Student Name: Patil Ratnakar Netaji
Student Roll no.: 322050
Class: TY- B
Batch: B2

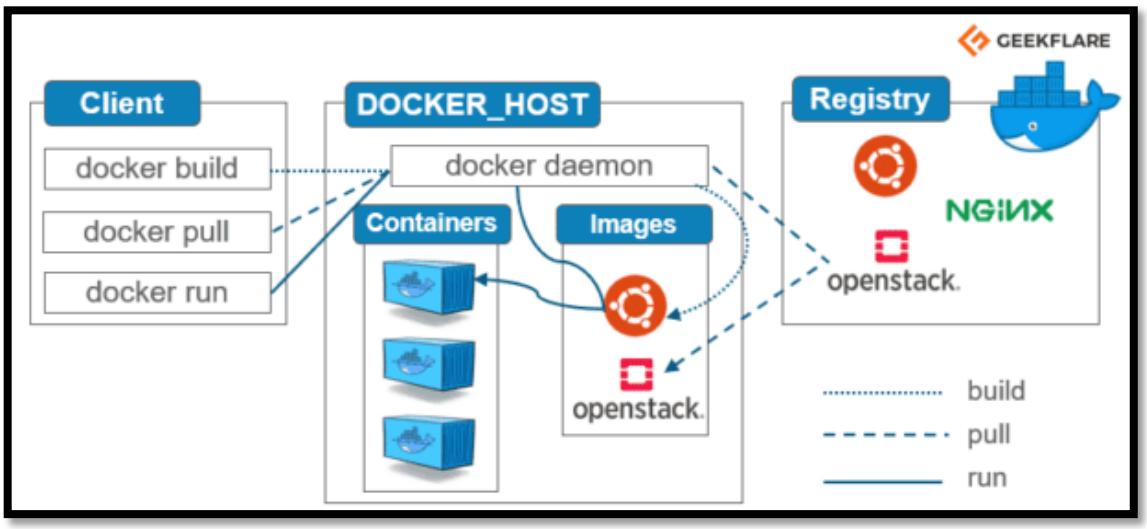
ASSIGNMENT 7

1. What is Docker?

Docker is a platform used to develop, deploy, and run applications in containers. Containers are lightweight, standalone, and portable packages that contain everything needed to run an application, including the code, runtime, system tools, libraries, and settings. Docker provides a way to package applications and their dependencies into a standardized unit, making it easier to build, ship, and run applications across different environments, such as development, testing, and production.

With Docker, developers can create containers based on predefined images, which are like templates containing all the necessary components. These containers can then be deployed on any Docker-compatible system, whether it's a developer's local machine, a cloud service, or an on-premises server. Docker also provides tools for managing containers, scaling applications, and automating deployment processes, making it popular in modern software development and DevOps practices.

2. Docker Architecture



Docker Architecture:

The Docker Client, Docker Host, Network and Storage components, and the Docker Registry/Hub make up the client-server architecture of Docker.

1. **Docker's Client:** Clients are used by Docker users to communicate with the platform. Any time a docker command is executed, the client transmits the command to the docked daemon, which executes it. Docker commands utilize Docker API. The Docker client has the option of interacting with several daemons.
2. **Docker Host:** Applications may be executed and operate in a full environment thanks to the Docker host. Images, Containers, Networks, and Storage are included in it. It also includes the Docker daemon. The daemon, which manages all container-related operations, accepts commands from the CLI or the REST API, as was already explained. To administer its services, it can also talk to other daemons.

Docker Objects:

1. **Image**: A Docker container may be created using an image, a read-only template. Frequently, a picture is derived from another image and then modified somewhat. You might create an image based on the Ubuntu image, for instance, but install your program, the Apache web server, and the configuration information required to execute it.
2. **Containers**: A private container registry may be used to distribute container images amongst teams inside an organization. A public registry like Docker Hub can be used to share them with the entire world. Images are important to the Docker experience because they allow developers to collaborate in previously impossible ways.

Networks: All separated containers interact with one another using Docker networking. Five network drivers make up the majority of docker's drivers.

1. **Bridge:** The container comes with Bridge as its default network driver. When several Docker instances communicate with the same Docker host, it is utilized.
2. **Host**: The resources that are now executing on your local machine may be seamlessly integrated with Docker thanks to this driver. Low-level IP tunneling and data link layer encryption between Docker programs running on various endpoints are provided by relying on the built-in network capabilities of your computer.
3. **None:** All networking is disabled with this driver.
4. **Overlay:** Containers may connect with one another thanks to this kind of software-defined networking technology. You must first construct a virtual bridge on one host to link it to an external host, and then you must set up an overlay network. Additionally, you will need to set up the overlay network and authorize access from one side to the other.
5. **macvlan:** The macvlan driver may be used to give containers an address and make them look like real devices. Its use of MAC addresses instead of IP addresses to route communication across containers makes it distinct from other solutions. Use this network when you want the containers to appear as physical objects, such as during a VM setup migration.

