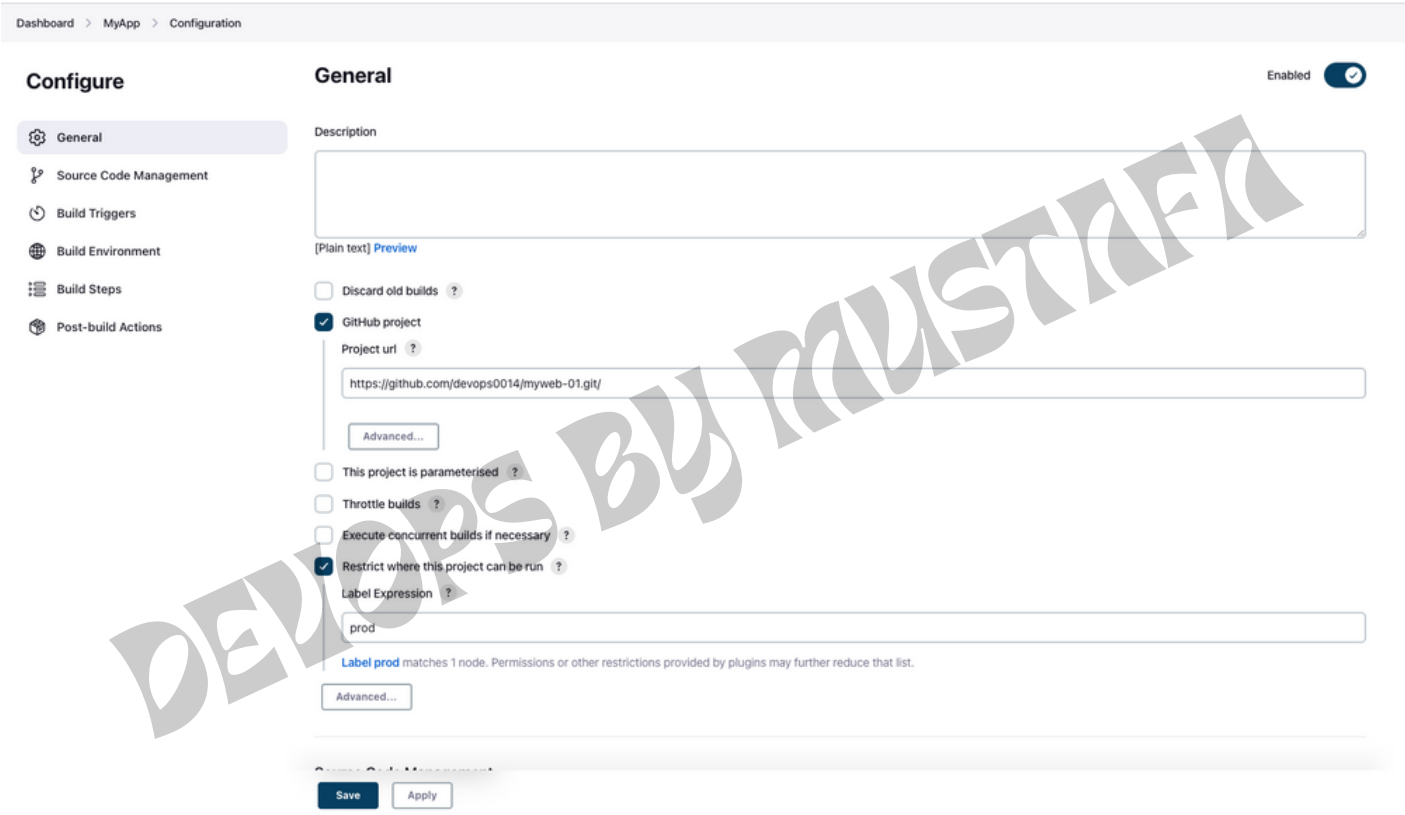
1. LAUNCH AN **TWO INSTANCES** WITH **8080 PORT** AND **PEM FILE.** (JENKINS & PROD SERVER)
2. SETUP **JENKINS IN JENKINS SERVER**
3. INSTALL **JAVA-11 IN PROD SERVER**
4. CEATE A NODE : go to manage jenkins >> manage nodes and clouds >> new node --> give node name & select permanent agent.

