Q1) Define Docker Container. Explain Docker container with its architecture. *(6 Marks)*

Definition of Docker Container:

A **Docker Container** is a lightweight, standalone, and executable package of software. It contains everything required to run an application such as:

- Application code
- Runtime environment
- System tools
- Libraries
- Configuration files

The main advantage of Docker containers is that they solve the problem of **"It works on my machine."** Without Docker, an application may run on one computer but fail on another due to different dependencies. With Docker, the container ensures the application runs exactly the same way on any machine, whether it is a laptop, a testing server, or a production server.

Docker containers are fast to start, easy to share, portable, and isolated from each other. This makes them ideal for modern software development, cloud computing, and DevOps.

Architecture of Docker:

The Docker architecture is built on the **client-server model**. It has the following components:

1. **Docker Client**
   - This is the interface that users interact with.
   - It uses commands such as `docker run`, `docker build`, `docker ps`, etc.
   - The client sends these commands to the Docker Daemon using REST API.
2. **Docker Daemon (dockerd)**
   - The daemon runs as a background service on the host machine.
   - It listens to requests from the Docker client and performs actions like building, running, and distributing containers.
   - It communicates with other daemons if needed.
3. **Docker Images**
   - These are templates or blueprints for containers.
   - An image contains all the instructions (from the Dockerfile) needed to create a container.
   - For example, an **NGINX image** contains the NGINX web server code, dependencies, and configuration.
4. **Docker Containers**
   - Containers are the running instances of images.
   - They are isolated environments that run the application, but share the kernel of the host operating system.
   - Multiple containers can run on the same machine without interfering with each other.
5. **Docker Registry**
   - Registries are used to store and distribute Docker images.
   - The most popular registry is **Docker Hub** where users can push and pull images.
   - Private registries are also used in organizations.

Summary of Flow:

- The **user** runs commands using **Docker Client**.
- The **Docker Daemon** executes these commands.
- **Images** are pulled from a **Registry**.
- **Containers** are created from those images and run the applications.

This architecture makes Docker powerful, flexible, and portable.

Q2) Commands to deploy Docker container (Container name: NGINX, Image name: Registry1). *(6 Marks)*

Docker provides different commands to create, manage, and monitor containers.

Here are the step-by-step commands for the given task:

a) To create and run a container "NGINX" from an image "Registry1":

bash Copy Edit

```bash
docker run --name NGINX Registry1
```

- This command creates a new container called **NGINX** using the image **Registry1**.
- If the image is not present locally, Docker will first pull it from the registry.
- Once created, the container starts running immediately.

b) To remove a stopped container "NGINX":

bash Copy Edit

```bash
docker rm NGINX
```

- This permanently deletes the container **NGINX** from the system.
- Note: The container must be in a stopped state before removal.
- If you try to delete a running container, Docker will give an error.

c) To open a shell inside a running container "NGINX":

bash Copy Edit

```bash
docker exec -it NGINX sh
```

- This command allows the user to enter inside the container environment.
- It opens a shell (like a Linux terminal) inside the container.
- From here, you can run commands as if you were inside the container's operating system.

d) To inspect a running container "NGINX":

bash Copy Edit

```bash
docker inspect NGINX
```

- This provides detailed information about the container in JSON format.
- It shows details such as container ID, network settings, IP address, mounted volumes, resource limits, and environment variables.

```bash
docker inspect NGINX
```

- This provides detailed information about the container in JSON format.
- It shows details such as container ID, network settings, IP address, mounted volumes, resource limits, and environment variables.
- This is useful for debugging and analysis.

Q3) Commands for managing image “Registry2”. *(3 Marks)*

a) To build the image “Registry2” from Dockerfile:

```bash
docker build -t Registry2 .
```

- This command reads the instructions from the **Dockerfile** in the current directory (`.`).
- It then creates a new Docker image named **Registry2**.

---

b) To build the image “Registry2” from Dockerfile without cache:

```bash
docker build -t Registry2 . --no-cache
```

- Normally, Docker uses cached layers to speed up the build process.
- With the `--no-cache` option, Docker ignores the cache and builds everything from scratch.
- This ensures that the latest versions of dependencies and instructions are applied.

bash Copy Edit

```bash
docker rmi Registry2
```

- This removes the image **Registry2** from the system.
- It is useful when you no longer need the image or want to free up disk space.
- If the image is used by a container, you must remove the container first.