**Assignments**

**Jenkins**

**=======**

1) Jenkins Port Change

2) How to change Jenkins Home Directory

3) How to take backup in Jenkins and Restore.

4) How to install Jenkins in Tomcat

5) Remote Job in Jenkins

6) Webhook through Git

7) Jenkins Installation Script - Pending

8) Install old version of Jenkins and then upgrade it to a latest version

9) How to put https in Jenkins URL (Open Cert option)

**1) Jenkins Port Change - Done**

**Solution:**

hostnamectl set-hostname **master** - create hostname permanent

#!/bin/bash

apt update -y && apt upgrade -y && apt install openjdk-17-jdk -y

java --version

curl -fsSL https://pkg.jenkins.io/debian-stable/jenkins.io-2023.key | sudo tee \

/usr/share/keyrings/jenkins-keyring.asc > /dev/null

echo deb [signed-by=/usr/share/keyrings/jenkins-keyring.asc] \

https://pkg.jenkins.io/debian-stable binary/ | sudo tee \

/etc/apt/sources.list.d/jenkins.list > /dev/null

apt update -y

apt install jenkins -y

systemctl start jenkins

cat /var/lib/jenkins/secrets/initialAdminPassword

start Jenkins – public ip:8080

goto > /etc/default/jenkins

Environment="JENKINS\_PORT=8080" to 9090

No goto > /usr/lib/systemd/system/jenkins.service

change the port in the line

Environment="JENKINS\_PORT=8080" to 9090

Save.

systemctl daemon-reload

systemctl restart jenkins

check.

**2) How to change Jenkins Home Directory – Partial Done**

**Solution:**

**drwxr-xr-x 2 root root 4096 Sep 1 18:16 jenkins\_home**

**drwxr-xr-x 2 jenkins jenkins 4096 Sep 1 18:16 jenkins\_home**

The chown command in Linux is used to assign ownership. Every file and directory can have only one user and one group as an owner.

**Copying Directories**

To copy a directory, including all its contents (files and subdirectories), you use the -r (recursive) option:

***For example:***

***cp -r /home/user/documents /home/user/backup/documents***

**usermod** command or modify user is a command in Linux that is used to change the properties of a user in Linux through the command line. After creating a user we have to sometimes change their attributes like password or login directory etc.

**Note:**usermod command needs to be executed only as a root user.

usermod -d /home/manav test\_user

This will change the home directory of the user to /home/manav.

Same code until:

start Jenkins – public ip:8080

systemctl stop Jenkins (Stop Jenkins Service)

mkdir /home/jenkins\_home (Create a new Jenkins Home directory using the mkdir command)

chown jenkins:jenkins /home/jenkins\_home (Change permissions for the new Home directory)

cp -r /var/lib/jenkins /home/jenkins\_home (Copy the contents from the old Jenkins Home directory to the new directory)

usermod -d /home/jenkins\_home jenkins (Assign jenkins as the user for the new Home directory, check it in system or config file)

vi /etc/default/Jenkins (Open Jenkins configuration file) (esc + i to edit in vi editor)

Scroll down until you reach the JENKINS\_HOME entry and add the new Home directory

JENKINS\_HOME=/var/lib/$NAME (old home directory path)

JENKINS\_HOME=/home/jenkins\_home/$NAME (new home directory path)

Esc, then :wq! (Save and Exit the file)

vi usr/lib/systemd/system/Jenkins.service (Open Jenkins Service)

Scroll down until you reach the Environment and Working Directory, add the same Home directory

Environment="JENKINS\_HOME=/var/lib/jenkins" [Old Entry]

WorkingDirectory=/var/lib/jenkins

Environment="JENKINS\_HOME=/home/jenkins\_home/jenkins" [New Entry]

WorkingDirectory=/home/jenkins\_home/jenkins

Esc, then :wq! (Save and Exit the file)

Rename the old Jenkins home directory

mv /var/lib/Jenkins /var/lib/Jenkins.old

systemctl daemon-reload (Reload the Daemon)

systemctl start Jenkins (Start Jenkins Service)

Verify the New Home directory under > Manage Jenkins > System > Home directory

**3) How to take backup in Jenkins and Restore.**

**Solution:**

Install jenkins and run.

Create 1-2 jobs and run it.

Create a bucket inside S3

Stop jenkins >

systemctl stop jenkins

systemctl status jenkins

Create a backup folder and

cd /home

mkdir jenkins\_backup

Take backup of jenkins home directory –

tar -zcvf jenkins-backup.tar.gz /var/lib/jenkins

ls

copy the backup to safe location

aws s3 cp jenkins-backup.tar.gz s3://jenkinsbackup02092024/jenkins-backup.tar.gz

it will give error, as we have not configured aws yet and provide role.

