10 – DAY – TASK (07-08-2024)

1. **Mention the actions of following comments:**
   git remote add origin "http://github/a.git"

   Git pull origin master

   Git push origin dev

   git remote add origin "http://github/a.git"
   Action: This command adds a new remote repository to your local Git configuration. The remote is named origin, and the URL "http://github/a.git" Points to the repository on a remote server (in this case, GitHub). After running this command, you can use the name origin to refer to this remote repository in other Git commands.
   git pull origin master
   Action: This command fetches changes from the master branch of the origin remote repository and merges those changes into your current branch. Essentially, it's a way to update your local branch with changes from the remote repository's master branch.
   git push origin dev
   Action: This command pushes your local dev branch to the origin remote repository. It uploads your commits from your local dev branch to the dev branch on the remote repository. If the dev branch doesn't exist on the remote, it will be created.

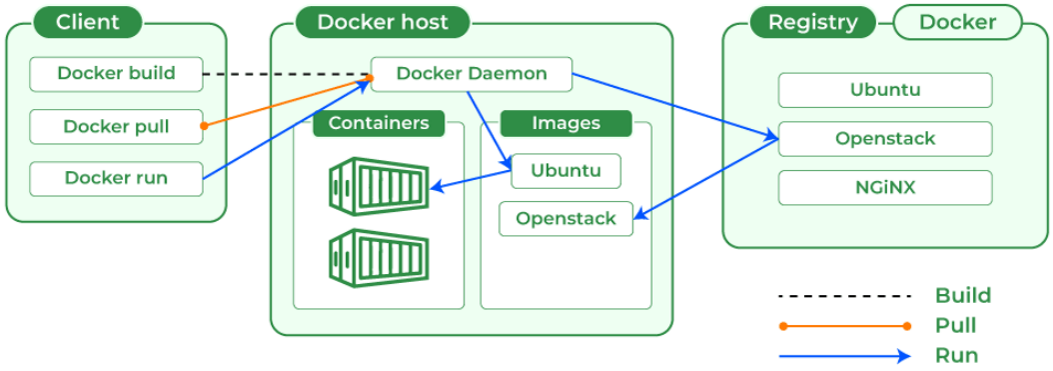
2. **What are the functions of following Docker objects and key components:**
   Dockerd:
   Dockerfile
   Docker-compose.yaml
   Docker Registries
   DockerHost

   Key Components of Docker

   The following are the some of the key components of Docker:

   - **Docker Engine:** It is a core part of docker, that handles the creation and management of containers.

- **Docker Image:** It is a read-only template that is used for creating containers, containing the application code and dependencies.
- **Docker Hub:** It is a cloud based repository that is used for finding and sharing the container images.
- **Dockerfile:** It is a script that containing instructions to build a docker image.
- **Docker Registry**: It is a storage distribution system for docker images, where you can store the images in both public and private modes.



dockerd

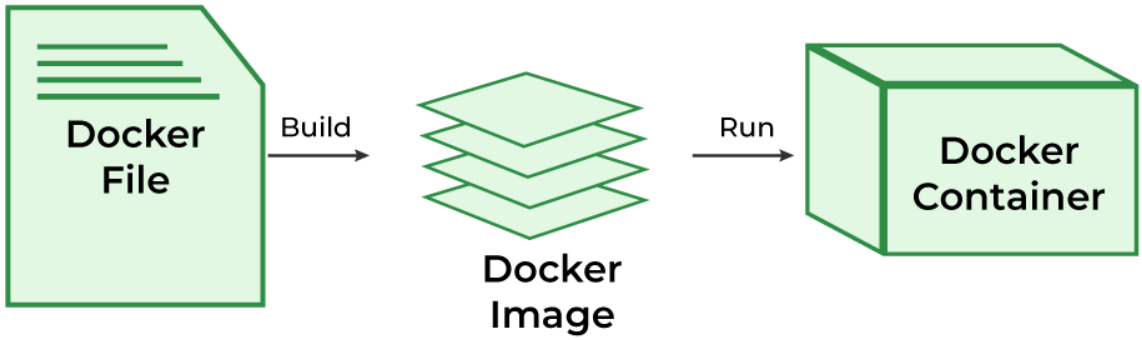
Function: dockerd is the Docker daemon, the background service that manages Docker containers on a host machine. It listens for Docker API requests and handles the creation, management, and orchestration of containers. The daemon also manages container images, networks, and volumes.

Configuration: Can be configured through various options and flags to customize its behavior (e.g., --storage-driver, --insecure-registry).

Dockerfile

Function: A Dockerfile is a script containing a series of instructions for building a Docker image. It defines the base image to use, the commands to run inside the container, files to copy, environment variables to set, and other configuration details. By using docker build with a Dockerfile, you create a custom Docker image

tailored to your application's requirements.



