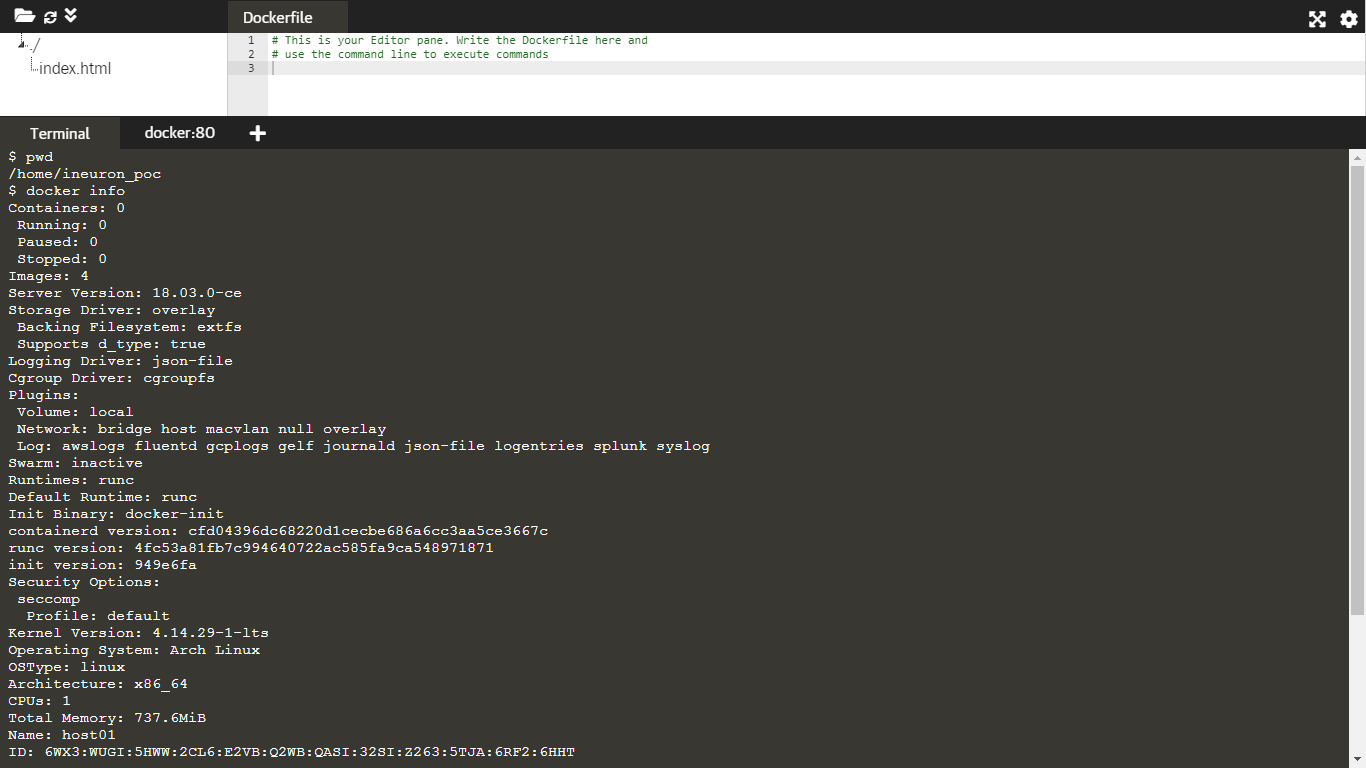
**Docker Assignment 7**

1. In Docker, where are docker volumes stored?

* In linux docker volumes are stored in **/var/lib/docker/volumes**

1. What is the purpose of the docker info command?

* The command gets detailed information about Docker installed on the host system. The information can be like what is the number of containers or images and in what state they are running and hardware specifications like total memory allocated, speed of the processor, kernel version, etc. Below is the example.