Provide role –

Iam > Role > create > aws service > ec2 > next > add policies of s3 > s full access > next > role name (sfullaccess)

EC2 > Instance > Actions > Security > Modify IAM role > attach the created iam role and update

Run the same command –

aws s3 cp jenkins-backup.tar.gz s3://jenkinsbackup02092024/jenkins-backup.tar.gz

if aws if not in ubuntu/linux, follow this -

curl "https://awscli.amazonaws.com/awscli-exe-linux-x86\_64.zip" -o "awscliv2.zip"

apt install unzip

unzip awscliv2.zip

sudo ./aws/install

it will create a file in s3 bucket

S3 > bucket > jenkins file

Destroy the old jenkins file –

Terminate the instance

Create a new instance and install java, jenkins using shell script

Run jenkins using pub ip and through powershell, check if its running

Do not add password in jenkins

Add IAM role to this instance

Restore backup –

systemctl stop jenkins

systemctl status jenkins

goto /home/jenkins\_backup directory or create a directory if needed else copy in /home/ubuntu

install aws using above syntax

aws s3 cp s3://jenkinsbackup02092024/jenkins-backup.tar.gz jenkins-backup.tar.gz

check using ls

rm -rf /var/lib/jenkins [delete jenkins from current instance server]

tar -zxvf jenkins-backup.tar.gz -C / [restore backup to particular file]

systemctl start jenkins

systemctl status jenkins

jenkins restored successfully.

Check on browser – all jobs are restored back.

**Another method – but have not used yet…**

To create backups and restores in Jenkins, you can use a plugin like thinBackup:

* Log in to your Jenkins account
* Go to Manage Jenkins and select Plugins
* Search for the Thin Backup plugin
* Click Thin Backup and select Settings
* Configure the backup settings, including the directory where you want to store the backup
* Click Backup now to create a backup
* To restore, click the Restore button and select the backup you want to restore

You can also back up the entire Jenkins controller by backing up the $JENKINS\_HOME directory. To restore, copy the entire backup to the new system. However, some files in JENKINS\_HOME don't need to be backed up, so you can also back up specific directories and files. This will create smaller backups, but may make restoring the system more difficult.

tar Command to Compress Files in Linux. The Linux 'tar' stands for tape archive, which is used to create Archive and extract the Archive files. tar command in Linux is one of the important commands that provides archiving functionality in Linux.

**4) How to install Jenkins in Tomcat**

Answer:

Step 1: Install Java

sudo apt update -y && apt upgrade -y && apt install java

Step 2: Install Tomcat

cd /tmp

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

[Use wget to download the link that you copied from the Tomcat website]

install Tomcat to the /opt/tomcat directory

mkdir /opt/tomcat  
tar xzvf apache-tomcat-8\*tar.gz -C /opt/tomcat - strip-components=1

Step 3: Update Permissions

cd /opt/tomcat   
chown -R ubuntu:ubuntu /opt/tomcat

Step 4: Create a systemd Service File

sudo vi /etc/systemd/system/tomcat.service

Environment=JAVA\_HOME=/usr/lib/jvm/java-1.8.0-openjdk-amd64/jre  
Environment=CATALINA\_PID=/opt/tomcat/temp/tomcat.pid  
Environment=CATALINA\_HOME=/opt/tomcat  
Environment=CATALINA\_BASE=/opt/tomcat  
Environment='CATALINA\_OPTS=-Xms512M -Xmx1024M -server -XX:+UseParallelGC'  
Environment='JAVA\_OPTS=-Djava.awt.headless=true -Djava.security.egd=file:/dev/./urandom'ExecStart=/opt/tomcat/bin/startup.sh  
ExecStop=/opt/tomcat/bin/shutdown.shUser=ubuntu  
Group=ubuntu  
UMask=0007  
RestartSec=10  
Restart=always[Install]  
WantedBy=multi-user.target

When you are finished, save and close the file.

Reload the systemd daemon so that it knows about our service file:

$ sudo systemctl daemon-reload

Start the Tomcat service by typing:

sudo systemctl start tomcat

Double check that it started without errors by typing:

$ sudo systemctl status tomcat

Step 5 : Download jenkins stable version war file

cd /tmp  
wget https://get.jenkins.io/war-stable/2.346.1/jenkins.war

Deploy download jenkins war file into tomcat webapps folder

$ sudo cp jenkins.war /opt/tomcat/webapps/

Restart tomcat service

$ sudo systemctl restart tomcat

Hit the tomcat url in your web browser **http://ip-address:8080/jenkins**

5) Remote Job in Jenkins

Answer:

