**Basic Assignment**

NOTE: Do not forget to see manual page using "--help" option in command when searching for options/commands for a particular task.

1. Install Docker, either on your native OS or on a VM. Make sure it runs. type "docker -v" to check if it's installed.

try below commands for help

docker --help ---> This command shows all available options and commands to work with images and containers

**ANSWER :**

C:\Users\lenovo>docker -v

Docker version 20.10.8, build 3967b7d

C:\Users\lenovo>docker --help

Usage: docker [OPTIONS] COMMAND

A self-sufficient runtime for containers

Options:

* --config string Location of client config files (default
  + - "C:\\Users\\lenovo\\.docker")
* -c, --context string Name of the context to use to connect to the
  + - daemon (overrides DOCKER\_HOST env var and
    - default context set with "docker context use")
* -D, --debug Enable debug mode
* -H, --host list Daemon socket(s) to connect to
* -l, --log-level string Set the logging level
  + - ("debug"|"info"|"warn"|"error"|"fatal")
    - (default "info")
* --tls Use TLS; implied by --tlsverify
* --tlscacert string Trust certs signed only by this CA (default
  + - "C:\\Users\\lenovo\\.docker\\ca.pem")
* --tlscert string Path to TLS certificate file (default
  + - "C:\\Users\\lenovo\\.docker\\cert.pem")
* --tlskey string Path to TLS key file (default
  + - "C:\\Users\\lenovo\\.docker\\key.pem")
* --tlsverify Use TLS and verify the remote
* -v, --version Print version information and quit
* Management Commands:
* builder Manage builds
* buildx\* Build with BuildKit (Docker Inc., v0.6.1-docker)
* compose\* Docker Compose (Docker Inc., v2.0.0-rc.1)
* config Manage Docker configs
* container Manage containers
* context Manage contexts
* image Manage images
* manifest Manage Docker image manifests and manifest lists
* network Manage networks
* node Manage Swarm nodes
* plugin Manage plugins
* scan\* Docker Scan (Docker Inc., v0.8.0)
* secret Manage Docker secrets
* service Manage services
* stack Manage Docker stacks
* swarm Manage Swarm
* system Manage Docker
* trust Manage trust on Docker images
* volume Manage volumes
* Commands:
* attach Attach local standard input, output, and error streams to a running container
* build Build an image from a Dockerfile
* commit Create a new image from a container's changes
* cp Copy files/folders between a container and the local filesystem
* create Create a new container
* diff Inspect changes to files or directories on a container's filesystem
* events Get real time events from the server
* exec Run a command in a running container
* export Export a container's filesystem as a tar archive
* history Show the history of an image
* images List images
* import Import the contents from a tarball to create a filesystem image
* info Display system-wide information
* inspect Return low-level information on Docker objects
* kill Kill one or more running containers
* load Load an image from a tar archive or STDIN
* login Log in to a Docker registry
* logout Log out from a Docker registry
* logs Fetch the logs of a container
* pause Pause all processes within one or more containers
* port List port mappings or a specific mapping for the container
* ps List containers
* pull Pull an image or a repository from a registry
* push Push an image or a repository to a registry
* rename Rename a container
* restart Restart one or more containers
* rm Remove one or more containers
* rmi Remove one or more images
* run Run a command in a new container
* save Save one or more images to a tar archive (streamed to STDOUT by default)
* search Search the Docker Hub for images
* start Start one or more stopped containers
* stats Display a live stream of container(s) resource usage statistics
* stop Stop one or more running containers
* tag Create a tag TARGET\_IMAGE that refers to SOURCE\_IMAGE
* top Display the running processes of a container
* unpause Unpause all processes within one or more containers
* update Update configuration of one or more containers
* version Show the Docker version information
* wait Block until one or more containers stop, then print their exit codes
  + docker images --help ---> This command shows all the avaialble options and commands to work with docker images
* C:\Users\lenovo>docker images --help
* Usage: docker images [OPTIONS] [REPOSITORY[:TAG]]
* List images
* Options:
* -a, --all Show all images (default hides intermediate images)
* --digests Show digests
* -f, --filter filter Filter output based on conditions provided
* --format string Pretty-print images using a Go template
* --no-trunc Don't truncate output
* -q, --quiet Only show image IDs
* docker ps --help ---> This command shows all the avaialble options and commands to work with docker containers
* C:\Users\lenovo>docker ps --help
* Usage: docker ps [OPTIONS]
* List containers
* Options:
* -a, --all Show all containers (default shows just running)
* -f, --filter filter Filter output based on conditions provided
* --format string Pretty-print containers using a Go template
* -n, --last int Show n last created containers (includes all
  + - states) (default -1)
* -l, --latest Show the latest created container (includes all
  + - states)
* --no-trunc Don't truncate output
* -q, --quiet Only display container IDs
* -s, --size Display total file sizes