1. What are the minimum system requirements for Docker to run?

* Docker can run on both Windows and Linux, Mac platforms.
* Below are the minimum system requirement   
  Windows 11 64-bit: Home or Pro version 21H2 or higher, or Enterprise or Education version 21H2 or higher.
* Windows 10 64-bit: Home or Pro 2004 (build 19041) or higher, or Enterprise or Education 1909 (build 18363) or higher.
* Enable the WSL 2 feature on Windows. For detailed instructions, refer to the Microsoft documentation.
* The following hardware prerequisites are required to successfully run WSL 2 on Windows 10 or Windows 11:
* 64-bit processor with Second Level Address Translation (SLAT)
* 4GB system RAM
* BIOS-level hardware virtualization support must be enabled in the BIOS settings. For more information, see Virtualization.
* For Linux platforms, Docker can run on various Linux flavors such as Ubuntu >=12.04, Fedora >=19, RHEL >=6.5, CentOS >=6 etc.

1. What are the most widely utilized Dockerfile instructions?

* Let me first explain docker file and then followed with instructions used and with one eg.
* Docker can build images automatically by reading the instructions from a file called Dockerfile. A Dockerfile is a text document that contains all the commands a user could call on the command line to assemble an image. Dockerfile is set of instructions these file runs and execute commands which will create a image and then it is used a template which is used to run as a application inside a container
* Let us see that in depth with one example. For eg. docker file instructions are as below..create a docker file with following

**FROM** ubuntu

**RUN** apt-get update

**RUN** apt-get install-u python3 pythin3-pip

**RUN** pip3 install flask

**RUN** mkdir /pot/app

**WORKDIR** /opt/app

**COPY** . /opt/app

**ENTRYPOINT** FLASK\_APP=/opt/app/app.py flash run --host=0.0.0.0

Now we need to build with below commands open terminal and execute below commands

docker build . -t calc -> to build a image of above commands on top of ubuntu image and tag

1. When using docker-compose, how will you ensure that container 1 runs before container 2? Also, could you provide a sample Yaml example file?

* Docker-compose does not wait for any container to be “ready” before going ahead with the next containers. In order to achieve the order of execution, we can use:
* The **depends\_on** which got added in version 2 of docker-compose can be used as shown in a **sample docker-compose.yaml** file below  
    
  in below sample docker-compose yaml file please note indentation we need to follow

|  |
| --- |
| version: "2.4"  services:  backend:  build: .  depends\_on:  - db  db:  image: postgres |

* The docker-compose up command starts and runs the services in the dependency order specified. For the above example, the DB container is started before the backend.
* docker-compose up SERVICE\_NAME by default includes the dependencies associated with the service. In the given example, running docker-compose up backend creates and starts DB (dependency of backend).
* Finally, the command docker-compose stop also stops the services in the order of the dependency specified. For the given example, the backend service is stopped before the DB service.

1. How can I utilise Docker to run several applications in a single environment?

* By using docker-compose, you can run multiple containers using a single service. All docker-compose files uses yaml/yml language.
* Docker compose is a tool for defining and running multi-container Docker applications.
* With Compose, you use a YAML file to configure your application’s services.
* Then, with a single command, you create and start all the services from your configuration.
* we just need to defined the docker compose file which has multiple services, networks, and containers along with the volume mapping in a clean manner, and then we can just call the command docker-compose up. This can be in one environment or multi environment fo eg. dev, SIT, PIT, TT, Pre-Prod, Prod etc.
* we can go ahead with creating environment-specific docker-compose files of the name “docker-compose.{environment}.yml” and then based on the environment, we can set up and run the application.