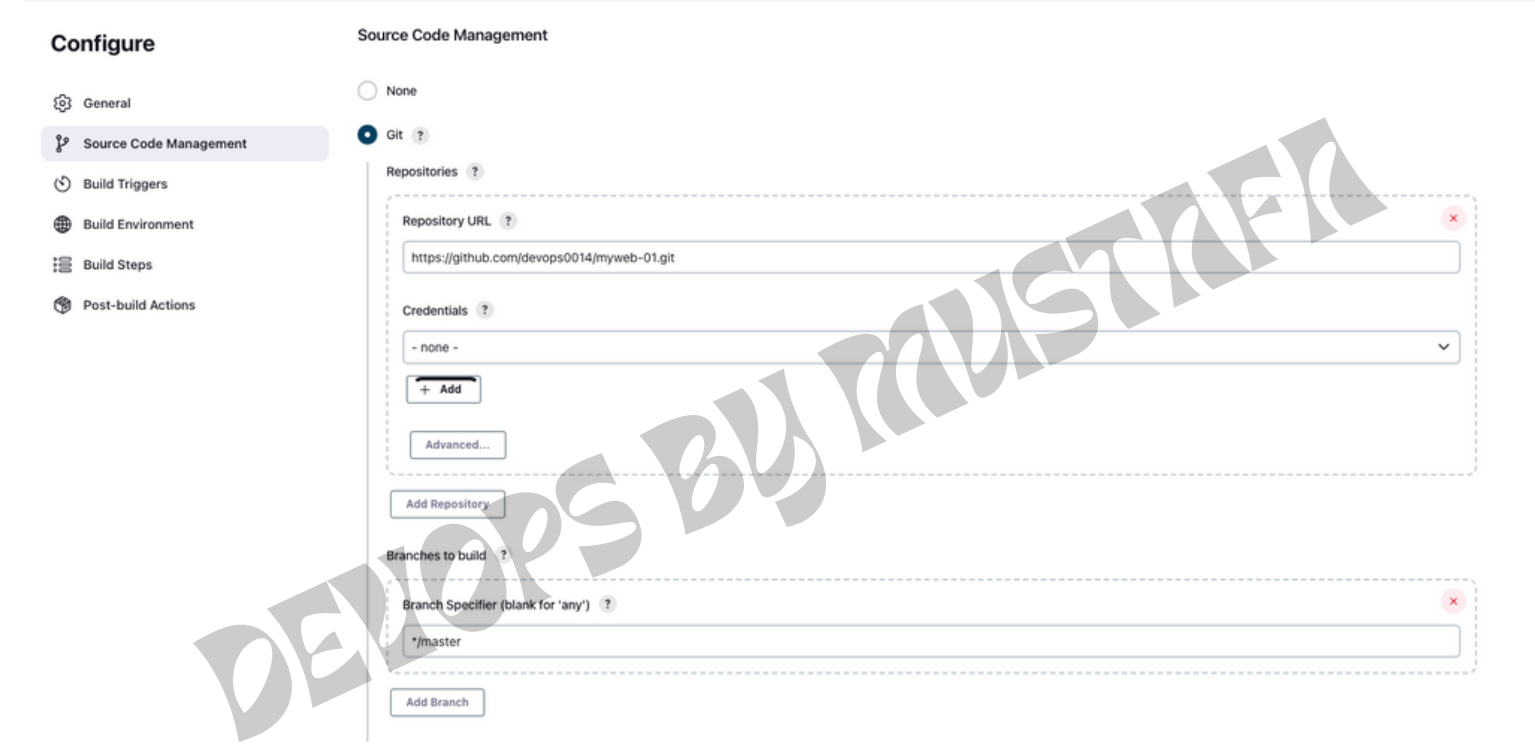


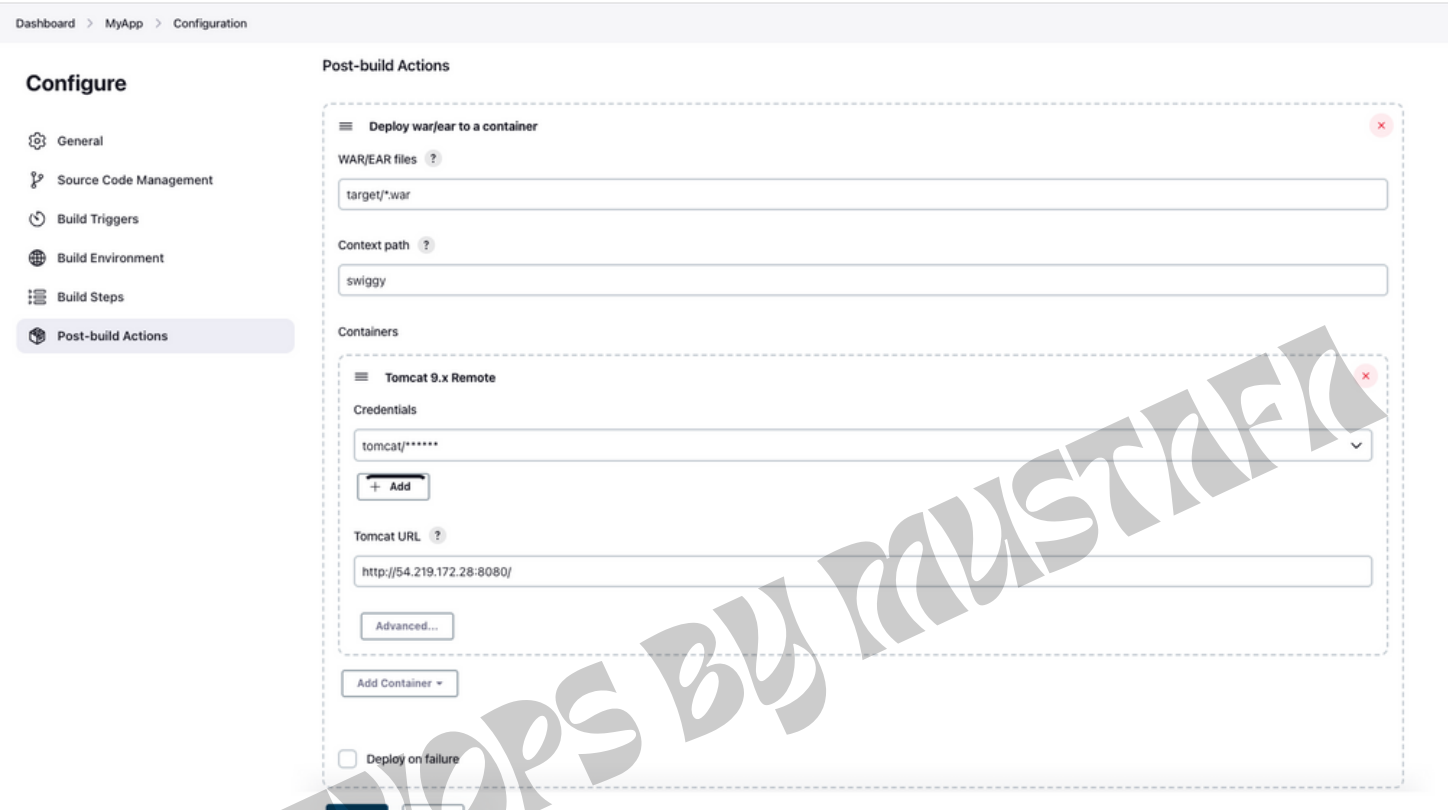
ADD CREDENTIALS



5. SETUP THE TOMCAT IN PROD SERVER (dont need to change the port number).
6. INSTALL PLUGIN: go to manage jenkins >> manage plugins >> available plugin >> deploy to container.
7. CREATE A JOB IN FREE STLE:







SAVE & BUILD

JENKINS JOB TO DEPLOY A WEBAPP IN CLIENT SERVER USING JENKINS PIPELINE:

1. launch 2 instances (jenkins & client) with 8080 and with keypair
2. setup jenkins in JENKINS SERVER
3. setup tomcat in CLIENT SERVER

ANSIBLE:

- is an open source software that automates software provisioning, configuration management, and application deployment.
- Orchestration, Security and compliance.
- Uses YAML Scripting language which works on KEY-VALUE PAIR
- Ansible GUI is called as Ansible Tower. It was just Drag and Drop.

The Keys Features of Ansible:

Agentless

There is no software or agent to be installed on the client that communicates back to the server.

Simple and extensible:

Ansible is written in Python and uses YAML for playbook language, both of which are considered relatively easy to learn.

PLAYBOOK:

Ansible playbooks are a way to send commands to remote computers in a scripted way. Instead of using Ansible commands individually to remotely configure computers from the command line, you can configure entire complex environments by passing a script to one or more systems.

WHY ANSIBLE:

While managing the multiple servers its hard to keep their configuration identical. If you have multiple servers which needs to configure the same setup in
all. while doing the one to one server their might be a chances to miss some configuration steps in some servers.

Thats why automation tools come into play! The automation tools like Ansible, Chef,
Puppet and SaltStack all are based on a same principle.

DESCRIBE THE DESIRED STATE OF THE SYSTEM

NORMAL PLAYBOOK

```
---
- hosts: remo
  user: ansible
  become: yes
  connection: ssh
  tasks:
    - name: installing git
      action: yum name=git state=present
```

TO EXECUTE: ansible-playbook playbook.yml

VARIABLES:

Ansible also provides various ways of setting variables. They are used to store values that can be later used in the playbook.

Variable names in Ansible should start with a letter. The variable can have letter, numbers and underscore. Invalid variable declaration comes when we use dot (.), a hyphen (-), a number or variable separated by mutlple words.

```
--- #VARIABLES
- hosts: remo
  user: ansible
  become: yes
  connection: ssh
  vars:
    pkgname: httpd
  tasks:
    - name: installing httpd
      action: yum name='{{pkgname}}' state=present
```

other way of passing arguments is by passing them to the command line while running using the –extra-vars parameter.

```
---
- hosts: dev
  user: ansible
  become: yes
  connection: ssh

  tasks:
    - name: install git
      action: yum name='{{abc}}' state=present
~
```

for single var: ansible-playbook one.yml --extra-vars "abc=git"

for multiple vars: ansible-playbook one.yml --extra-vars "abc=git def=maven"

Passing a Varaible file - A Varaible can be defined in a variable file and can be passed to a playbook using the **include**

```
---
  - set_fact: abc=httpd
  - name: install Apache
    yum: name=httpd state=present
~
~
```

one.yml

two.yml

```
---
- hosts: dev
  become: yes
  tasks:
    - include: one.yml
    - name: install git
      service: name='{{abc}}' state=restarted
~
```

HANDLERS:

Handler is same as task but it will run when called by another task. (OR)
It will run if the task contains a notify directive and also indicates that it changed something.

```
--- # HANDLER
- hosts: remo
  user: ansible
  become: yes
  connection: ssh
  tasks:
        - name: install httpd server on centos
          action: yum name=httpd state=installed
          notify: restart httpd
  handlers:
          - name: restart httpd
            action: service name=httpd state=restarted
~
```

CONDITIONS:

Sometimes while writing the playbook we never think about dividing the playbook logically and we end up with a long playbook. But what if we can divide the long playbook into logical units. We can achieve this with tags, which divides the playbook into logical sections.
Ansible Tags are supported by tasks and play. You can use tags keyword and provide any tag to task or play. Tags are inherited down to the dependency chain which means that if you applied the tags to a role or play, all tasks associated under that will also get the same tag.

If we have different scenarios, then we apply conditions according to the scenarios.
WHEN STATEMENT
Sometimes we want to skip a particular command on a particular node.

```
--- # CONDITIONS
- hosts: remo
  user: ansible
  become: yes
  connection: ssh
  tasks:
          - name: Install apache server for debian family
            command: apt-get-y install apache2
            when: ansible_os_family== "Debian"
          - name: install apache server for redhat family
            command: yum install httpd -y
            when: ansible_os_family== "RedHat"
```

TAGS:

If you have a large playbook, it may be useful to run only specific parts of it instead of running the entire playbook. You can do this with Ansible tags. Using tags to execute or skip selected tasks

```
---
- hosts: remo
  user: ansible
  become: yes
  connection: ssh
  tasks:
    - name: installing git
      action: yum name=git state=present
      tags: install
    - name: uninstalling git
      action: yum name=git state=absent
      tags: uninstall
```

TO EXECUTE A SINGLE TASK: ansible-playbook abc.yml --tags tagname
TO EXECUTE A MULTIPLE TASK: ansible-playbook abc.yml --tags tagname1,tagname2
TO SKIP A TASK: ansible-playbook abc.yml --skip-tags “uninstall”

LOOPS:

Ansible loop includes changing ownership on several files & directories with file module, creating multiple users with user modules and repeating a polling step until result reached.

```
--- # LOOPS
- hosts: remo
  user: ansible
  become: yes
  connection: ssh
  tasks:
          - name: add list of users in my nodes
            user: name='{{item}}' state=present
            with_items:
                    - raham
                    - mustafa
                    - shafi
                    - nazeer
```

PLAYBOOK TO CREATE A FILE/FOLDER:

```
---
- hosts: dev
  user: ansible
  become: yes
  connection: ssh

  tasks:
    - name: creating a file
      file:
        path: "jenkins.txt"
        state: touch
~
```

```
---
- hosts: dev
  user: ansible
  become: yes
  connection: ssh

  tasks:
    - name: creating a file
      file:
        path: "folder"
        state: directory
~
```

TO ENTER A DATA IN A FILE:

```
---
- hosts: dev
  tasks:
    - name: inserting a data in a file
      copy:
        dest: "devops.txt"
        content: |
          hi this is devops file
          we are inserting the data ij a file
          using ansible playbook
~
```


PLAYBOOK TO CHANGE PERMISSIONS TO A FILE:

```
---
- hosts: dev
  tasks:
    - name: change permissions to a file
      file:
        path: "devops.txt"
        state: touch
        mode: 777
```

PLAYBOOK TO DEPLOY A WEBAPP:

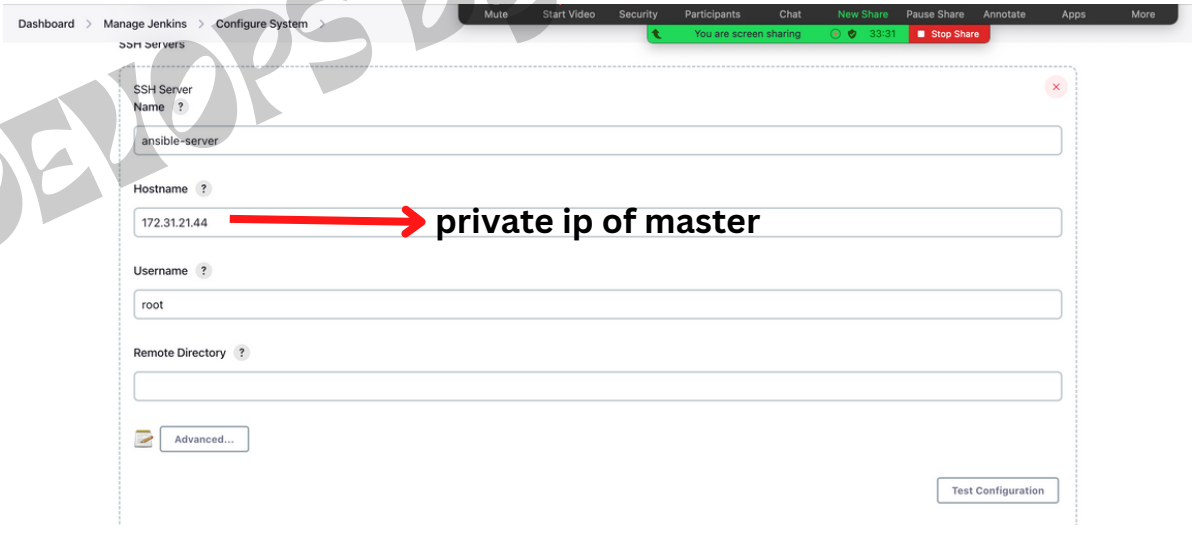
```yaml
---
- hosts: dev
  user: ansible
  become: yes
  connection: ssh

  tasks:
    - name: install httpd
      action: yum name=httpd state=present

    - name: restart httpd
      service: name=httpd state=restarted

    - name: create a file
      file:
        path: "/var/www/html/index.html"
        state: touch

    - name: enter data in a file
      copy:
        dest: "/var/www/html/index.html"
        content: |
          <h1>this is my webapplication, i have deployed using ansible </h1>
~
```