2. Find a image from dockerhub of your choice(recommeded: nginx), don't use browser, pull the official image from dockerhub

**ANSWER :** C:\Users\lenovo>docker pull ubuntu

3. List all the available images in your machine/vm, make sure you see recently pulled image in the list.

**ANSWER :** C:\Users\lenovo>docker images -a

REPOSITORY TAG IMAGE ID CREATED SIZE

ubuntu latest fb52e22af1b0 2 days ago 72.8MB

jenkins 2.60.3 cd14cecfdb3a 3 years ago 696MB

4. Find out the "Full" ImageId of the image that you pulled and write it below.

**ANSWER :** C:\Users\lenovo>docker images -q

fb52e22af1b0

cd14cecfdb3a

C:\Users\lenovo>docker images ubuntu -q

fb52e22af1b0

5. Create a container of your image

* **ANSWER :** C:\Users\lenovo>docker run -it ubuntu
* root@d839f8b25003:/# ls
* bin boot dev etc home lib lib32 lib64 libx32 media mnt opt proc root run sbin srv sys tmp usr var
* root@d839f8b25003:/# exit
* C:\Users\lenovo>docker run -d -it ubuntu
* df4ab85325c2dcd1a44a220ae1fd2eb514a3702383a7605697df0c543efeb92d

6. List all the running containers .

**ANSWER :** C:\Users\lenovo>docker ps

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

df4ab85325c2 ubuntu "bash" 9 seconds ago Up 7 seconds romantic\_panini

7. List all the running and stopped containers

**ANSWER :** C:\Users\lenovo>docker ps -a

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

df4ab85325c2 ubuntu "bash" 9 seconds ago Up 7 seconds romantic\_panini

8. Find out the "Full" containerId of the container and write it below.

* **ANSWER** : docker ps -q
* df4ab85325c2

9. Find out how many image layers are used to build this image.

* **ANSWER** C:\Users\lenovo>docker history fb52e22af1b0
* IMAGE CREATED CREATED BY SIZE COMMENT
* fb52e22af1b0 2 days ago /bin/sh -c #(nop) CMD ["bash"] 0B
* <missing> 2 days ago /bin/sh -c #(nop) ADD file:d2abf27fe2e8b0b5f… 72.8MB

10. Get the Apache Tomcat 7 server image from the docker hub.

**ANSWER** C:\Users\lenovo>docker pull tomcat:7.0

11. Run the Apache Tomcat 7, I mean create a container of Apache Tomcat.

**ANSWER** docker run tomcat:7.0

C:\Users\lenovo>docker run <containerid>

12. Find out what is the IP Address of the Apache Tomcat Container that it is running on

**ANSWER**

C:\Users\lenovo>docker inspect 37638abf29ee

13. Which Port it is using?

* 8080

14. Try to access the Tomcat's home page from your machine/vm.

* Docker run -p 8888:8080 tomcat:7.0

15. What is the disk size of Apache Tomcat image?

* Docker system df -v

16. Find out list of all environment variables that is configured for tomcat image, can you see JAVA\_HOME and CATALINA\_HOME? What did you notice about it?