**Automation Project build-**

* Install plugins > **build pipeline, maven Integration, deploy to container**
* Install tomcat and tomcat-admin and define user and role
* Configure and maven > Tools configuration>Maven
* Create job (maven) - hello-world-test > git url and maven "test" (goal and option)
* Create job (maven) - hello-world-build> git url and maven "install" (goal and option)
* Build pipeline dashboard > click on + icon > name of pipeline > build pipeline view > create > select initial job (test) > ok.
* Create job (free style) - hello-world-deploy-test> (no need to define anything)
  + Build steps > execute shell – sleep 10 and pwd
* Create job (free style) - hello-world-deploy-prod> (no need to define anything)
* Define dependency on each job (as per sequence mentioned above)
* hello-world\_build> add post build action>Archive the artifact> \*\*/\*.war
* Install plugins > **Copy Artifact**
  + hello-world-build>General>Permission to Copy Artifact> hello-world-deploy-test, hello-world-deploy-prod
* hello-world-deploy-test>Build Steps>Copy artifact from other project>hello-world-build and Build Steps>execute shell> ls
  + hello-world-deploy-test>add post build action>Archive the artifact> \*\*/\*.war
* hello-world-deploy-test>add post build action>Deploy war/ear to a container>war/ear files
  + WAR/ear files > \*\*/\*.war
  + context path > /hello
  + Add container > tomcat 9xx
  + Add tomcat credentials > tomcat + secret
  + Set the credentials
  + Tomcat url > public ip:9090
* hello-world-deploy-test>add post build action>Deploy war/ear to a container>Context path> /app
* hello-world-deploy-test>add post build action>Deploy war/ear to a container>Container>Tomcat9 (login details) and tomcat URL - http://<tomcat URL>

Goto > /var/lib/jenkins/workspace/hello-world-build/target

mv \*.war hello.war

copy war file to /var/lib/tomcat10/webapps = with the name of hello.war

cp hello.war /var/lib/tomcat10/webapps

check it in browser using tomcat url/war file name

i.e. <http://public> ip//hello

**Tomcat installation –**

apt install tomcat10 -y = Install Tomcat

**>> tomcat and jenkins share the same port 8080**

**>> to change the tomcat port**

vi /etc/tomcat10/server.xml

[ how to search > esc / = n for next occurrence = r for entering/relacing char = use 9090]

service tomcat10 restart

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

**>> deploy war file using tomcat GUI**

restart tomcat using = service tomcat10 restart

check on browser : 3.90.180.236:9090

apt install tomcat10-admin -y

restart tomcat10 again = service tomcat10 restart

**>> assign role to tomcat**

Go to > cd /var/lib/tomcat10/conf

modify tomcat-users .xml > vi tomcat-users.xml

k > upward movement

j > downward movement

h > left movement

l > right movement

paste below code above </tomcat-users>

<role rolename="manager-gui"/>

<role rolename="manager-script"/>

<role rolename="manager-jmx"/>

<role rolename="manager-status"/>

<role rolename="host-manager-gui"/>

<role rolename="host-manager-script"/>

<user username="tomcat" password="tomcat" roles="manager-gui, manager-script, manager-jmx, manager-status, host-manager-gui, host-manager-script"/>

chown -R tomcat:tomcat /var/lib/tomcat10 > changing ownership

systemctl restart tomcat10

[manager-gui - allows access to the HTML GUI and the status pages

manager-script - allows access to the text interface and the status pages

manager-jmx - allows access to the JMX proxy and the status pages

manager-status - allows access to the status pages only]

check status of tomcat on browser > public ip:9090/manager/html

Enter username – tomcat

Password – secret

[<role rolename="manager-gui"/>

<user username="tomcat" password="tomcat" roles="manager-gui, manager-gui, manager-script, manager-jmx, manager-status"/>]

Vi /etc/tomcat10/server.xml

6) Webhook through Git

7) Jenkins Installation Script - Pending

8) Install old version of Jenkins and then upgrade it to a latest version

Answer:

**Important Jenkins files and directory’s in Linux**

1. **/usr/share/jenkins** — Jenkins binaries and the jenkins.war file are kept here.
2. **/var/log/jenkins** — Jenkins logs are kept in this place.
3. **/var/lib/jenkins** — This is an important directory of Jenkins. This directory contains all the details related Jenkins. In this directory it will stored jobs, workspace, users, nodes details, plugins and job configurations.
4. **/run/jenkins**— This directory contains PID of currently running Jenkins.

[pid file is a process identification file that stores the process ID (PID) of running processes.]

**Steps :**

1. Install old version of jenkins (during instance creating using shell script)

Check the version

cat /var/lib/jenkins/config.xml | grep '<version>' (currently it is 2.462.1)

or: jenkins --version

Stop Jenkins service

sudo systemctl stop jenkins //For SystemD Based Linux  
OR, sudo service jenkins stop //For Init Based Linux

2. Take Jenkins instance Backup for safety pre-caution

#Take backup of Jenkins home directory. In my case my Jenkins home directory is /var/lib/jenkins.

cd /var/lib  
tar -cvzf jenkins\_date.tar.gz jenkins/

mv jenkins\_date.tar.gz $HOME

**3. Take The Backup Of the Current Jenkins Version Binary**

#Take the backup of Jenkins current version binary using following commands:

cd /usr/share/java

mv jenkins.war jenkins.war.old

4. Download The New LTS Jenkins Version

#Download the new version of Jenkins WAR file using the following commands:

wget https://updates.jenkins-ci.org/latest/jenkins.war

5. Start The Jenkins Service

#Start the Jenkins service using the following command:

sudo systemctl start jenkins //For SystemD Based Linux  
OR, sudo service jenkins start //For Init Based Linux