PLAYBOOK TO SETUP A JENKINS

```yaml
---
- hosts: localhost
  connection: ssh


  tasks:
    - name: getting links from jenkins.io
      get_url:
        url: https://pkg.jenkins.io/redhat-stable/jenkins.repo
        dest: /etc/yum.repos.d/jenkins.repo

    - name: import key from jenkins.io
      ansible.builtin.rpm_key:
        state: present
        key: https://pkg.jenkins.io/redhat-stable/jenkins.io.key

    - name: install java-11
      command: amazon-linux-extras install java-openjdk11 -y

    - name: install jenkins
      action: yum name=jenkins state=present

    - name: restart jenkins
      service: name=jenkins state=restarted
```


INTEGRATION OF ANSIBLE WITH JENKINS

- INSTALL PLUGIN: PUBLISH OVER SSH
- GO TO MANAGE JENKINS >> CONFIGURE SYSTEM >> SSH Server >> ADD Server



PLAYBOOK TO SETUP A TOMCAT

```yaml
---
- hosts: ops
  connection: ssh

  tasks:
    - name: gettling link
      get_url:
        url: https://dlcdn.apache.org/tomcat/tomcat-9/v9.0.71/bin/apache-tomcat-9.0.71.tar.gz
        dest: "/root/"

    - name: untar file
      command: tar -zxvf apache-tomcat-9.0.71.tar.gz

    - name: rename the file
      command: mv apache-tomcat-9.0.71 tomcat

    - name: java11
      command: amazon-linux-extras install java-openjdk11 -y

    - name: context.xml file
      template:
        src: context.xml
        dest: "/root/tomcat/webapps/manager/META-INF/context.xml"

    - name: add credits
      template:
        src: tomcat-users.xml
        dest: "/root/tomcat/conf/tomcat-users.xml"


    - name: start tomcat
      shell: nohup ./tomcat/bin/startup.sh
```


PLAYBOOK TO COPY FILE FROM MASTER TO SLAVE

```yaml
---
- hosts: dev
  connection: ssh


  tasks:
    - name: copy files from master to slave
      copy:
        src: jenkins.yml
        dest: jenkins.yml
```

ADHOC COMMANDS:

These commands can be run individually to perform Quick functions.
Not used for configuration management and deployment, bcz the cmds are one time usage.
The ansible ad-hoc cmds uses /usr/bin/ansible/ command line tool to automate single task.
Go to ansible server and switch to ansible server

ansible remo -a “ls” [remo: Group name, -a: argument, ls: command]
ansible remo [0] -a “touch file1”
ansible all -a “touch file2”
ansible remo -a “sudo yum install httpd -y”
ansible remo -ba “yum install httpd -y” (b: become you will become sudo user)
ansible remo -ba “yum remove httpd -y”

MODULES:

Ansible ships with number of modules (called library modules) that can be executed directly to remote hosts or playbooks.
Your library of modules can reside on any machine, and there are no servers, daemons or database required.
The default location for the inventory file is /etc/ansible/hosts
Go to ansible server and switch to ansible server

ansible remo -b -m yum -a “pkg=httpd state=present” (install: present)
ansible remo -b -m yum -a “pkg=httpd state=latest” (update: latest)
ansible remo -b -m yum -a “pkg=httpd state=absent” (uninstall: absent)
ansible remo -b -m service -a “name=httpd state=started” (started: start)
ansible remo -b -m user -a “name=raj” (to check go to that servers and sudo cat /etc/passwd).
ansible remo -b -m copy -a “src=filename dest=/tmp” (to check go to that server and give ls /tmp).

DEVOPS BY MUSTAFA

PLAYBOOK TO GET CODE FROM GIT (PUBLIC REPO):

```
---
- hosts: localhost
  become: yes
  tasks:
    - name: getting code from git
      git:
        repo: "https://github.com/devops0014/pubg.git"
        dest: "/home/mycode"
~
~
```


PLAYBOOK TO GET CODE FROM GIT (PRIVATE REPO):

```
---
- hosts: localhost
  become: yes

  tasks:
    - name: link
      git:
         repo: 'https://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@github.com/devops0014/ansible.git'
         dest: "/home/mygitcode"
~
~
```

SYNTAX: TOKEN@GITHUB.COM/USERNAME/REPO.GIT

HOST PATTREN:

'all' patterns refer to all the machines in an inventory.

ansible all-list-hosts ansible <groupname[remo]> --list-hosts

ansible <groupname> [remo][0] --list-hosts

groupname [0] – picks first machine of group

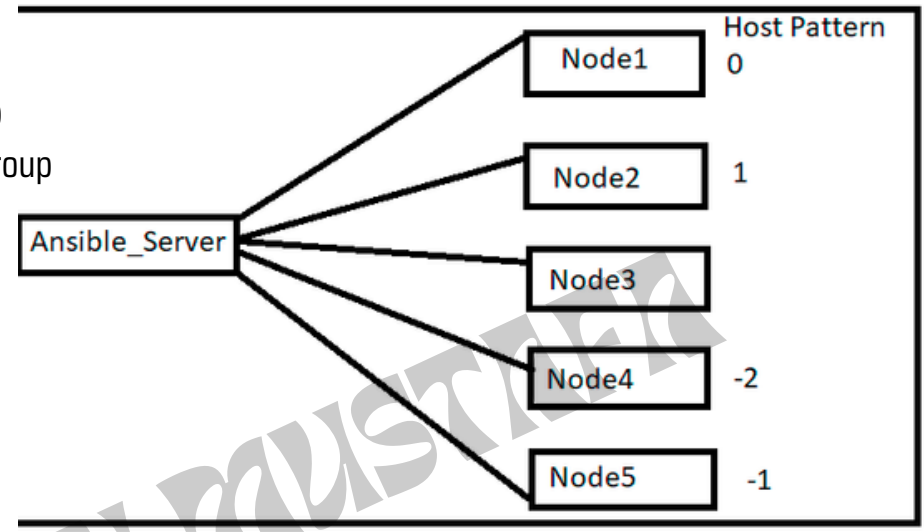
groupname [1] – picks second machine of group

groupname [-1] – picks last machine of group

groupname [1:4] – picks 2,3,4,5 machines in group

groupname [2:5] – picks 3 to 6 machines in the group

ansible all -m ping -v



ADHOC COMMANDS:

These commands can be run individually to perform Quick functions.

Not used for configuration management and deployment, bcz the cmds are one time usage.

The ansible ad-hoc cmds uses /usr/bin/ansible/ command line tool to automate single task.

Go to ansible server and switch to ansible server

ansible remo -a "ls" [remo: Group name, -a: argument, ls: command]

ansible remo [0] -a "touch file1"

ansible all -a "touch file2"

ansible remo -a "sudo yum install httpd -y"

ansible remo -ba "yum install httpd -y" (b: become you will become sudo user)

ansible remo -ba "yum remove httpd -y"

MODULES:

Ansible ships with number of modules (called library modules) that can be executed directly to remote hosts or playbooks.
Your library of modules can reside on any machine, and there are no servers, daemons or database required.
The default location for the inventory file is /etc/ansible/hosts
Go to ansible server and switch to ansible server

ansible remo -b -m yum -a “pkg=httpd state=present” (install: present)
ansible remo -b -m yum -a “pkg=httpd state=latest” (update: latest)
ansible remo -b -m yum -a “pkg=httpd state=absent” (uninstall: absent)
ansible remo -b -m service -a “name=httpd state=started” (started: start)
ansible remo -b -m user -a “name=raj” (to check go to that servers and sudo cat /etc/passwd).
ansible remo -b -m copy -a “src=filename dest=/tmp” (to check go to that server and give ls /tmp).

ANSIBLE SETUP MODULES:

ansible_os_family

os name like RedHat, Debian, Ubuntu etc..

ansible_processor_cores

No of CPU cores

ansible_kernel

Based on the kernel version

ansible_devices

connected devices information

ansible_default_ipv4

IP Mac address, Gateway

ansible_architecture

64 Bit or 32 Bit

DEVOPS BY MUSTAFA

After executing a playbook, if you want to see the output in json format
ansible -m setup private_ip

if you want to apply a see particular output, you can apply filter.
ansible -m setup -a "filter=ansible_os_family" private_ip
ansible -m setup -a "filter=ansible_devices" private_ip
ansible -m setup -a "filter=ansible_kernel" private_ip

ANSIBLE DEBUG:

it can fix errors during execution instead of editing your playbook.

```
---
- hosts: dev
  user: ansible
  become: yes
  connection: ssh

  tasks:
    - debug:
        msg: "os family for {{ansible_fqdn}} is {{ansible_os_family}}"
~
```

You can see that the task is performing on which OS.

If you run a command to check the files along with its content in a server, it will not shows us the output. But we can debug the output.

```
---
- hosts: dev
  user: ansible
  become: yes
  connection: ssh

  tasks:
    - name: get users
      command: cat /etc/passwd
      register: output

    - debug:
        msg: "users list in the ansible is {{output.stdout}}"
```

ANSIBLE INTERVIEW QUESTIONS

What Is Ansible?

Ansible is a configuration management system. It is used to set up and manage infrastructure and applications. It allows users to deploy and update applications using SSH, without needing to install an agent on a remote system.

What's the use of Ansible?
Ansible is used for managing IT infrastructure and deploy software apps to remote nodes.
For example, Ansible allows you to deploy as an application to many nodes with one single command. However, for that, there is a need for some programming knowledge to understand the ansible scripts.

Explain a few of the basic terminologies or concepts in Ansible.

Modules — Ansible uses modules to execute tasks. (Core and Custom)
Playbook — playbooks in a file which is having a list of commands to execute on target servers.
Inventory File — Defines a collection of hosts managed by Ansible (Static and Dynamic)
Ansible roles — Roles provide a framework for fully independent, or interdependent collections of variables, tasks, files, templates, and modules.
Task — Unit of work (or) Each task represents a single procedure that needs to be executed, e.g. Install a library.
APIs — Ansible Tower's REST API, provides the REST interface to the automation platform
Play — A task executed from start to finish or the execution of a playbook is called a play.
Role — An Ansible role is a pre-defined way for organizing playbooks and other files in order to facilitate sharing and reusing portions of provisioning.
Facts: Facts are global variables that store details about the system, like network interfaces or operating system.
Handlers: Are used to trigger the status of a service, such as restarting or stopping a service.

DEVOPS BY MUSTAFA

Ansible Ad-Hoc Commands?

Ansible ad-hoc command uses the ansible command-line tool to automate a task on managed nodes. These tasks avoid using playbooks and use modules to perform any operation quickly on nodes.

e.g. ping all nodes

ansible all -m ping
format of command
ansible [host-pattern] -m [module] -a "[module options]" -i "[inventory]"

e.g. ensure service is started on all webservers:
$ ansible all -m ansible.builtin.service -a "name=httpd state=started"

Ansible Variables

Ansible support variables that can be reusable in the project. Variables provide the way to handle dynamic values. Variables are scoped as well.

Global – Set for all hosts
Host – For particular host
Play – Set for all but in the context of the play

How is Ansible different from Puppet?

Metrics	Ansible	Puppet
1.Availability	•Highly available	•Highly available
2.Ease of set up	•Easy	•Comparatively hard to set up
3.Management	•Easy management	•Not very easy
4.Scalability	•Highly scalable	•Highly scalable
5.Configuration language	•YAML(Python)	•DSL(PuppetDSL)
6.Interoperability	•High	•High
7.Pricing nodes	•$10,000	•$11200-$19900

Compare Ansible with Chef.

Metrics	Ansible	Chef
1.Availability	•Highly available	•Highly available
2.Ease of set up	•Easy	•Comparatively hard to set up
3.Management	•Easy management	•Not very easy
4.Scalability	•Highly scalable	•Highly scalable
5.Configuration language	•YAML(Python)	•DSL(Ruby)
6.Interoperability	•High	•High
7.Pricing nodes	•$10,000	$13700

DEVOPS BY MUSTAFA

What are Ad-hoc commands? Give an example.

Ad-hoc commands are simple one-line commands used to perform a certain task. You can think of Adhoc commands as an alternative to writing playbooks. An example of an Adhoc command is as follows:

```
- hosts: your hosts
vars:
port_Tomcat : 8080
```

Here, we've defined a variable called port_Tomcat and assigned the port number 8080 to it. Such a variable can be used in the Ansible Playbook.

What are Ansible Vaults and why are they used?

Ansible Vault is a feature that allows you to keep all your secrets safe. It can encrypt entire files, entire YAML playbooks or even a few variables. It provides a facility where you can not only encrypt sensitive data but also integrate them into your playbooks.

Vault is implemented with file-level granularity where the files are either entirely encrypted or entirely unencrypted. It uses the same password for encrypting as well as for decrypting files which makes using Ansible Vault very user-friendly.

How to create encrypted files using Ansible?

To create an encrypted file, use the 'ansible-vault create' command and pass the filename.

```
ansible-vault create filename.yaml
```

Is Ansible an Open Source tool?

Yes, Ansible is open source. That means you take the modules and rewrite them. Ansible is an open-source automated engine that lets you automate apps.

DOCKER

MONOLITHIC

If an application contains N number of services (Let's take Paytm has Money Transactions, Movie Tickets, Train tickets, etc..) If all these services are included in one server then it will be called Monolithic Architecture. Every monolithic Architecture has only one database for all the services.

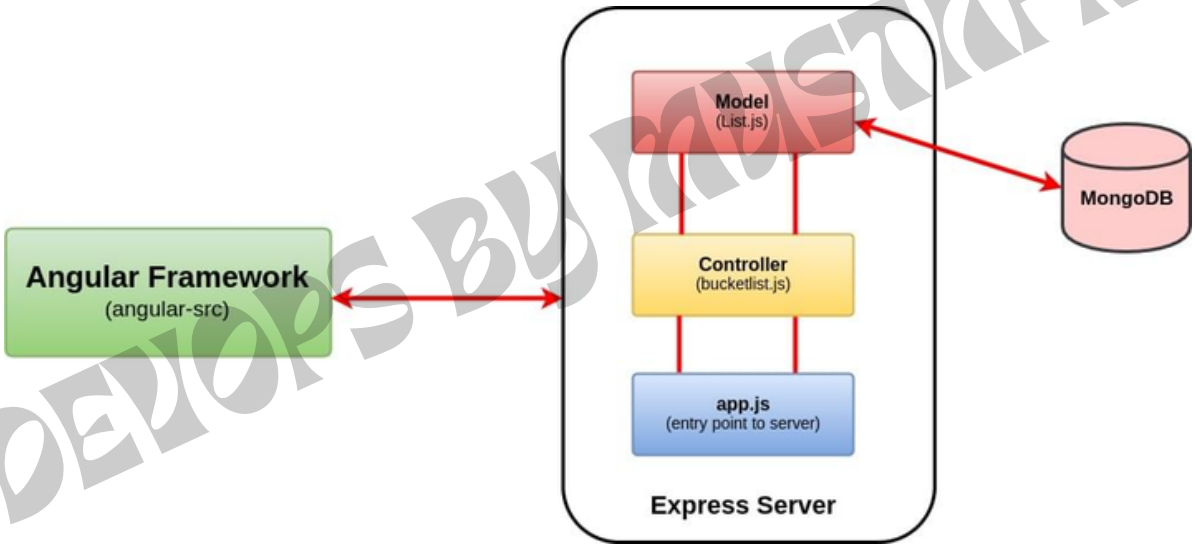
MICRO SERVICE

If an application contains N number of services (Let's take Paytm has Money Transactions, Movie Tickets, Train tickets, etc..) if every service has its own individual servers then it is called microservices. Every microservice architecture has its own database for each service.

DEVOPS BY MUSTAFA

WHY DOCKER

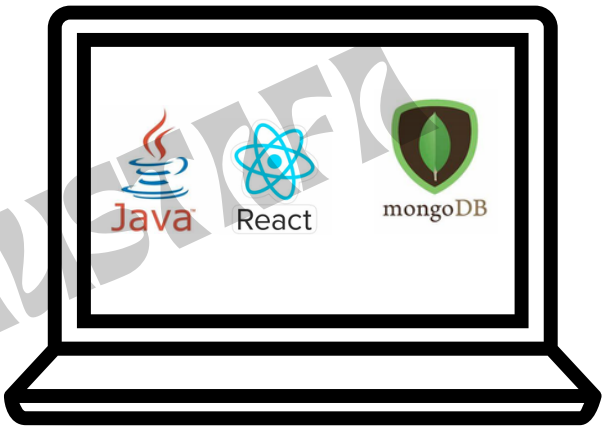
let us assume that we are developing an application, and every application has front end, backend and Database.



To develop the application we need install those dependencies to run to the code.

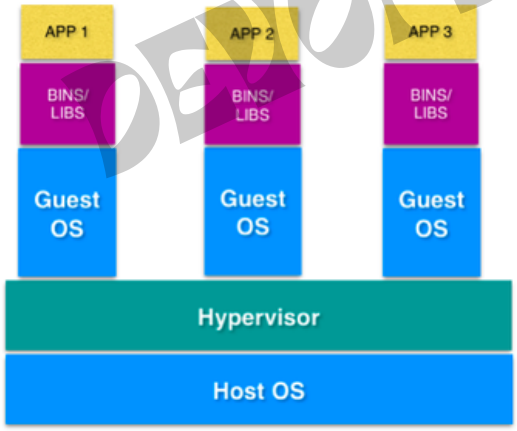
So i installed Java11, ReactJS and MongoDB to run the code.
After some time, i need another versions of java, react and mongo DB for my application to run the code.

So its really a hectic situation to maintain multiple versions of same tool in our system.



VIRTUALIZATION:

It is used to create a virtual machines inside on our machine. in that virtual machines we can hots guest OS in our machine.
by using this Guest OS we can run multiple application on same machine.
Hypervisor is used to create the virtualization.



VIRTUAL MACHINE ARCHITECTURE

DRAWBACKS:

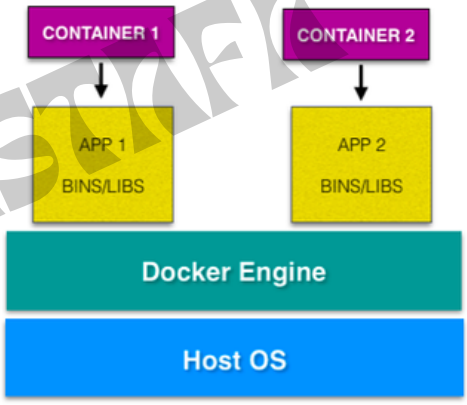
- It is old method.
- If we use multiple guest OS then the performance of the system is low.

CONTAINERIZATION:

- It is used to pack the application along with its dependencies to run the application.