* C:\Users\Lenovo>docker exec 6d24ba63ca57 env
* PATH=/usr/local/tomcat/bin:/usr/local/openjdk-11/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
* HOSTNAME=6d24ba63ca57
* JAVA\_HOME=/usr/local/openjdk-11
* LANG=C.UTF-8
* JAVA\_VERSION=11.0.12
* CATALINA\_HOME=/usr/local/tomcat
* TOMCAT\_NATIVE\_LIBDIR=/usr/local/tomcat/native-jni-lib
* LD\_LIBRARY\_PATH=/usr/local/tomcat/native-jni-lib
* GPG\_KEYS=48F8E69F6390C9F25CFEDCD268248959359E722B A9C5DF4D22E99998D9875A5110C01C5A2F6059E7 DCFD35E0BF8CA7344752DE8B6FB21E8933C60243
* TOMCAT\_MAJOR=9
* TOMCAT\_VERSION=9.0.52
* TOMCAT\_SHA512=35e007e8e30e12889da27f9c71a6f4997b9cb5023b703d99add5de9271828e7d8d4956bf34dd2f48c7c71b4f8480f318c9067a4cd2a6d76eaae466286db4897b
* HOME=/root
* C:\Users\Lenovo>docker exec 4fae31f8fce0 env
* PATH=/usr/local/tomcat/bin:/usr/local/openjdk-11/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
* HOSTNAME=4fae31f8fce0
* JAVA\_HOME=/usr/local/openjdk-11
* LANG=C.UTF-8
* JAVA\_VERSION=11.0.12
* CATALINA\_HOME=/usr/local/tomcat
* TOMCAT\_NATIVE\_LIBDIR=/usr/local/tomcat/native-jni-lib
* LD\_LIBRARY\_PATH=/usr/local/tomcat/native-jni-lib
* GPG\_KEYS=48F8E69F6390C9F25CFEDCD268248959359E722B A9C5DF4D22E99998D9875A5110C01C5A2F6059E7 DCFD35E0BF8CA7344752DE8B6FB21E8933C60243
* TOMCAT\_MAJOR=9
* TOMCAT\_VERSION=9.0.52
* TOMCAT\_SHA512=35e007e8e30e12889da27f9c71a6f4997b9cb5023b703d99add5de9271828e7d8d4956bf34dd2f48c7c71b4f8480f318c9067a4cd2a6d76eaae466286db4897b
* HOME=/root

17. Find out which port is exposed for tomcat?

* 8888

18. Run multiple conntainers of tomcat on different port and access it's home page.

* Docker run -it -d -p 8888:8080 tomcat8.0
* Docker run -it -d -p 8889:8080 tomcat8.0

19. Pull ubuntu os from dockerhub, try to pull 2 images of ubuntu, Except the latest one.

* Docker pull ubuntu:xenial
* Docker pull ubuntu:rolling

20. Run the container of ubuntu in attached mode.

* Docker attach/start ubuntu:xenial

21. Run the container of another ubuntu in detached mode.

* Docker run -d ubuntu:rolling

22. Check how many ubuntu containers are running and stopped

23. Is the tomcat container running? If no, start one.

* Docker start tomcat

24. Check the logs, generated by tomcat container(don't forget to make request to tomcat's home page to see the log).

25. Check if ubuntu conatiner is running? If no, start one in attached mode to the terminal.

* Docker start ubuntu:xenial

26. Login as root user in ubuntu container

* Docker exec -it <container id> bash

27. Create a file with any name in root directory

* Ls
* Touch file

28. Install software of your choice in ubuntu container using "apt-get install"

* Apt-get update / apt-get install nginx
* Docker pull ubuntu
* Docker run -d -it ubuntu
* Apt-get update
* Apt-get install nginx

29. Now exit the ubuntu shell, are you back to your host machine, if not, come back to the host machine.

* exit

30. Check if the ubuntu container is running.

* Docker ps , No container is running

31. Create a new ubuntu container out of the same image as that previous container in attached mode.

* docker run ubuntu:xenial

32. Login as a root user

* Docker exec -it <container id> bash

33. Check if you can see the file created in previous container, you will not see the file as well as software that you installed in the previous container. Now kill this Container.

* Docker kill <container id>

34. Do you have the previous ubuntu container where you created the file and installed the software? If no reapeat step 25 to 29.

* Docker start ubuntu:xenial
* Docker exec -it <container id>
* Ls
* Touch file
* Apt-get update
* exit

35. Create an Image out of the existing container.

* Docker container run -it ubuntu bash
* Make a file inside the shell and check it using ls
* Docker container ls
* Docker container commit <containerid> <image name to be created>
* Docker image ls
* Image is created .

36. Now Create a Container out of this image and login into it to see if you can see the file and software installed by you in the previous container.

* Docker container ls (delete the one existing )
* Docker rm -f <containerid>
* Docker container run -it <image name which created earlier>
* Inside the shell : Ls ; cd tmp ; ls (we get the file created )

37. Do you have running tomcat container? If yes, Stop it and kill all tomcat container.

* Docker stop tomcat
* Docker kill <container id>

38. Create an index.html file with following code in it:-

<h1>This is Tomcat Container</h1>

Now, Start a ubuntu container, login into the ubuntu, install nginx, access nginx's home page from the container,

Now, come out of the ubuntu container and replace the existing index.html page of nginx with above index.html page that you created, Now access the nginx server from host machine, it should display the index.html that you have created.