6: Check The Jenkins Version

We need to check the Jenkins version using the following command

cat /var/lib/jenkins/config.xml | grep '<version>'

or: jenkins --version

Congratulations! You successfully upgrade the Jenkins. (2.474)

Or, execute the following command from the command line, to see the new version.

java -jar jenkins-cli.jar -s http://localhost:8080/ version

9) How to put https in Jenkins URL (Open Cert option)

Answer:

The default Jenkins installation runs on ports 8080 and 8443. Typically, HTTP/HTTPS servers run on ports 80 and 443, respectively.

Linux Basics & Shell Scripting

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

1) Passwordless SSH automate using shell script

2) If we put any number, the output should give 100 numbers after that using shell script. Script should ask the number and then when number is entered then it should 100 numbers following that number

3) If we put any number, it should print the table of that number using shell script

4) If I have a file with contents, the script should print the number of lines, words and characters in that file using shell script

5) Every evening 5'o clock script should run automatically all the files that are placed in one folder (source) should be copied to another folder (destination). after 30 mins all the files and folder should be deleted from the destination folder

6) How storage is mounted in Unix

7) Soft link and Hard Link

8) If we mount 15 GB, Will i see all storage or some storage and why

9) Iptables

10) Swap Memory

11) Cron Job

**1) Passwordless SSH automate using shell script**

Answer:

Create master and worker hostname with 2 diff servers

ssh -keygen

master pub file - ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAIBQyiVw6Tx59xMhTs1a5AqV5YsnInW2FR9L3xK/bBCGp root@master

goto worker : cd /home/.ssh or cd /root/.ssh

In master: cat id\_ed25519.pub or cat \*.pub

In worker directly goto

cat >> authorized\_keys

ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAIBQyiVw6Tx59xMhTs1a5AqV5YsnInW2FR9L3xK/bBCGp root@master

goto master –

ssh public ip of worker

started worker in master

2) If we put any number, the output should give 100 numbers after that using shell script. Script should ask the number and then when number is entered then it should 100 numbers following that number

3) If we put any number, it should print the table of that number using shell script

4) If I have a file with contents, the script should print the number of lines, words and characters in that file using shell script

5) Every evening 5'o clock script should run automatically all the files that are placed in one folder (source) should be copied to another folder (destination). after 30 mins all the files and folder should be deleted from the destination folder

6) How storage is mounted in Unix

**7) Soft link and Hard Link**

Answer:

**Links** in UNIX are pointers pointing to a file or a directory. Creating links is a kind of shortcuts to access a file.

There are two types of links :

1. Soft Link or Symbolic links
2. Hard Links

**Symbolic links** are not updated (they merely contain a string which is the path name of its target).

**Hard links** always refer to the source, even if moved or removed.

For example, if we have a file a.txt. If we create a hard link to the file and then delete the file, we can still access the file using hard link. But if we create a soft link of the file and then delete the file, we can’t access the file through soft link and soft link becomes dangling. Basically, hard link increases reference count of a location while soft links work as a shortcut (like in Windows).

**Hard Link**

* A hard link cannot be linked to a directory.
* ls -l command shows all the links with the link column shows number of links.
* Command to create a hard link is:
  + $ ln [original filename] [link name]

**Soft Link**

* ls -l command shows all links
* Removing soft link doesn’t affect anything but removing original file, the link becomes “dangling” link which points to non-existent file.
* If we change the name of the original file, then all the soft links for that file become dangling i.e. they are worthless now.
* A soft link can link to a directory.
* Command to create a Soft link is:
  + $ ln -s [original filename] [link name]

8) If we mount 15 GB, Will i see all storage or some storage and why

9) Iptables

10) Swap Memory

**11) Cron Job**

Answer:

A cron job is a Linux command used for scheduling tasks to be executed sometime in the future. This is normally used to schedule a job that is executed periodically.

For example, to send out a notice every morning. Some scripts, such as Drupal and WHMCS may require you to set up cron jobs to perform certain functions.

* Cron is a job scheduling utility present in Unix like systems.
* A cron job is an entry written into a table called the *cron table (crontab)*.
* The cron daemon (crond) looks for entries in the crontab to determine what jobs it should run, and when it should run them according to the specified schedule.

cron consists of two commands:

* **cron** or **crond**, which is the daemon that runs the scheduling utility
* **crontab**, which is the command that allows you to edit the cron entries for your jobs

**On Ubuntu command to install cron:**

$ sudo apt update && sudo apt install cron

**Enable cron:**

$ sudo systemctl enable cron

**Check if Cron is running, run the ps command:**

$ ps aux | grep cron

You should see an output like this:

root 617 0.0 0.0 9420 2800 ? Ss 17:00 0:00 /usr/sbin/cron -f

Once cron is running, it checks for crontab entries in the following files every minute:

* **/etc/crontab**
* **/var/spool/cron/crontabs/$USER** (where **$USER** is the currently logged-in user)

The first file, **/etc/crontab**, is a system-generated file containing shortcut commands designed to check for cron table entries in the following directories: **/etc/cron.hourly**, **/etc/cron.daily**, **/etc/cron.weekly**, and **/etc/cron.monthly**.