3. Difference between Docker and Virtual machine

Docker	Virtualization
It boots in a few seconds.	It takes a few minutes for VMs to boot.
Pre-built docker containers are readily available.	Ready-made VMs are challenging to find.
Docker has a complex usage mechanism consisting of both third-party and docker-managed tools.	Tools are easy to use and more straightforward to work with. third-party.
Limited to Linux.	Can run a variety of guest OS.
Dockers make use of the execution engine.	VMs make use of the hypervisor.
It is lightweight.	It is heavyweight.
Host OS can be different from container OS.	Host OS can be different from guest OS.
Can run many docker containers on a laptop.	Cannot run more than a couple of VMS on an average laptop.
Docker can get a virtual network adapter. It can have separate IPs ad Ports.	Each VMS gets its virtual network adapter.
Sharing of files is possible.	Sharing library and files are not possible.
Lacks security measures.	Security depends on the hypervisor.
A container is portable.	VMS is dependent on a hypervisor.
It allows running an application in an isolated environment known as a container	It provides easiness in managing applications, recovery mechanisms, and isolation from the host operating system

4. Docker Commands:

1. docker run: Start a new container from an image.

   - Example: `docker run -d -p 8080:80 nginx` (Starts an Nginx web server container in detached mode, mapping port 8080 on the host to port 80 on the container.)

2. docker build: Build a Docker image from a Dockerfile.

   - Example: `docker build -t myapp .` (Builds a Docker image named "myapp" using the Dockerfile in the current directory.)

3. docker ps: List running containers.

   - Example: `docker ps` (Lists all running containers along with their IDs, names, and statuses.)

4. docker images: List locally available Docker images.

   - Example: `docker images` (Lists all Docker images available locally on the system.)

5. docker pull: Pull an image or a repository from a registry.

   - Example: `docker pull ubuntu:latest` (Pulls the latest version of the Ubuntu image from Docker Hub.)

6. docker stop: Stop one or more running containers.

   - Example: `docker stop container_name` (Stops the container with the specified name.)

7. docker rm: Remove one or more containers.

   - Example: `docker rm container_id` (Removes the container with the specified ID.)

8. docker rmi: Remove one or more images.

   - Example: `docker rmi image_id` (Removes the image with the specified ID.)

9. docker exec: Run a command in a running container.

   - Example: `docker exec -it container_name bash` (Opens an interactive shell inside the running container.)

10. docker-compose: Define and run multi-container Docker applications using a YAML file.

- Example: `docker-compose up -d` (Starts all services defined in the docker-compose.yml file in detached mode.)

5. Docker-Compose and Docker-swarm.

Docker Compose and Docker Swarm are both tools used in the realm of container orchestration, but they serve different purposes and are designed for different scales of deployment.

1. Docker Compose:

- **Purpose**: Docker Compose is used for defining and running multi-container Docker applications. It allows you to define a multi-container application in a single file (typically named `docker-compose.yml`) and then use that file to spin up all the services comprising your application with a single command.
- **Usage**: Developers often use Docker Compose to define local development environments, as well as to deploy small-scale applications where a full-fledged orchestration system like Kubernetes might be overkill. With Docker Compose, you can define the services, networks, and volumes needed for your application in a declarative manner, making it easy to manage and replicate your development environment across different machines.

2. Docker Swarm:

- **Purpose**: Docker Swarm is a native clustering and orchestration tool provided by Docker. It allows you to deploy and manage a cluster of Docker hosts as a single virtual system. With Swarm, you can distribute containerized applications across multiple nodes, scale them up or down as needed, and ensure high availability and fault tolerance.
- **Usage**: Docker Swarm is used for orchestrating production-grade containerized applications at scale. It provides features such as service discovery, load balancing, rolling updates, and health checks, making it suitable for deploying and managing complex applications across a cluster of Docker hosts. Swarm is simpler to set up and use compared to some other orchestration platforms like Kubernetes, making it a good choice for organizations looking for container orchestration without the complexity.