**ANSWER :**

* docker run -d -p 8888:80 nginx
* mkdir -p docker-share/html
* ls
* cd html
* touch index.html
* vi index.html
* <h1>This is Tomcat Container</h1>
* docker stop d2c0127704200
* cd ..
* cd ..
* docker run -d -p 8888:80 -v ~/docker-share/html:/usr/share/nginx/html --hostname nginx\_home nginx

39. type below command:-

docker images --help

Now, try to run command that proves the concept of following three options:-

write atleast 1 command using each option above and prove their concepts as described in the --help.

* 1. -a : docker images -a ( -a, --all Show all images
* 2. -f : docker images -f “dangling=false”false = not associated with running container .
* 3. - q : docker images -q ( -q, --quiet Only show image IDs)

40. type below command:- docker ps –help

write atleast 1 command using each option above and prove their concepts as described in the --help.

Now, try to run command that proves the concept of following six options:-

1. -a : -a, --all Show all containers

Docker ps -a / docker ps -a –last 3

2.-f : filters output based on conditions provided

Docker ps -a -f “ID=”

3. -q : (only displays container Ids ) $docker ps -q

4. -n : (show n last created containers) docker ps -n

5. -l : (show the latest created container) docker ps -l

6. -s : (display total file sizes) docker ps -a -s

41. Type below command:-

docker --help

you will various sections of commands apart from options like "Managemnet Commands" and "Commands".

Write some texts below describing the use of "Management Commands".

Use each command mentioned below and prove its concepts as described in the --help desription. write what you have understood from the output of the command after its successful execution.

1. cp : $docker cp <container nam:/filename . > (to copy the file in docker )

2. create : $docker run <name> (Create a new container)

3. export : $docker export –output=”<name of the tar file>” <image name | container id> (this command is used to Docker image into a tar file in your local system )

4. history : $docker history <id> (Show the history of an image)

5. info : $docker info (Display system-wide information)

6. login : $docker login (this command is used to login to docker hub repo from command line.)

7. logout :$docker logout ( Log out from a Docker registry )

8. rename: $docker rename <oldname> <newname> (rename the existing file into new name)

9. save : Save one or more images to a tar archive

10. stats :$docker stats <containername|id> ( give container id, name , cpu, input/output, memory location,)

11. top : $docker container top <containerid> Display the running processes of a container

42. Kill all running container in one liner command.

43. Delete all images in one liner command.

$docker

44. Create a simple Dockerfile, build it and run it.

**ANSWER :**

* Step1 : Create a file named Dockerfile
* Cd Desktop
* mkdir Dockerfiles
* Cd DockerFiles
* Dir > Dockerfile
* Vim dockerfile
* Step2 : add instruction in Dockerfile
* FROM ubuntu
* RUN apt-get update
* CMD [“echo,” “hello from Docker file ”]
* Step3 : Build dockerfile to create image
* Docker build -t myimage1:1.0 .
* Docker images
* Step4: Run image to create container
* Docker run <id>
* Output : hello from Docker file

45. Create one or more Dockerfile that demostrate the following commands in Dockerfile (Write a PoC for each in one or more Dockerfile)

* FROM ubuntu
* MAINTAINER 9167109118 YASHASVI-SANJAY.PAWAR@CAPGEMINI.COM
* USER 9167109118 YASHASVI-SANJAY.PAWAR@CAPGEMINI.COM
* RUN apt -get update
* ENV abc="hello"
* CMD [ "echo" , " hello world" ]
* ENTRYPOINT [ "echo", "Hello, World" ]
* EXPOSE 80
* VOLUME /file1

1. USER >> The USER instruction sets the user name (or UID) and optionally the user group (or GID) to use when running the image and for any RUN, CMD and ENTRYPOINT instructions that follow it in the Dockerfile.

2. RUN >> The RUN instruction will execute any commands in a new layer on top of the current image and commit the results. The resulting committed image will be used for the next step in the Dockerfile.

3. ENV >> Environment variables (declared with the ENV statement) can also be used in certain instructions as variables to be interpreted by the Dockerfile.

4. CMD >> There can only be one CMD instruction in a Dockerfile. If you list more than one CMD then only the last CMD will take effect.

6. ENTRYPOINT >> An ENTRYPOINT allows you to configure a container that will run as an executable.

7. EXPOSE >> The EXPOSE instruction informs Docker that the container listens on the specified network ports at runtime. You can specify whether the port listens on TCP or UDP, and the default is TCP if the protocol is not specified.

8. VOLUME >> The VOLUME instruction creates a mount point with the specified name and marks it as holding externally mounted volumes from native host or other containers.

46. Dockerhub:-

Find a application you care about on docker hub.

Launch the container.

Install another application in it.

Save (commit) the image.

Upload that to docker hub in your account and share it with a colleague, ask them to use your image and run the container out of it in their machine/vm.

**ANSWER :**

* docker pull Ubuntu
* Docker run –d –it Ubuntu
* Docker exec –it <container id> bash
* Apt-get update
* Apt-get install ngnix
* Ans: docker commit container id (username/ngnix1<name you want to give>)
* Docker login
* docker push 9370713459/ngnix1:latest

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

**Advanced Assignment :-**

Assignment - Dockers and Containers

---Introduction

The goal of this assignment is to get you familiarized with dockers. You will learn about how to dockerize any application by building a simple client and server containers using dockers.

---What is to be done?

You will be building two containers using docker. These containers can be built starting from an official base image. Official Images has a list of different official base images. Start from the image that is suitable for you.

---Server :

Create a volume by name "servervol".

The server container will mount "servervol" in "/serverdata".

This container runs a server application which will create a file of size 1KB with random text data in "/serverdata" and then transfer the file to the client along with the checksum.

The server application itself can be built using any language you are comfortable with. But, the container should include all the packages that are required to run your application. Choose your base image wisely and install only the necessary packages.

The port on which the server runs must be specified as a command line argument when we run docker.

---Client:

Create a volume by name "clientvol".

The client container will mount "clientvol" in "/clientdata".

The client container runs an application that connects to the server, recieves the file that the server sends and saves it in "/clientdata".

Verify that the file is received properly at the clientside by verifying the checksum.

The client application again can be wriiten in any language that you are comfortable with, but the container should include all the necessary packages. Choose your base image wisely and install only necessary packages.

You need to create a user-defined network in docker and run both these containers on the network created. The containers should run these applications by default (i.e, on run command). Mention in README as to what should be done to get into the container shell instead of running the applications by default (i.e, on run command).

NOTE: You should be able to get into the shell of the client container to physically check if the file has been recieved.

---What to submit?

Create a new empty branch in your git repos named "assignment2".

You will be submitting the files that you use to build the server and client containers. Checkout the assignment2 branch and submit the files related to server and client in two folders named "server" and "client" respectively.

As you go about doing the assignment, you will learn that there are multiple ways to create volumes and to have the containers communicate to each other. So submit a README file that expains what method you use.

Create two scripts. one for the server("fileserver.sh") and the other for the client ("fileclient.sh") that has all the commands to build and run server and client containers respectively. These scripts should include the volume creation for client and server respectively. The server script should also include the creation of user-defined network before server runs.

The scripts and the README should be in the root of your branch (outside your server and client directories).

---What is tested?

Run the "fileserver.sh" script on one terminal. It should create the user-defined network, the "servervol", start the server and wait for the client connection.

Run "fileclient.sh" in another terminal. This script should find out the server's IP, create the "clientvol" and run the client container by specifying server's IP and port as command line parameters.

When the client container is run, a connection between the server and client is established and the file is transferred from the server to the client.

Your code and scripts will be tested on creation of volumes, proper running of server and client and file transfer.

You should make sure that the server can transfer any file with random text and the checksum verification succeeds on the client when it receives the file.

---Grade Distribution

Build and run server container - 30 points

Build and run client container - 30 points

Proper communication between the two - 15 points

Scripts to run the containers - 15 points

README file - 10 points

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

**Advanced Assignment 2**

---Introduction

The goal of this assignment is to get you familiarized with CI-CD Pipeline for Creating docker images after successfull build.

You already built 2 app(client and server) from the "Part 1" Assignment.

NOTE: You should be able to get the docker image for both Client and Server after successfull build using Jenkins Pipeline.

Your Client app and Server app must be in the github.

Whenever developer commits changes to client/server app in github, it should trigger a build on Jenkins.

Jenkins Pipeline should do maven clean and package

Jenkins Pipeline should also do the static-code analysis using (checkstyle/sonarqube)

Jenkins pipeline at the end should create docker image

On build failure, Jenkins pipeline should send an email to any emailId about the failure.

---Grade Distribution

Proper Build trigger - 20 points

Proper Declarative Jenkins file - 20 points

Clean, Package and Code Analysis - 20 points

Creating docker image on successful build - 20 points

Sending email on build failure - 20 points