Docker

======

1) Difference between virtualization vs Containerization

2) Difference between Docker attach and Exec

3) Difference between Docker kill and Stop

4) Docker commands

5) Docker Swarm Commands

6) Docker monitoring commands

7) Docker log file location and home path location

8) Image has been downloaded and where it referenced

9) Install jenkins in docker. - Create a container with jenkins installed inside it

10) Install docker old version and upgrade to new version

11) Docker se container banaya registry image se then wo ek container ban gya, fir container ka URL ban gya and then tag and put your image on it

12) Push an image on Docker Hub. (Need to pull image, then create a tag on it and then push it back to docker)

13) Wo image jinke container nai hai, what are they called and command on how to find out those and how to delete those images (Dangling Images)

14) Image se container banaya and then index file create ki container mei, need to run that container on different server. (Will convert container to a file and then run on other server) docker export is the hint

15) Mere pass ek index file tmp mei hai and mai usko container mei daalna chahta hu.. th where will i put it and port and command to execute, your container created very good print hona chaiye jab container bane. Command cat should run inside container but should print outside it

16) Create Docker Swarm with 2 workers and shutdown the master and see if leader is relected or not

17) Docker swarm Installation and worker join installation script

18) Docker install without internet

19) Docker uninstall

**1) Difference between virtualization vs Containerization**

**Answer:**

**Virtualization vs. Containerization in Docker**

1. Virtualization:

* Concept: Virtualization involves creating multiple virtual machines (VMs) on a single physical server. Each VM runs its own operating system (OS) and shares the physical server's resources.
* Hypervisor: A hypervisor (like VMware, Hyper-V, or KVM) is used to create and manage VMs. The hypervisor sits between the hardware and the VMs, allocating resources such as CPU, memory, and storage.
* Resource Efficiency: VMs are resource-intensive because each VM includes a full OS, its own libraries, and applications. This can lead to increased overhead, as each VM requires significant resources to operate.
* Isolation: VMs provide strong isolation, as each VM operates independently with its own OS. If one VM fails or gets compromised, the other VMs remain unaffected.
* Boot Time: VMs typically have longer boot times because the entire OS needs to start up.
* Use Case: VMs are commonly used when you need to run multiple OSes on the same hardware or when strong isolation between different environments is required.

2. Containerization:

* Concept: Containerization involves packaging an application and its dependencies into a container that can run on any environment. Containers share the host OS kernel but have isolated user spaces.
* Docker: Docker is a popular platform for containerization. It allows developers to package applications into lightweight containers that can run consistently across different environments.
* Resource Efficiency: Containers are lightweight compared to VMs because they share the host OS and do not require a full OS for each container. This leads to better resource utilization and less overhead.
* Isolation: Containers provide process-level isolation. While not as strong as VM isolation, it is sufficient for many use cases. Containers share the host OS, so they are not fully isolated from each other like VMs.
* Boot Time: Containers have fast startup times because they don't need to boot a full OS, only the application and its dependencies.
* Use Case: Containers are ideal for microservices, DevOps practices, and scenarios where lightweight, portable, and scalable applications are required.

Key Differences:

* OS Dependency: VMs run full OSes, while containers share the host OS.
* Resource Consumption: VMs consume more resources due to full OS requirements; containers are more efficient.
* Isolation: VMs provide stronger isolation, but containers offer faster performance.
* Portability: Containers are more portable across different environments due to their lightweight nature.
* Startup Time: Containers start up faster than VMs.

**2) Difference between Docker attach and Exec**

Answer:

**1. docker attach:**

* **Purpose:** The docker attach command allows you **to connect to a running container's main process** and interact with its standard input, output, and error streams. It is used to view and interact with the terminal of a container that is already running.
* **Behavior:**
  + When you attach to a container, you are connected to its primary process (often the one specified in the CMD or ENTRYPOINT in the Dockerfile).
  + If the container was started with a terminal (TTY), you can interact with it directly as if you were inside that terminal.
  + If the container's main process exits, the docker attach session will end.
  + docker attach can only connect to a single process within a container, typically the one that was started first.
* **Use Case:** It is typically used to view the real-time output of a container or to provide input to a running container’s primary process. For example, you might use docker attach to interact with a shell session or a REPL environment running inside a container.
* **Limitations:**
  + Limited to attaching to the main process.
  + Exiting from docker attach can stop the container, depending on how it was started.