1. What are the System commands for controlling Docker?

* **docker system df** -> this will Show docker disk usage
* **docker system events** -> this command is to get real time events from the server
* **docker system info** -> this command is to Display system-wide information
* **docker system prune** -> this command is used to Remove unused data,remove all the stopped containers, all the networks that are not used, all dangling images and all build caches.
* fyi - Apart from above docker system command mostly used general commands are like
* **docker run image** eg. **docker run nginx** -> docker run command is used to run a container from an image by specifying the Image ID or the Repository and/or Tag name.
* **docker ps** -> to list all running containers
* **docker ps -a** -> to list all containers whether in running or not and exited and its state
* **docker container ls** -> Like ps command, ls can also be used for listing containers.
* **docker stop container\_name\_or\_id** -> command is used to stop a running container. Here we need to put container name or ID along with this.
* **docker rm container\_name\_or\_id** -> this command removes a stopped or exited container.
* **docker exec –it container\_name\_or\_id command** -> We can use exec command to go inside a running container. This is useful to debug running containers or do some stuff within a container.
* **docker logs container\_name\_or\_id** -> In case a container is launched in detached mode, and we want to see its logs, we can use logs command to review its logs
* **docker container cp container\_name\_or\_id**:Source\_Path Dest\_Path ->To copy files between a container and localhost filesystem, you can use cp command.
* **docker container export container\_name\_or\_id** -> this is used to export the File system to archive as a tar
* **docker inspect container\_name\_or\_id** -> this is used to check detailed information about a container using inspect command
* **docker start container\_name** -> To start a stopped container, we can use the docker start command.
* **docker run -it --name container\_name image\_name** -> A container running with all its processes
* **docker pause container\_name** -> A container whose processes have been paused
* **docker unpause container\_name** --> A container whose processes have been unpaused
* **docker stop container\_name** -> To stop any container
* **docker rm container\_name** -> A container needs to be first stop then remove/delete
* **docker kill container\_name** ->We can kill containers with command

1. How do you scale your Docker containers, and what command do you use to do so?

with the help of docker swarm we can scale

Docker Swarm is native clustering for Docker. It turns a pool of Docker hosts into a single, virtual Docker host.

Docker Swarm serves the standard Docker API, any tool that already communicates with a Docker daemon can use Swarm to transparently scale to multiple hosts.

**step 1** : open the machine where you want to run your manager mode or do SSH

**step 2**: **docker swarm init --advertise-addr <MANAGER-IP>** -> to create a new swarm

Please keep a note that the output of this command will generate a new command which we need to keep handy as this is used in next steps

**step 3**: run command **docker info** and check the swarm state, there you can say a parameter to stanza like swarm: active

**step 4** : now let us view info about nodes run -> **command docker node ls**

Here \* indicted you are currently in that node

**step 5**: let lets see how to add say for eg. 2 more nodes to scale this docker

**step 6** : till this point we have create a swarm with manger node and we are ready to add the worker node

**step 7** : open the terminal on worker node

**step 8** : Now check above **step 2's output and copy that command** which is generated in output of step2 and run that command It will be like eg.   
 docker swarm join --token ID IP:portno.

**step 9** : Now open the second machine worker node where you want to add this worker node 2

As followed in step 8 copy the command generated from step 2

**step 10** : so till now swarm has one manager node and two worker node. we can verify that by command **docker node ls**

**step 11** : next step would be to deploy a service to swarm

**step 12** : open the manger node terminal and run

**docker service create --replicas 1 --name helloworld alpine ping docker.com**

**step 13**: Now check the list of running service **docker service ls**

Here you can see the service running into swarm

**step 14** : you can inspect the service -> **ocker service inspect --pretty helloworld**

**step 15** : to see which node is running a service -> **docker service ps helloworld**

**step 16** : Now let's scale the service in swarm

open terminal on manager node : **docker service scale <service-id>=<number of task>**

In our case let's say 8, **docker service scale helloworld=8**

**step 17** : finally to check the updated task list -> **docker service ps helloworld**

1. What exactly is CNM?

* CNM full form is Container Networking Model. It is a standard or specification from Docker, Inc. that forms the basis of container networking in a Docker environment. This docker’s approach provides container networking with support for multiple network drivers.
* CNM or Container Network Model is a specification that formally defines the steps needed to provide networking for containers while maintaining abstraction used to support multiple network drivers. CNM is built on three components, namely, sandbox, endpoint, and Network.