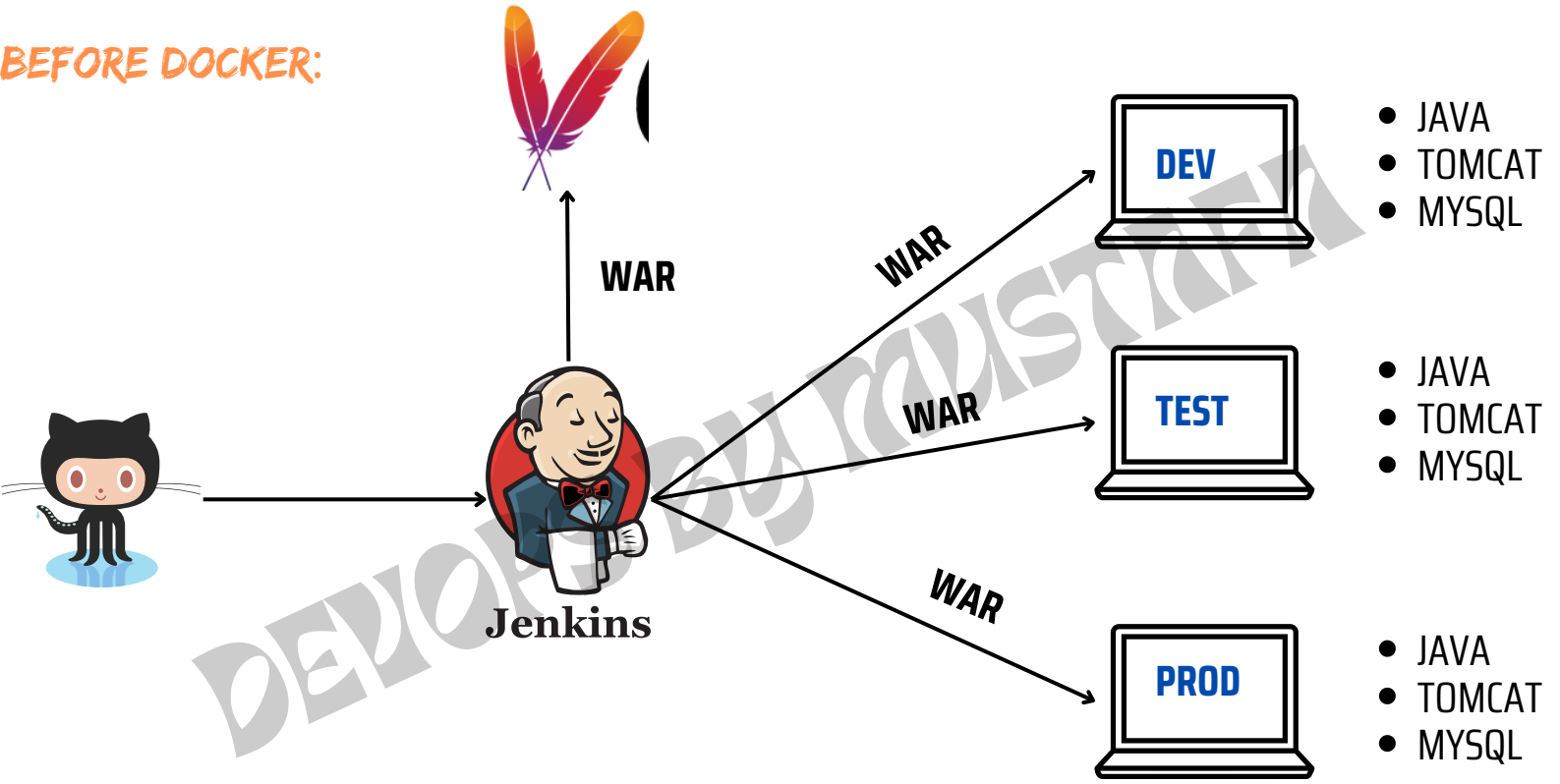
CONTAINER:

- Container is nothing but, it is a virtual machine which does not have any OS.
- Docker is used to create these containers.



DOCKER ARCHITECTURE

BEFORE DOCKER:



AFTER DOCKER:

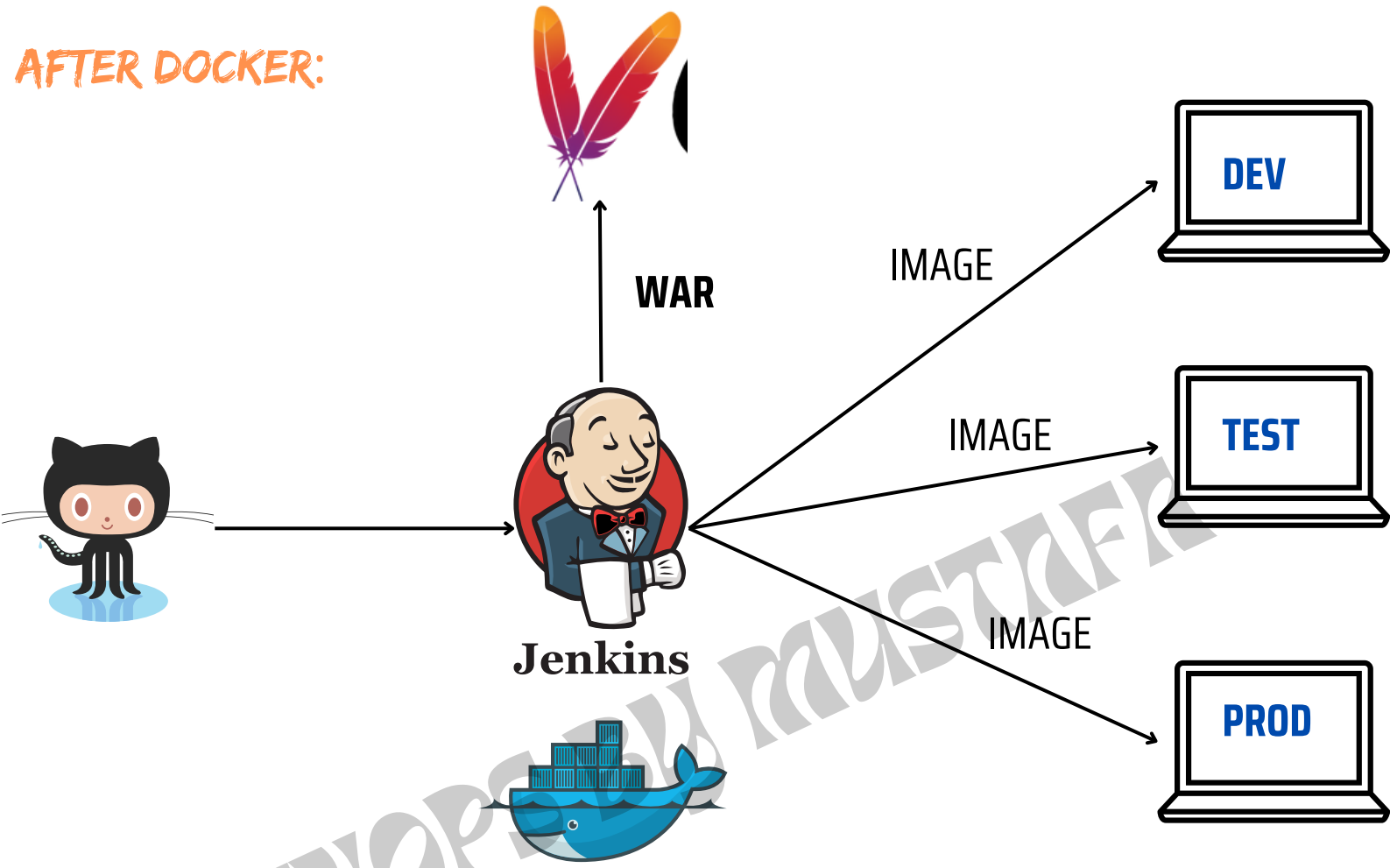
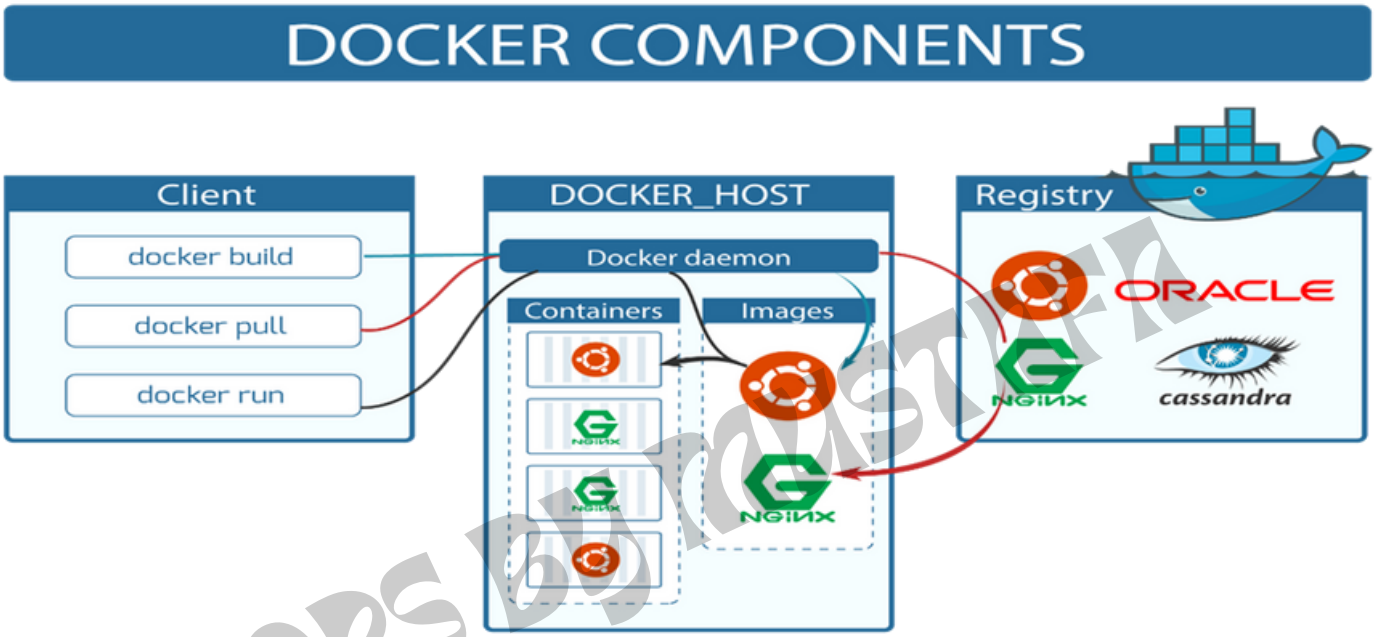


IMAGE = WAR + JAVA + TOMCAT + MYSQL

DOCKER

- It is an open source centralized platform designed to create, deploy and run applications.
- Docker is written in the Go language.
- Docker uses containers on host O.S to run applications. It allows applications to use the same Linux kernel as a system on the host computer, rather than creating a whole virtual O.S.
- We can install Docker on any O.S but the docker engine runs natively on Linux distribution.
- Docker performs O.S level Virtualization also known as Containerization.
- Before Docker many users face problems that a particular code is running in the developer's system but not in the user system.
- It was initially released in March 2013, and developed by Solomon Hykes and Sebastian Pahl.
- Docker is a set of platform-as-a-service that use O.S level Virtualization, where as VM ware uses Hardware level Virtualization.
- Container have O.S files but its negligible in size compared to original files of that O.S.