**2. docker exec:**

* **Purpose:** The docker exec command allows you to run a new command or start a new process inside a running container. This is useful for inspecting the container, debugging, or running additional commands without stopping the container's main process.
* **Behavior:**
  + When you use docker exec, it creates a new process within the container, independent of the main process.
  + You can specify the command you want to run inside the container. For example, you might use docker exec to start a bash shell (/bin/bash) or execute commands like ls, top, or any other command within the container.
  + The container continues to run its main process while the new command runs in parallel.
* **Use Case:** It is often used for administrative tasks such as inspecting the file system, checking process statuses, or running additional tools inside the container. It’s also used for troubleshooting by opening an interactive shell inside a container.
* **Flexibility:**
  + You can run multiple commands with docker exec, each starting a new process.
  + docker exec doesn’t interfere with the main process running in the container.

**Key Differences:**

* **Primary Function:**
  + docker attach: Connects you to the main process inside the container, allowing you to interact with it.
  + docker exec: Runs a new process inside the container without affecting the main process.
* **Use Case:**
  + docker attach: Used when you need to interact with or monitor the main process.
  + docker exec: Used when you need to run additional commands or inspect the container without stopping or interfering with the main process.
* **Impact on Container:**
  + docker attach: Exiting the attached session may stop the container, depending on how it's set up.
  + docker exec: Exiting the exec command doesn't stop the container; it only terminates the new process started by exec.

**3) Difference between Docker kill and Stop**

Answer:

**1. docker stop:**

* **Purpose:** The docker stop command is used to gracefully stop a running container. It allows the container's processes to exit cleanly by sending a termination signal.
* **Behavior:**
  + When you issue a docker stop command, Docker sends a **SIGTERM signal** to the main process inside the container.
  + The container's main process gets the chance to perform cleanup operations, close open files, and terminate gracefully.
  + If the process does not terminate within a specified timeout (default is 10 seconds), Docker will send a **SIGKILL signal** to forcefully stop the process.
* **Use Case:** Use docker stop when you want to shut down a container safely and give it time to clean up resources and save data. This is important for applications that need to close files, save state, or shut down network connections properly.
* **Graceful Shutdown:**
  + Provides an opportunity for the process to exit cleanly.
  + Helps prevent data corruption or incomplete transactions.

**2. docker kill:**

* **Purpose:** The docker kill command is used to forcefully stop a running container by immediately terminating its main process.
* **Behavior:**
  + When you issue a docker kill command, Docker sends a **SIGKILL signal** to the main process inside the container.
  + The SIGKILL signal immediately terminates the process without allowing it to perform any cleanup.
  + The process is stopped abruptly, and the container is shut down.
* **Use Case:** Use docker kill when you need to immediately stop a container, such as in cases where the container is unresponsive, stuck in a loop, or not shutting down properly with docker stop.
* **Forceful Shutdown:**
  + Does not allow the process to perform any cleanup.
  + Useful for emergency situations where the container must be stopped instantly.

**Key Differences:**

* **Signal Sent:**
  + docker stop: Sends a SIGTERM followed by SIGKILL after a timeout if the process doesn’t terminate.
  + docker kill: Sends a SIGKILL immediately.
* **Process Termination:**
  + docker stop: Gracefully stops the container, allowing processes to terminate cleanly.
  + docker kill: Forcefully stops the container without any cleanup.
* **Use Case:**
  + docker stop: Preferred when you want to shut down a container in an orderly manner.
  + docker kill: Used in situations where the container needs to be stopped immediately, regardless of the state of the processes inside.
* **Impact:**
  + docker stop: Safer for applications that need to save state or close resources properly.
  + docker kill: Can result in data loss or corruption if the container was performing critical operations.

**4) Docker commands**

Answer:

Here are some essential Docker commands, categorized by their functionality:

**Basic Docker Commands**

* **docker version**: Displays Docker version and system information.
* **docker info**: Shows detailed information about the Docker installation.
* **docker images**: Lists all Docker images on the system.
* **docker ps**: Lists running Docker containers.
* **docker ps -a**: Lists all containers, including stopped ones.
* **docker pull <image>**: Downloads a Docker image from Docker Hub.
* **docker run <image>**: Runs a Docker container from a specified image.
* **docker stop <container\_id>**: Stops a running container.
* **docker start <container\_id>**: Starts a stopped container.
* **docker restart <container\_id>**: Restarts a running or stopped container.
* **docker rm <container\_id>**: Removes a stopped container.
* **docker rmi <image\_id>**: Removes a Docker image.

**Container Management**

* **docker exec -it <container\_id> <command>**: Executes a command in a running container (e.g., to get a bash shell).
* **docker attach <container\_id>**: Attaches to a running container (similar to exec, but attaches to the main process).
* **docker logs <container\_id>**: Retrieves logs from a container.
* **docker inspect <container\_id>**: Provides detailed information about a container.
* **docker kill <container\_id>**: Forcefully stops a running container by sending a SIGKILL signal.

**Image Management**

* **docker build -t <image\_name> .**: Builds a Docker image from a Dockerfile in the current directory.
* **docker tag <image\_id> <new\_image\_name>**: Tags an image with a new name.
* **docker push <image\_name>**: Pushes an image to a Docker registry (e.g., Docker Hub).
* **docker save -o <path\_to\_output\_file> <image\_name>**: Saves an image to a tar archive.
* **docker load -i <path\_to\_input\_file>**: Loads an image from a tar archive.

**Volume Management**

* **docker volume ls**: Lists all Docker volumes.
* **docker volume create <volume\_name>**: Creates a new Docker volume.
* **docker volume rm <volume\_name>**: Removes a Docker volume.
* **docker run -v <volume\_name>:/path/in/container <image>**: Runs a container with a volume attached.

**Docker Compose Commands**

* **docker-compose up**: Starts all services defined in a docker-compose.yml file.
* **docker-compose down**: Stops and removes containers, networks, volumes, and images created by docker-compose up.
* **docker-compose ps**: Lists containers managed by Docker Compose.
* **docker-compose build**: Builds or rebuilds services in a docker-compose.yml file.
* **docker-compose logs**: Shows logs for services managed by Docker Compose.

**Miscellaneous**

* **docker system prune**: Removes all stopped containers, unused networks, dangling images, and build cache.
* **docker history <image\_id>**: Shows the history of an image (layers and commands used to build it).
* **docker stats**: Displays a live stream of resource usage statistics for running containers.
* **docker top <container\_id>**: Displays the running processes inside a container.
* **docker rename <old\_name> <new\_name>**: Renames a container.

**5) Docker Swarm Commands**

Answer:

Docker Swarm is Docker's native clustering and orchestration tool, which allows you to manage a cluster of Docker nodes as a single system. Below are essential Docker Swarm commands categorized by their functionality:

**Initialize and Manage a Swarm**

* **docker swarm init**: Initializes a new Swarm, turning the current node into the Swarm manager.
  + Example: docker swarm init --advertise-addr <MANAGER-IP>
* **docker swarm join**: Joins a node to an existing Swarm.
  + Example: docker swarm join --token <WORKER-TOKEN> <MANAGER-IP>:<PORT>
* **docker swarm leave**: Removes a node from the Swarm.
  + Example: docker swarm leave --force (used to force the manager to leave)
* **docker swarm update**: Updates the Swarm configuration, such as changing the dispatcher heartbeat.
  + Example: docker swarm update --task-history-limit 5

**Node Management**

* **docker node ls**: Lists all nodes in the Swarm, showing their status, availability, and roles.
* **docker node inspect <node-name>**: Provides detailed information about a specific node.
* **docker node update --availability <availability> <node-name>**: Updates a node's availability (options: active, pause, drain).
  + Example: docker node update --availability drain worker-node1
* **docker node promote <node-name>**: Promotes a worker node to a manager.
  + Example: docker node promote worker-node1
* **docker node demote <node-name>**: Demotes a manager node to a worker.
  + Example: docker node demote manager-node1
* **docker node rm <node-name>**: Removes a node from the Swarm.
  + Example: docker node rm worker-node1

**Service Management**

* **docker service create --name <service-name> <image>**: Creates and starts a new service.
  + Example: docker service create --name my-web --replicas 3 -p 80:80 nginx
* **docker service ls**: Lists all services running in the Swarm.
* **docker service inspect <service-name>**: Provides detailed information about a specific service.
  + Example: docker service inspect --pretty my-web
* **docker service scale <service-name>=<replica-count>**: Scales a service to the specified number of replicas.
  + Example: docker service scale my-web=5
* **docker service update <service-name>**: Updates a service's configuration, such as changing the image or the number of replicas.
  + Example: docker service update --image nginx:latest my-web
* **docker service logs <service-name>**: Displays the logs for a specific service.
  + Example: docker service logs my-web
* **docker service rm <service-name>**: Removes a service from the Swarm.
  + Example: docker service rm my-web

**Task Management**

* **docker service ps <service-name>**: Lists all tasks (containers) for a service.
  + Example: docker service ps my-web
* **docker task ls**: Lists all tasks running in the Swarm, across all services.
* **docker task inspect <task-id>**: Provides detailed information about a specific task.
  + Example: docker task inspect <task-id>
* **docker task logs <task-id>**: Displays logs for a specific task.
  + Example: docker task logs <task-id>

**6) Docker monitoring commands**

Answer:

Docker monitoring commands are essential for tracking the performance, resource usage, and overall health of Docker containers, services, and the Docker daemon. Below are some key Docker monitoring commands:

**Monitor Containers**

* **docker stats**: Provides real-time statistics (CPU, memory, network, and I/O usage) for all running containers.
  + Example: docker stats (for all containers)
  + Example: docker stats <container-id> (for a specific container)

**Inspect Containers**

* **docker inspect <container-id>**: Displays detailed information about a container, including its configuration, state, and resource usage.
  + Example: docker inspect <container-id>

**View Container Logs**

* **docker logs <container-id>**: Fetches and displays the logs of a specific container.
  + Example: docker logs <container-id>
  + Options:
    - **--tail <number>**: Show only the last <number> lines of logs.
    - **-f**: Follow log output in real-time.