6. Docker file, Docker Image and Running Instance:

index.html

```
Web-app > index.html > ...
  2   <html class="no-js" lang="en">
  4     <head>
 16       <!-- title of site -->
 17       <title>CarVilla</title>
 18
 19       <!-- For favicon png -->
 20       <link rel="shortcut icon" type="image/icon" href="assets/logo/favicon.png"/>
 21
 22       <!--font-awesome.min.css-->
 23       <link rel="stylesheet" href="assets/css/font-awesome.min.css">
 24
 25       <!--linear icon css-->
 26       <link rel="stylesheet" href="assets/css/linearicons.css">
 27
```

Dockerfile

```
Web-app > Dockerfile > FROM
1  FROM httpd:2.4
2  COPY . /usr/local/apache2/htdocs/
3
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\Ratnakar\Desktop\TY- Sem 2\Cloud Computing\Assg\Assg7> cd .\Web-app\
PS C:\Users\Ratnakar\Desktop\TY- Sem 2\Cloud Computing\Assg\Assg7\Web-app> docker build -t web-app .
[+] Building 17.0s (8/8) FINISHED
 => [internal] load build definition from Dockerfile
 => => transferring dockerfile: 86B
 => [internal] load metadata for docker.io/library/httpd:2.4
 => [auth] library/httpd:pull token for registry-1.docker.io
 => [internal] load .dockerignore
 => => transferring context: 2B
 => [1/2] FROM docker.io/library/httpd:2.4@sha256:b19cace6539a05579c55fda6be0a873c1d2c2e7392e7c08805141f79852ab07b
 => => resolve docker.io/library/httpd:2.4@sha256:b19cace6539a05579c55fda6be0a873c1d2c2e7392e7c08805141f79852ab07b
 => => sha256:fa0099f1c09d7d1dc852123c9456a23436df95715812df3db8b960ef6609e288 8.02kB / 8.02kB
 => => sha256:13808c22b207b066ef43572e57e4fb8c6172e887dd9a918c089a174a19371b7a 29.13MB / 29.13MB
 => => sha256:4f4fb700ef54461cfa02571ae0db9a0dc1e0cdb5577484a6d75e68dc38e8acc1 32B / 32B
 => => sha256:b19cace6539a05579c55fda6be0a873c1d2c2e7392e7c08805141f79852ab07b 10.16kB / 10.16kB
 => => sha256:6834aeaa0023c674c53a8891ce77a363579ae9a9679222f9634f95aa72c5781b 2.10kB / 2.10kB
 => => sha256:6e9a8835eae400bbb7b5572cc9aeea0a187e7216dd2def605c432cbe1d4955c5 145B / 145B
 => => sha256:b927d001db705e0c4192a7c5ea2ca65c7c0f4858349d9a785e40e044d617724b 4.20MB / 4.20MB
 => => sha256:559cc51378ed1226b3d18ddb9d2149586a427338728bc7bd567bda03ca43b590 26.03MB / 26.03MB
 => => sha256:d2b091e65160919e8aa45ccc4e2187a65be454ce437692ec811e3fe47386bd4f 292B / 292B
 => => extracting sha256:13808c22b207b066ef43572e57e4fb8c6172e887dd9a918c089a174a19371b7a
 => => extracting sha256:6e9a8835eae400bbb7b5572cc9aeea0a187e7216dd2def605c432cbe1d4955c5
 => => extracting sha256:4f4fb700ef54461cfa02571ae0db9a0dc1e0cdb5577484a6d75e68dc38e8acc1
 => => extracting sha256:b927d001db705e0c4192a7c5ea2ca65c7c0f4858349d9a785e40e044d617724b
 => => extracting sha256:559cc51378ed1226b3d18ddb9d2149586a427338728bc7bd567bda03ca43b590
                                                                                    0.0s
 => => writing image sha256:7c79a211d6a0f10e8ed3924730ac79dd7d98d8c46bf9336c6b0493d47712a020
 => => naming to docker.io/library/web-app

What's Next?
  View a summary of image vulnerabilities and recommendations → docker scout quickview
PS C:\Users\Ratnakar\Desktop\TY- Sem 2\Cloud Computing\Assg\Assg7\Web-app> docker images
REPOSITORY   TAG       IMAGE ID       CREATED         SIZE
web-app      latest    7c79a211d6a0   8 seconds ago   152MB
PS C:\Users\Ratnakar\Desktop\TY- Sem 2\Cloud Computing\Assg\Assg7\Web-app>
```

```
PS C:\Users\Ratnakar\Desktop\TY- Sem 2\Cloud Computing\Assg\Assg7\Web-app> docker run -itd -p 80:80 --name web-app web-app
56a95c5eae3f16bd07ac62a466c59792fbaddae8791a6a8b7d5bae50d097692d
PS C:\Users\Ratnakar\Desktop\TY- Sem 2\Cloud Computing\Assg\Assg7\Web-app> docker ps
CONTAINER ID   IMAGE     COMMAND              CREATED          STATUS          PORTS                NAMES
56a95c5eae3f   web-app   "httpd-foreground"   14 seconds ago   Up 12 seconds   0.0.0.0:80->80/tcp   web-app
```

Running Instance: