Experiment -1.2

Student Name: Ayush Pandey
UID: 22BDO10038
Branch: CSE-DevOps
Section/Group: 22BCD-1/A
Semester: 5th.
Date of Performance:05/08/24
Subject Name: Docker & Kubernetes
Subject Code: 22CSH-343

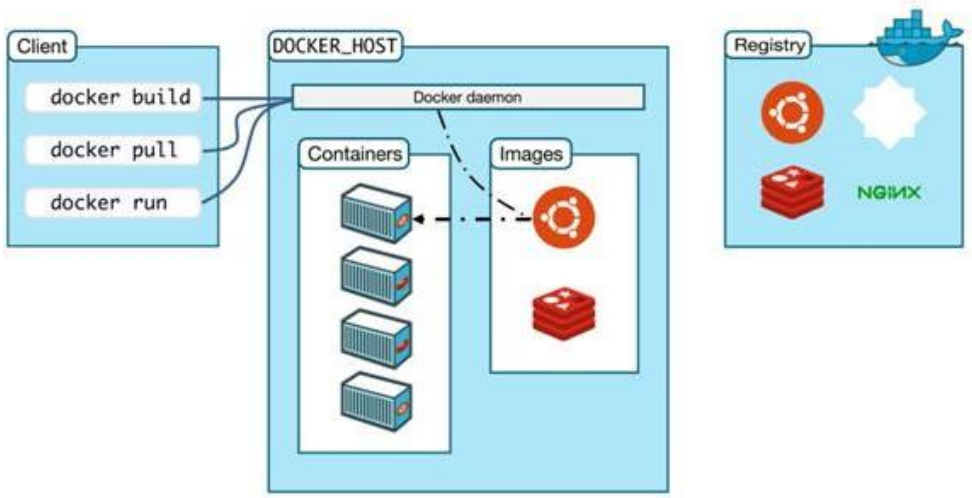
1. **Aim/Overview of the practical:** To understand the Container Lifecycle Management with Docker.

   - Docker Images.
   - Container.
   - Docker Repository.
   - Docker Commands.
   - DockerFile.

2. Apparatus: Docker.

3. Procedure:

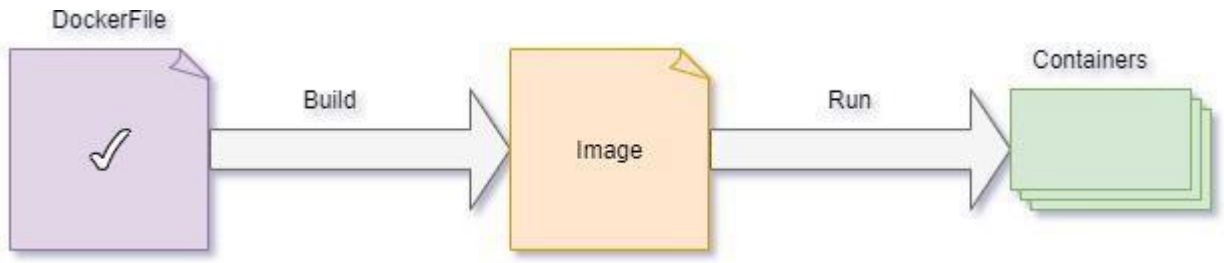
- **Docker:** Docker is an open-source platform based on Linux containers for developing and running applications inside containers. Docker is used to deploy many containers simultaneously on a given host. Containers are very fast and lightweight because they don't need the extra load of a hypervisor as they run directly within the host machine's kernel.



- **Docker Images:** Docker images are read-only templates that contain instructions for creating a container. A Docker image is a snapshot or blueprint of the libraries and dependencies required inside a container for an application to run.

  Docker images are built using the Docker file which consists of a set of instructions that are required to containerize an application.

  A Docker image contains application code, libraries, tools, dependencies and other files needed to make an application run.



- **Components of Docker Image:**

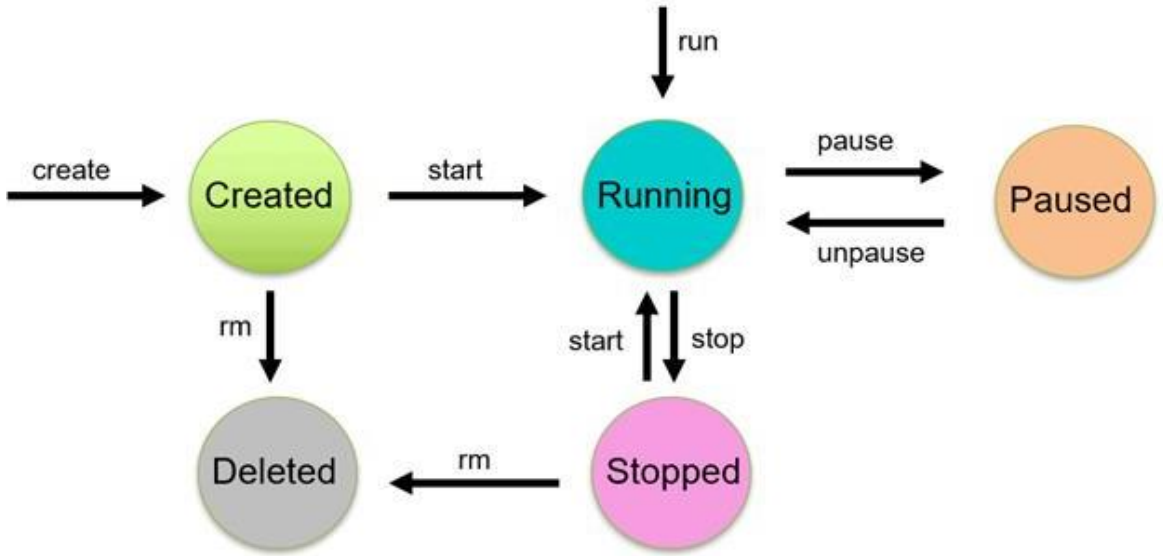
  (1). Layers: Immutable filesystem layers stacked to form a complete image.
  (2). Base Image: The foundational layer, often a minimal OS or runtime environment.
  (3). Dockerfile: A text file containing instructions to build a Docker image.
  (4). Image ID: A unique identifier for each Docker image.
  (5). Tags: Labels used to manage and version Docker images.

There are five major components in the Docker architecture:
1. Docker Daemon: listens to Docker API requests and manages Docker objects such as images, containers, networks and volumes.
2. Docker Clients: With the help of Docker Clients, users can interact with Docker. Docker client provides a command-line interface (CLI) that allows users to run, and stop application commands to a Docker daemon.
3. Docker Host: provides a complete environment to execute and run applications. It comprises of the Docker daemon, Images, Containers, Networks, and Storage.
4. Docker Registry: stores Docker images. Docker Hub is a public registry that anyone can use, and Docker is configured to use images on Docker Hub by default. You can run your own registry on it.
5. Docker Images: are read-only templates that you build from a set of instructions written

in Dockerfile. Images define both what you want your packaged application and its dependencies to look like what processes to run when it's launched.

- **Container:** A Docker container image is a lightweight, standalone, executable package of software that includes everything needed to run an application: code, runtime, system tools, system libraries and settings**.**

  Containers share resources with other containers in the same host OS and provide OS-level process isolation. They don't contain a guest OS for each container and rely on the underlying OS kernel, which makes the containers lightweight.



- **Create Containers**

Using the docker create command will create a new Docker container with the specified docker image.

$ docker create --name <container name> <image name>

- **Start Container**

To start a stopped container, we can use the docker start command.
$ docker start <container name>.

- **docker create - -name:** This command is used to create a container using a particular image.
- **docker run -dit --name:** This command is used to create and start a container from a docker image in background and interactive mode.

- **docker ps -a:** This command is used to see all the containers (running / stopped) present in the system.
- **docker pause:** This command is used to pause the state of a running container.
- **docker unpause:** This command is used to resume the state of a particular container.
- **docker start $(docker ps -aq)**: This command is used to start all the containers in the system which are not currently in the start state.
- **docker login:** This command is used to login to the hub.docker.com so that we can push our image from our system to the docker hub.

➢ **Docker Repository:**

A Docker repository is a collection of Docker images, often versions of the same application or service, that are stored together as tags to create a container's file system. Repositories are essential for managing and distributing Docker images, and you can use them to share images with your team, customers, or the Docker community.

You can organize images into repositories within a Docker registry, which is a centralized location that stores and manages container images. Think of a repository like a folder where you organize your images based on projects. Images can be public or private.

- **Public**: Anyone can download images from public repositories.
- **Private:** Restricts access to the repository creator or members of its organization

➢ **Docker Commands:**

- **docker --version**
  Displays the installed Docker version.
- **docker info**
  Shows detailed information about Docker installation and configuration.
- **docker run [OPTIONS] IMAGE [COMMAND] [ARG...]**
  Runs a command in a new container. Example: docker run -d nginx (runs Nginx container in detached mode).

- **docker ps**

Lists running containers. Use docker ps -a to show all containers (including stopped ones).

- **docker stop CONTAINER_ID**

Stops a running container. Example: docker stop [CONTAINER_ID].

- **docker start CONTAINER_ID**

Starts a stopped container. Example: docker start [CONTAINER_ID].

- **docker restart CONTAINER_ID**

Restarts a container. Example: docker restart [CONTAINER_ID].

- **docker rm CONTAINER_ID**

Removes a container. Example: docker rm [CONTAINER_ID]. Use -f to force remove a running container.

➢ **Docker File:**
A Dockerfile is a script-like text document that contains a series of instructions used to create a Docker image. Each instruction in the Dockerfile creates a layer in the image, which can include setting up an environment, installing software, copying files, or defining entry points.

Key Concepts of a Dockerfile:

- **Base Image**: Specifies the starting point for your Docker image. It could be a minimal Linux distribution like alpine, or a more complex image with pre-installed software, like node or python.

- **Layers**: Each instruction in the Dockerfile adds a layer to the image. Docker caches these layers, making subsequent builds faster by reusing unchanged layers.

- **Commands**: Dockerfile instructions tell Docker what to do when building the image. Some common commands include:

Learning outcomes (What I have learnt):

1. Learnt the concept of containerization
2. Learnt how to build docker images using Dockerfile.
3. Learnt to configure Docker to work with different environments
4. Learnt the purpose of Docker volumes and their role in data persistence.
5. Learnt how to use Docker Hub to pull and push Docker images.

Evaluation Grid (To be created as per the SOP and Assessment guidelines by the faculty):

Sr. No.	Parameters	Marks Obtained	Maximum Marks
1.			
2.			
3.			