**Monitor Events**

* **docker events**: Streams real-time events from the Docker daemon, such as container start, stop, create, or delete.
  + Example: docker events
  + You can filter events by type, container, image, etc.:
    - Example: docker events --filter 'event=stop'

**Resource Usage and Limits**

* **docker top <container-id>**: Displays the running processes inside a container, similar to the Linux top command.
  + Example: docker top <container-id>
* **docker stats --no-stream <container-id>**: Provides a one-time snapshot of the resource usage for a specific container.
  + Example: docker stats --no-stream <container-id>

**Disk Usage**

* **docker system df**: Displays the disk usage for Docker images, containers, volumes, and the build cache.
  + Example: docker system df

**Health Checks**

* **docker inspect --format='{{json .State.Health}}' <container-id>**: Displays the health check status of a container.
  + Example: docker inspect --format='{{json .State.Health}}' <container-id>
* **docker ps --filter health=unhealthy**: Lists all containers that are in an "unhealthy" state.
  + Example: docker ps --filter health=unhealthy

**Docker Daemon Status**

* **docker info**: Provides detailed information about the Docker daemon, including the number of containers, images, and the storage driver used.
  + Example: docker info

**Network Monitoring**

* **docker network inspect <network-name>**: Displays detailed information about a specific Docker network, including connected containers and configuration details.
  + Example: docker network inspect <network-name>

**Swarm Monitoring**

* **docker node ls**: Lists all nodes in a Docker Swarm cluster, showing their status and availability.
  + Example: docker node ls
* **docker service ps <service-name>**: Lists tasks related to a specific Swarm service, including their state and node assignment.
  + Example: docker service ps <service-name>

**Prune Unused Resources**

* **docker system prune**: Cleans up unused containers, networks, images (both dangling and unreferenced), and optionally, volumes.
  + Example: docker system prune -a --volumes

**7) Docker log file location and home path location**

Answer:

**Docker Log File Location in Ubuntu:**

On Ubuntu, Docker logs for containers are typically stored in the following path:

* **Log File Location for Containers:**
  + /var/lib/docker/containers/<container-id>/<container-id>-json.log

For example:

* + /var/lib/docker/containers/abc12345def6789ghi01234jkl5678mno/abc12345def6789ghi01234jkl5678mno-json.log

Each container has its own log file named <container-id>-json.log.

**Docker Home Path Location in Ubuntu:**

The Docker home path, where Docker stores images, containers, volumes, and other data, is located at:

* **Home Path Location:**
  + /var/lib/docker/

This directory contains all the necessary Docker data, including images, containers, volumes, and network configurations.

**8) Image has been downloaded and where it referenced**

Answer:

**9) Difference between docker info and docker version command.**

Answer:

**10) Install old docker version and update it to new version.**

Answer:

**9) Install jenkins in docker. - Create a container with jenkins installed inside it**

Answer:

**10) Install docker old version and upgrade to new version**

Answer:

**11) Docker > container using registry image. Make container > container URL > Add tag and put your image on it**

Answer:

**12) Push an image on Docker Hub. (Need to pull image, then create a tag on it and then push it back to docker)**

Answer:

**13) What do we call to the images who do not have container? What is the command to find those images? How to delete those images (Dangling Images)?**

Answer:

**14) Make a container using image. Create an index file inside the container. Need to run that container on different servers. (Will convert container to a file and then run on other servers) docker export is the hint**

Answer:

**15) Mere pass ek index file tmp mei hai and mai usko container mei daalna chahta hu.. th where will i put it and port and command to execute, your container created very good print hona chaiye jab container bane. Command cat should run inside container but should print outside it**

Answer:

**16) Create Docker Swarm with 2 workers and shutdown the master and see if leader is switched or not**

Answer:

**17) Docker swarm Installation and worker join installation script.**

Answer:

**18) Docker install without internet**

Answer:

**19) Docker uninstall**

Answer:

Ansible

=======

1) Docker install on worker nodes without going to each worker

2) Things to Learn full document

3) Jenkins install through ansible

4) Docker install through ansible.

5) Tomcat through ansible

6) MySQL through ansible

Big Assignment - Project

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

1) Created a dockerfile.

2) Docker file is placed on git.

3) Create job in Jenkins, integrate the job with git

4) Any changes in the docker file, then Jenkins job will run

5) Image is downloaded

6) Image ki Image create hui and pushed in docker hub

7) docker hub se download hui server pe

8) Check whether container is created or not

9) If container is created then delete the container and create a new one.

10) what we did for tomcat in Jenkins same we have to do for docker (Hint)

Simple question: How to change the jenkins home directory location?

By default it points to /var/lib/jenkins whereas I want it to point to /home/jenkins.