1. Explain what some of the more complex Docker commands do.

Let us try to see below few commands

* **docker run image** eg. **docker run nginx** -> docker run command is used to run a container from an image by specifying the Image ID or the Repository and/or Tag name.
* **docker ps** -> to list all running containers
* **docker ps -a** -> to list all containers whether in running or not and exited and its state
* **docker container ls** -> Like ps command, ls can also be used for listing containers.
* **docker stop container\_name\_or\_id** -> command is used to stop a running container. Here we need to put container name or ID along with this.
* **docker rm container\_name\_or\_id** -> this command removes a stopped or exited container.
* **docker exec –it container\_name\_or\_id command** -> We can use exec command to go inside a running container. This is useful to debug running containers or do some stuff within a container.
* **docker logs container\_name\_or\_id** -> In case a container is launched in detached mode, and we want to see its logs, we can use logs command to review its logs
* **docker container cp container\_name\_or\_id**:Source\_Path Dest\_Path ->To copy files between a container and localhost filesystem, you can use cp command.
* **docker container export container\_name\_or\_id** -> this is used to export the File system to archive as a tar
* **docker inspect container\_name\_or\_id** -> this is used to check detailed information about a container using inspect command
* **docker start container\_name** -> To start a stopped container, we can use the docker start command.
* **docker run -it --name container\_name image\_name** -> A container running with all its processes
* **docker pause container\_name** -> A container whose processes have been paused
* **docker unpause container\_name** --> A container whose processes have been unpaused
* **docker stop container\_name** -> To stop any container
* **docker rm container\_name** -> A container needs to be first stop then remove/delete
* **docker kill container\_name** ->We can kill containers with command
* To check what all images are there on host use command

**docker image ls** or **docker images**

* Command to remove image is **docker rmi image\_id**
* To focefully remove we can use the -f option. f means forcefully
* If you use the -f flag and specify the images then this will untags and removes all images that matchs the specified id.
* **docker rmi -f image\_id**
* Note that for a running contmainer image either we an stop and remove or we can remove forcefully with -f option
* docker build . -t calc -> to build a image of above commands on top of ubuntu image and tag
* docker images -> to check the image and there should be image with name as calc
* docker run -p 8080:5000 cal -> to do a port mapping to allow access external to container and run from this step output we get the IP which we can check in browser with port 8080
* If you need to push it somewhere then first login
* docker login -provide user id and password
* docker push calc - Now you can sign into your docker hub and check it repository from there
* The below command is used to remove all the stopped containers, all the networks that are not used, all dangling images and all build caches. It’s one of the most useful docker commands. **docker system prune**
* To save one or mutliple images into a tar archieve can be done woithe below comamnd and the syntax is **docker save -o exported\_name.tar container-name**
* -o means write a file to output/stdout
* **docker save image\_name > image\_name.tar**
* You can use gzip to save the image file and make the backup smaller.
* **docker save image\_name:latest | gzip > image\_name\_latest.tar.gz**
* Bind mount
* **docker run -v <path of external host> :<path of/inside container> app\_name/image\_name**
* **docker system df** -> this will Show docker disk usage
* **docker system events** -> this command is to get real time events from the server
* **docker system info** -> this command is to Display system-wide information

1. How do Docker object labels work?

* Docker object labels is a method for applying metadata to docker objects including, images, containers, volumes, network, swam nodes, and services.
* Docker object label is a key-value pair that is stored as a string. We can apply metadata using labels. Labels can be applied for Docker objects like images, containers, volumes, networks, local daemons, swarm nodes, and services. The key-value pair should be unique for each object. Labels are static for the entire lifetime of the object.