ARCHITECTURE:



DOCKER CLIENT:

It is the primary way that many Docker users interact with Docker. When you use commands such as docker run, the client sends these commands to docker daemon, which carries them out. The docker command uses the Docker API.

DOCKER HOST:

Docker host is the machine where you installed the docker engine.

DOCKER DAEMON:

Docker daemon runs on the host operating system. It is responsible for running containers to manage docker services. Docker daemon communicates with other daemons. It offers various Docker objects such as images, containers, networking, and storage.

DOCKER REGISTRY:

A Docker registry is a scalable open-source storage and distribution system for docker images.

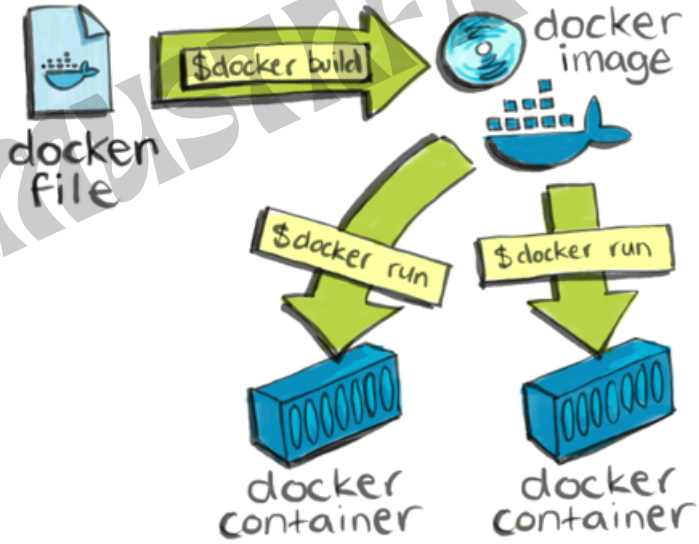
DEVOPS BY MUSTAFA

DOCKER FILE:

- It is basically a text file which contains some set of instructions.
- Automation of Docker image creation.
- Always D is capital letters on Docker file.
- And Start Components also be Capital letter.

HOW IT WORKS:

- First you need to create a Docker file
- Build it
- Create a container using the image



DOCKER FILE COMPONENTS:

FROM: For base image this command must be on top of the file. Ex: ubuntu, Redis, Jenkins
LABEL: Labeling like EMAIL, AUTHOR, etc.
RUN: To execute commands and commit the layer.
COPY: Copy files/folders from local system (docker VM) where need to provide Source and Destination.
ADD: It can download files from the internet and also, we can extract files at docker image side.
EXPOSE: To expose ports such as 8080 for tomcat and port 80 nginx etc.
WORKDIR: To set working directory for the Container.
CMD: Executes commands but during Container creation.
ENTRYPOINT: The command that executes inside of a container. like running the services in a container.
ENV: Environment Variables.

DEVOPS BY MUSTAFA

DOCKER FILE TO CREATE AN IMAGE:

FROM: ubuntu
RUN: touch aws devops linux

FROM: ubuntu
RUN: touch aws devops linux
RUN echo "hello world">/tmp/file1

TO BUILD: docker build -t image_name . (. represents current directory)

Now see the image and create a new container using this image.
go to container and see the files that you created.


```
FROM ubuntu
WORKDIR /tmp
RUN echo "hello world!" > /tmp/testfile
ENV myname raham
COPY testfile1 /tmp
ADD test.tar.gz /tmp
```

```
[root@ip-172-31-83-27 ~]# touch testfile1
[root@ip-172-31-83-27 ~]# touch test
[root@ip-172-31-83-27 ~]# ls
Dockerfile  test  testfile1
[root@ip-172-31-83-27 ~]# tar -cvf test.tar test
test
[root@ip-172-31-83-27 ~]# ls
Dockerfile  test  testfile1  test.tar
[root@ip-172-31-83-27 ~]# gzip test.tar
[root@ip-172-31-83-27 ~]# ls
Dockerfile  test  testfile1  test.tar.gz
[root@ip-172-31-83-27 ~]# rm -rf test
[root@ip-172-31-83-27 ~]# ls
Dockerfile  testfile1  test.tar.gz
[root@ip-172-31-83-27 ~]# docker build -t raham .
```

DOCKER RUN VS CMD VS ENTRYPOINT:

RUN: it is used to execute the commands while we build the images and add a new layer into the image.

```
FROM centos:centos7
RUN yum install git -y
 or
RUN [“yum”, “install”, “git” “-y”]
```

CMD: it is used to execute the commands when we run the container.
    It is used to set the default command.
    if we have multiple CMD’s only last one will gets executed.

```
FROM centos:centos7
CMD yum install maven -y
 or
CMD [“yum”, “install”, “maven”, “-y”]
```

If you want to overwrite the parameters:
 docker run image_name httpd (FAILED)
 docker run image_name yum install httpd -y (only httpd will gets installed)

ENTRYPOINT: it overwrites the CMD when you pass additional parameters while running the container.

```
FROM centos:centos7
ENTRYPOINT [“yum”, “install”, “maven”, “-y”]
```

If you want to overwrite the parameters:
 docker run image_name httpd (both maven and httpd will gets installed)
 docker run image_name yum install httpd -y (both maven and httpd will gets installed)

```
FROM centos:centos7
ENTRYPOINT [“yum”, “install”, “-y”]
CMD [“httpd”]
```

Bydefault it will executes httpd command, if you specify the command while running the container it will gets executed.
 docker run image_name (httpd will install)
 docker run image_name git (only git will install)
 docker run image_name git tree(both git & tree will install)

DEVOPS BY MUSTAFA

ARG: it is used to define a variable that is used to build a docker image, it will not available once we build it. In containers also it's not possible to access it.
we can change these variables in command line arguments (docker build -t test --build-arg var_name=value)
ENV: These variables are used for inside the container
we can't change the values in command line arguments. f we need to change the ENV variable using CMD line then we have to use ARG and place ARG variable in ENV variable

```
ARG abc=devops
ENV xyz=aws
RUN echo $abc
RUN echo $xyz
```


DOCKER DIFFERENCES:

ARG argument is not available inside the Docker containers and
ENV argument is accessible inside the container

RUN: it is used to execute the commands while we build the images and add a new layer into the image.
CMD: it is used to execute the commands when we run the container.
if we have multiple CMD's only last one will gets executed.
ENTRYPOINT: it overwrites the CMD when you pass additional parameters while running the container.

COPY: Used to copy local files to containers
ADD: Used to copy files form internet and extract them

STOP: attempts to gracefully shutdown container, issues a SIGTERM signal to the main process.
KILL: immediately stops/terminates them, while docker kill (by default) issues a SIGKILL signal.

DOCKER FILES:

DOCKER FILE TO DEPLOY STATIC WEBSITE USING NGINX:

```
FROM ubuntu
RUN apt-get update
RUN apt-get install nginx -y
COPY index.html /var/www/html/
EXPOSE 80
CMD ["nginx","-g","daemon off;"]
~
~
```

OR

```
FROM nginx
COPY . /usr/share/nginx/html
~
~
~
~
~
```

REFERENCE: **https://github.com/devops0014/staticsite-docker.git**

TIC-TAC-TOE GAME

REFERENCE: **https://github.com/devops0014/tic-tac-toe-docker.git**

DOCKER FILE TO DEPLOY NODE JS FILE:

```
FROM node:16
WORKDIR /app
COPY package*.json ./
RUN npm install
COPY . .
EXPOSE 8081
CMD ["node", "index.js"]
~
```

REFERENCE: **https://github.com/devops0014/nodejs-docker.git**

```
FROM ubuntu:14.04
MAINTAINER abc "abc@abc.com"
RUN apt-get update
RUN apt-get install -y nginx
RUN echo 'Our first Docker image for Nginx' > /usr/share/nginx/html/index.html
EXPOSE 80
~
~
```

TO BUILD:

docker build -t image1 .

TO RUN:

docker run -dit --name mustafa -p 8081:80 image1 nginx -g "daemon off;"

DOCKER FILE FOR APACHE2

```
FROM UBUNTU
RUN APT-GET UPDATE -Y
RUN APT-GET INSTALL APACHE2 -Y
COPY INDEX.HTML /VAR/WWW/HTML/
CMD ["/USR/SBIN/APACHECTL", "-D",
"FOREGROUND"]
```

DOCKER FILE FOR HTTPD

```
FROM CENTOS:CENTOS7
RUN YUM UPDATE -Y
RUN YUM INSTALL HTTPD -Y
COPY INDEX.HTML /VAR/WWW/HTML/
CMD ["/USR/SBIN/HTTPD", "-D", "FOREGROUND"]
```

DOCKER VOLUME:

- When we create a Container then Volume will be created.
- Volume is simply a directory inside our container.
- First, we have to declare the directory Volume and then share Volume.
- Even if we stop the container still, we can access the volume
- You can declare directory as a volume only while creati
- We can't create volume from existing container.
- You can share one volume across many number of Con
- Volume will not be included when you update an image.
- If Container-1 volume is shared to Container-2 the changes made by Container-2 will be also available in the Container-1.

```
FROM ubuntu
ADD file1 /ubuntu1/file
VOLUME /ubuntu1
~
```

- You can map Volume in two ways:

1. Container < ------ > Container
2. Host < ------- > Container

CREATING VOLUMES FROM DOCKER FILE:

- Create a Docker file and write
  FROM ubuntu
  VOLUME["/myvolume"]
- build it - docker build -t image_name .
- Run it - docker run -it - -name container1 ubuntu /bin/bash
- Now do ls and you will see myvolume-1 add some files there
- Now share volume with another Container - docker run -it - -name container2(new) - -privileged=true - -volumes-from container1 ubuntu
- Now after creating container2, my volume1 is visible
- Whatever you do in volume1 in container1 can see in another container
- touch /myvolume1/samplefile1 and exit from container2.
- docker start container1
- docker attach container1
- ls/volume1 and you will see your samplefile1

CREATING VOLUMES FROM COMMAND:

- docker run -it - -name container3 -v /volume2 ubuntu /bin/bash
- now do ls and cd volume2.
- Now create one file and exit.
- Now create one more container, and share Volume2 - docker run-it - -name container4 - - -privileged=true - -volumes-from container3 ubuntu
- Now you are inside container and do ls, you can see the Volume2
- Now create one file inside this volume and check in container3, you can see that file

VOLUMES (HOST TO CONTAINER):

- Verify files in /home/ec2-use
- docker run -it - -name hostcont -v /home/ec2-user:/raham - -privileged=true ubuntu
- cd raham [raham is (container-name)]
- Do ls now you can see all files of host machine.
- Touch file1 and exit. Check in ec2-machine you can see that file.

SOME OTHER COMMANDS:

- docker volume ls
- docker volume create <volume-name>
- docker volume rm <volume-name>
- docker volume prune (it will remove all unused docker volumes).
- docker volume inspect <volume-name>
- docker container inspect <container-name>

MOUNT VOLUMES:

- To attach a volume to a container: docker run -it --name=example1 --mount source=vol1,destination=/vol1 ubuntu
- To send some files from local to container:
  - create some files
  - docker run -it --name cont_name -v "$(pwd)":/my-volume ubuntu
- To remove the volume: docker volume rm volume_name
- To remove all unused volumes: docker volume prune

BASE VOLUMES:

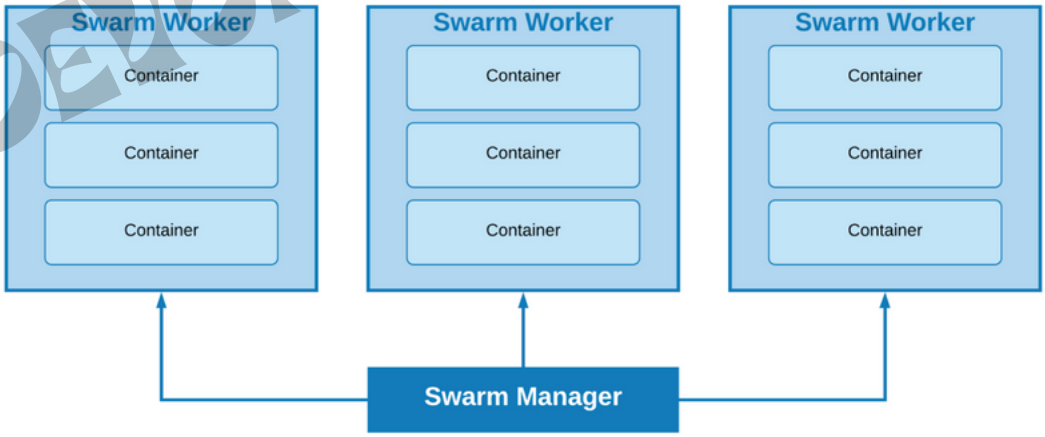
STEPS

- create a volume : docker volume create volume99(volume-name)
- mount it: docker run -it -v volume99:/my-volume --name container1 ubuntu
- now go to my-volume and create some files over there and exit from container
- mount it: docker run -it -v volume99:/my-volume-01 --name container2 ubuntu

DEVOPS BY MUSTAFA

DOCKER SWARM:

- Docker swarm is an orchestration service within docker that allows us to manage and handle multiple containers at the same time.
- It is a group of servers that runs the docker application.
- It is used to manage the containers on multiple servers.
- This can be implemented by the cluster.
- The activities of the cluster are controlled by a swarm manager, and machines that have joined the cluster is called swarm worker.



- Docker Engine helps to create Docker Swarm.
- There are mainly worker nodes and manager nodes.
- The worker nodes are connected to the manager nodes.
- So any scaling or update needs to be done first it will go to the manager node.
- From the manager node, all the things will go to the worker node.
- Manager nodes are used to divide the work among the worker nodes.
- Each worker node will work on an individual service for better performance.

DOCKER SWARM COMPONENTS:

SERVICE: Represents a part of the feature of an application.
TASK: A single part of work.
MANAGER: This manages the work among the different nodes.
WORKER: Which works for a specific purpose of the service.

SETUP:

Create 3 node one is manager and another two are workers
Manager node: docker swarm init --advertise-addr (private ip)
Run the below command to join the worker nodes
To check nodes on docker swarm: docker node ls
Here * Indicates the current node like master branch on git
Now we created the docker swarm cluster
docker info : To see all the docker info running on our machine.
docker swarm leave : To down the docker node (need to wait few sec)
docker node rm node-id : To remove the node permenantly
docker swarm leave : To delete the swarm but will get error
docker swarm leave –force : To delete the manager forcefully
docker swarm join-token worker. : To get the token of the worker
docker swarm join-token manager : To get the token of the worker

SWARM SERVICE:

Now we want to run a service on the swarm
So we want to run a specific container on all these nodes
To do that we will use a docker service command which will create a service for us
That service is nothing but a container.
We have a replicas here when one replica goes down another will work for us.
Atleast one of the replica needs to be up among them.

docker service create --name raham --replicas 3 --publish 80:80 httpd
raham : service name replicas : nodes publish : port reference image: apache

docker service ls : To list the services
docker service ps service-name : To see where the services are running
docker ps : To see the containers (Check all nodes once)
docker service rm service_name : To remove the service (it will come again later)
public ip on browser : To check its up and running or not
docker service rm service-name : To remove the service

To create a service: docker service create —name devops —replicas 2 image_name
Note: image should be present on all the servers
To update the image service: docker service update —image image_name service_name
Note: we can change image,
To rollback the service: docker service rollback service_name
To scale: docker service scale service_name=3
To check the history: docker service logs
To check the containers: docker service ps service_name
To inspect: docker service inspect service_name
To remove: docker service rm service_name

DOCKER COMPOSE:

- It is a tool used to build, run and ship the multiple containers for application.
- It is used to create multiple containers in a single host.
- It used YAML file to manage multi containers as a single service.
- The Compose file provides a way to document and configure all of the application's service dependencies (databases, queues, caches, web service APIs, etc).

COMMANDS:

- Start all services: Docker Compose up.
- Stop all services: Docker Compose down.
- Run Docker Compose file: Docker-compose up -d.
- List the entire process: Docker ps.

COMPOSE FILE:

The Docker Compose file includes Services, Networks and Volumes.
The Default Path is ./docker-compose.yml
It contains a service definition which configures each container started for that service

COMPOSE INSTALLATION:

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
sudo chmod +x /usr/local/bin/docker-compose
sudo ln -s /usr/local/bin/docker-compose /usr/bin/docker-compose
docker-compose --version

COMPOSE FILE:

- version - specifies the version of the Compose file.
- services - it the services in your application.
- networks - you can define the networking set-up of your application.
- volumes - you can define the volumes used by your application.
- configs - configs lets you add external configuration to your containers. Keeping configurations external will make your containers more generic.

CREATING DOCKER-COMPOSE.YML:

```
version: '3'
services:
  webapp1:
    image: nginx
    ports:
      - "8000:80"
```

vim docker-compose.yml
Version: It is the compose file format which supports the relavent docker engine
Services: The services that we are going to use by this file (Webapp1 is service name)
Image: Here we are taking the Ngnix image for the webserver
Ports: 8000 port is mapping to container port 80
Docker-compose up -d
Public-ip:8000 -- > You can see the Nginx image
Docker network ls -- > you can see root_default
Docker-compose down -- > It will delete all the Created containers

DEVOPS BY MUSTAFA

```yaml
version: '3'
services:
  webapp1:
    image: nginx
    ports:
      - "8000:80"
  webapp2:
    image: nginx
    ports:
      - "8001:80"
```

Docker-compose up -d

Public-ip:8000 & public-ip:8001-- > You can see the Nginx image on both ports

Docker container ls

Docker network ls

CHANGING DEFAULT FILE:

mv docker-compose.yml docker-compose1.yml

docker-compose up -d

You will get some error because you are changing by default docker-compose.yml

Use the below command to overcome this error

docker-compose -f docker-compose1.yml up -d

docker-compose -f docker-compose1.yml down

```yaml
version: "3.1"
services:
  mobile_recharge:
    image: nginx
    ports:
      - "9999:80"
    volumes:
      - "mobile-recharge-volume1"
    networks:
      - "mobile-recharge-network"

  mobile_recharge2:
    image: nginx
    ports:
      - "9998:80"
networks:
  mobile-recharge-network:
    driver: bridge
~
```

docker-compose up -d - used to run the docker file
docker-compose build - used to build the images
docker-compose down - remove the containers
docker-compose config - used to show the configurations of the compose file
docker-compose images - used to show the images of the file
docker-compose stop - stop the containers
docker-compose logs - used to show the log details of the file
docker-compose pause - to pause the containers
docker-compose unpause - to unpause the containers
docker-compose ps - to see the containers of the compose file

DOCKER STACK:

- Docker stack is used to create multiple services on multiple hosts. i.e it will create multiple containers on multile servers with the help of compose file.
- To use the docker stack we have initialized docker swarm, if we are not using docker swarm, docker stack will not work.
- once we remove the stack automatically all the containers will gets deleted.
- We can share the containers from manager to worker according to the replicas
- Ex: Lets assume if we have 2 servers which is manager and worker, if we deployed a stack with 4 replicas. 2 are present in manager and 2 are present in worker.
- Here manager will divide the work based on the load on a server

COMMAND:
TO DEPLOY : docker stack deploy --compose-file docker-compose.yml stack_name
TO LIST : docker stack ls
TO GET CONTAINERS OF A STACK : docker stack ps stack_name
TO GET SERVICES OF A STACK: docker stack services stack_name
TO DELETE A STACK: docker stack rm stack_name

PORTAINER:

- it is a container organizer, designed to make tasks easier, whether they are clustered or not.
- abel to connect multiple clusters, access the containers, migrate stacks between clusters
- it is not a testing environment mainly used for production routines in large companies.
- Portainer consists of two elements, the Portainer Server and the Portainer Agent.
- Both elements run as lightweight Docker containers on a Docker engine

PORTAINER:

- Must have swarm mode and all ports enable with docker engine
- curl -L https://downloads.portainer.io/ce2-16/portainer-agent-stack.yml -o portainer-agent-stack.yml
- docker stack deploy -c portainer-agent-stack.yml portainer
- docker ps
- public-ip of swamr master:9000

DEVOPS BY MUSTAFA

LIMITS TO CONTAINER

It is used to set a memory limits to containers

docker run -dit --name cont_name --memory=250m --cpu="0.25" image_name

to check: docker inspect cont_name | grep -i memory

to check: docker inspect cont_name | grep -i nanocpu

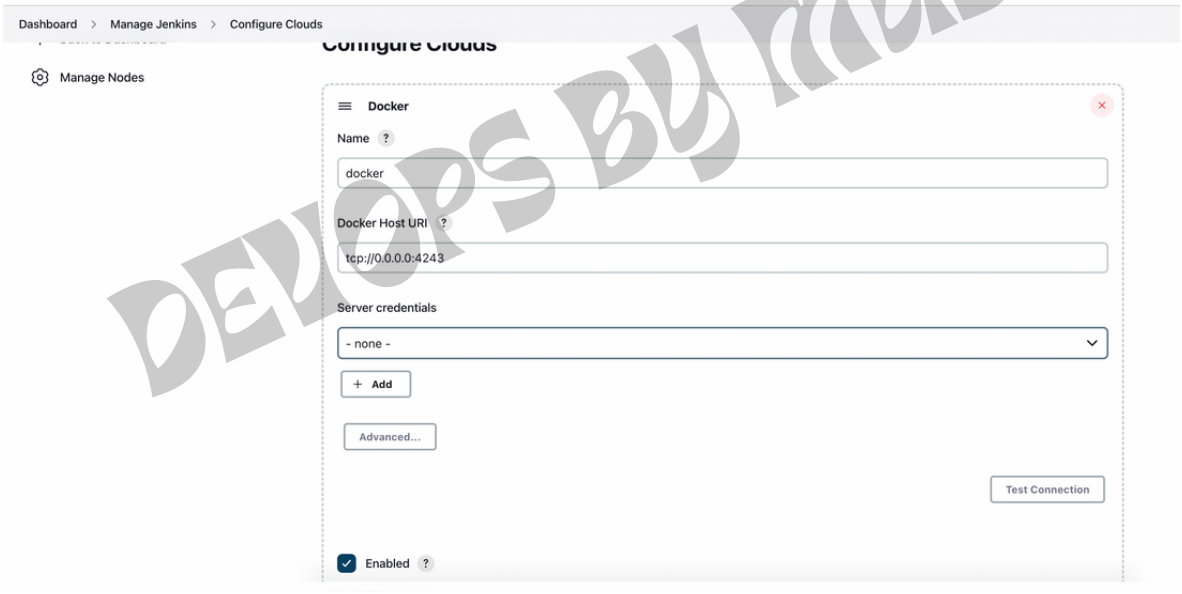
DOCKER INTEGRATION WITH JENKINS

- Install docker and Jenkins in a server.
- vim /lib/systemd/system/docker.service

```
[Service]
Type=notify
# the default is not to use systemd for cgroups because the delegate issues still
# exists and systemd currently does not support the cgroup feature set required
# for containers run by docker
ExecStart=/usr/bin/dockerd -H tcp://0.0.0.0:4243 -H unix:///var/run/docker.sock
ExecReload=/bin/kill -s HUP $MAINPID
TimeoutSec=0
RestartSec=2
Restart=always
```

- Replace the above line with
  - ExecStart=/usr/bin/dockerd -H tcp://0.0.0.0:4243 -H unix:///var/run/docker.sock
- systemctl daemon-reload
- service docker restart
- curl http://localhost:4243/version

- Install Docker plugin in Jenkins Dashboard.
- Go to manage jenkins>Manage Nodes & Clouds>>Configure Cloud.
- Add a new cloud >> Docker
- Name: Docker
- add Docker cloud details.



DOCKER DIRECTORY DATA:

We use docker to run the images and create the containers. but what if the memory is full in instance. we have a add a another volume to the instance and mount it to the docker engine. Lets see how we do this.

- Uninstall the docker - yum remove docker -y
- remove all the files - rm -rf /var/lib/docker/*
- create a volume in same AZ & attach it to the instance
- to check it is attached or not - fdisk -l
- to format it - fdisk /dev/xvdf --> n p 1 enter enter w
- set a path - vi /etc/fstab (/dev/xvdf1 /var/lib/docker/ ext4 defaults 0 0)
- mkfs.ext4 /dev/xvdf1
- mount -a
- install docker - yum install docker -y && systemctl restart docker
- now you can see - ls /var/lib/docker
- df -h

CONTAINER LOAD BALANCER:

```
.
├── docker-compose.yml
├── nginx
│   ├── Dockerfile
│   └── nginx.conf
├── swiggy
│   ├── Dockerfile
│   └── index.html
└── zomato
    ├── Dockerfile
    └── index.html
```

DOCKER FILE FOR SWIGGY AND ZOMATO SERVICES

```
FROM nginx
COPY . /usr/share/nginx/html
~
~
```

DOCKER FILE FOR NGINX

```
FROM nginx
RUN rm /etc/nginx/conf.d/default.conf
COPY nginx.conf /etc/nginx/conf.d/default.conf
~
```

NGINX.CONF FILE

```
upstream loadbalancer {
server 172.17.0.1:5001 weight=6;
server 172.17.0.1:5002 weight=4;
}
server {
location / {
proxy_pass http://loadbalancer;
}}
```

COMPOSE FILE

```
version: '3'
services:
  swiggy:
    image: image1
    ports:
      - "5001:80"
  zomato:
    image: image2
    ports:
      - "5002:80"
  nginx:
    build: ./nginx
    ports:
      - "8080:80"
    depends_on:
      - swiggy
      - zomato
~
```

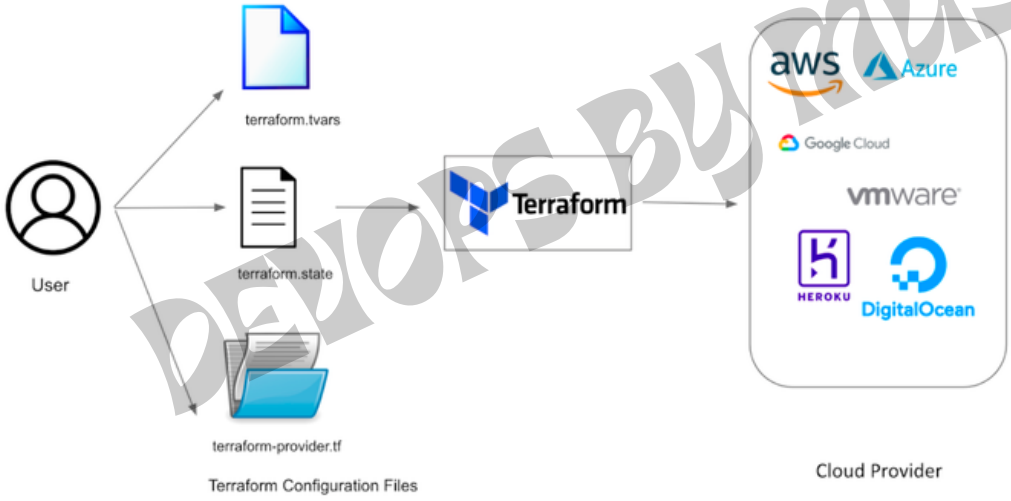
After writing all the file, we need to build the Dockerfiles for both the zomato and swiggy services.
command: **docker build -t image_name .**

write the docker-compose file and build it.
command: **docker-compose up -d**

access the application: **publicip:8080**

TERRAFORM:

- Terraform is an open source "Infrastructure as a Code" tool, created by HashiCorp.
- It was developed by Mitchell Hashimoto with Go Language in th year 2014 which
- All the configuration files used (HashiCorp Configuration Language) language for the code.
- Terraform uses a simple syntax, can provision infrastructure across multiple clouds & On premises.
- It is Cloud Agnostic it means the systems does not depends on single provider.



IAAC:

- Infrastructure as a Code (IaaC) is the managing and provisioning of infrastructure through code instead of through manual processes.
- With IaaC, configuration files are created that contain your infrastructure specifications, which makes it easier to edit and distribute configurations.
- terraform could handle low-level elements like networking, storage, compute instances, also high-level elements like SaaS features, DNS entries, etc.
- It is famous for easy to use but not true for complex environments it is not easy.
- Eliminates the need for manual infrastructure provisioning and management.

ALTERNATIVES:

AWS -- > CFT (JSON/YAML)
AZURE -- > ARM TEMPLATES (JSON)
GCP -- > CLOUD DEPLOYMENT MANAGER (YAML/ PYTHON)
PULUMI -- (PYTHON, JS, C#, GO & TYPE SCRIPT)
ANSIBLE -- > (YAML)
PUPPET
CHEF
VAGRANT
CROSSPLANE

ADVANTAGES:

Readable code.
Dry run.
Importing of Resources is easy.
Creating of multiple resources.
Can create modules for repeatable code.

DIS ADVANTAGES:

- Currently under development. Each month, we release a beta version.
- There is no error handling
- There is no way to roll back. As a result, we must delete everything and re-run code.
- A few things are prohibited from import.
- Bugs

TERRAFORM SETUP:

- wget https://releases.hashicorp.com/terraform/1.1.3/terraform_1.1.3_linux_amd64.zip
- sudo apt-get install zip -y
- Unzip terraform
- mv terraform /usr/local/bin/
- terraform version
- cd ~
- mkdir terraform & vim main.tf
- write the basic code
- Go to IAM andcreate a user called terraform and give both access give admin access.

DEVOPS BY MUSTAFA

TERRAFORM INIT:

It initializes the provider, module version requirements, and backend configurations.

TERRAFORM PLAN:

Determines the state of all resources and compares them with real or existing infrastructure. It uses terraform state file data to compare and provider API to check.

TERRAFORM APPLY:

Executes the actions proposed in a Terraform plan.

TERRAFORM DESTROY:

It will destroy terraform-managed infrastructure or the existing enviornment

NOTE: We can use -auto-approve command for Apply and Destroy phases.

CREATING EC2:

```
provider "aws" {
 region          = "ap-south-1"
 access_key      = "AKIAWW7WL2JMJKCCMORC"
 secret_key      = "DraPAxLZinm+ONtvchniWNG91MpqkwMvyrJVZo/B"
}

resource "aws_instance" "example" {
  ami           = "ami-0af25d0df86db00c1"
  instance_type = "t2.micro"

  tags = {
    name = "web-server"
  }
}
```

DEVOPS BY MUSTAFA