Instructions:

- FROM: Specifies the base image.
- RUN: Executes commands inside the image (e.g., installing packages).
- COPY / ADD: Copies files from the host to the image.
- CMD / ENTRYPOINT: Specifies the command to run when the container starts.

docker-compose.yaml

Function: The docker-compose.yaml file is used by Docker Compose to define and run multi-container Docker applications. It specifies the services (containers), networks, and volumes needed for the application. This file allows you to configure and manage multiple containers with a single command (docker-compose up), simplifying the setup of complex applications.

Key Sections:

- services: Defines each container in the application.
- networks: Configures network settings and links between services.
- volumes: Defines data storage options and mounts.

Docker Registries

Function: Docker Registries are repositories where Docker images are stored and managed. Docker Hub is the default public registry, but you can also use private registries. Registries allow you to push (upload) and pull (download) images to and from the registry, facilitating the sharing and distribution of Docker images.

Examples:

• Docker Hub: The default public registry, hosting a large number of open-source images.

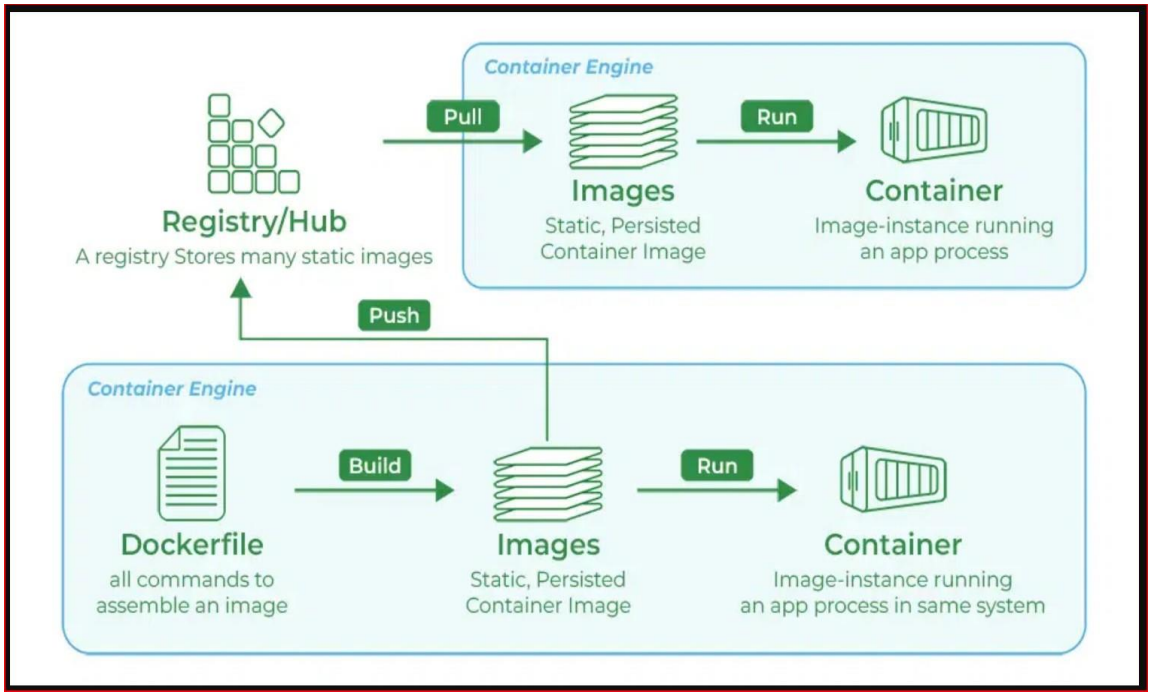
• Private Registries: Custom or enterprise registries for internal use.

Docker Host

Function: A Docker Host is the physical or virtual machine where the Docker daemon (`dockerd`) runs and where Docker containers are executed. The Docker Host provides the underlying infrastructure, including CPU, memory, storage, and network resources, necessary for running and managing Docker containers. It can be a local machine, a VM, or a cloud-based instance.

Types:

- Local Docker Host: Typically, a developer's workstation or a local server.
- Cloud-Based Docker Host: VMs or instances in cloud environments like AWS, Azure, or Google Cloud.



3. **What's the isolation in Docker container**

   Isolation in Docker container:

   Process Isolation:

   Namespace: Docker uses Linux namespaces to provide process isolation. Namespaces ensure that processes in one container cannot see or interact with processes in another container or on the host system. There are several types of namespaces, including:

   - **PID Namespace:** Isolates process IDs, so each container has its own process tree.
   - **NET Namespace:** Provides isolated network interfaces and routing tables.

- **IPC Namespace:** Isolates inter-process communication resources like shared memory.
- **UTS Namespace:** Allows each container to have its own hostname and domain name.

File System Isolation:

Filesystem Layering: Docker uses a union file system (such as OverlayFS or aufs) to create a layered file system. Each container gets its own file system that is built from layers derived from the Docker image. Changes in one container's file system do not affect other containers.

Resource Isolation:

Control Groups (cgroups): Docker uses cgroups to limit and prioritize resource usage (CPU, memory, I/O) for each container. This ensures that containers do not consume more resources than allocated and that the host remains responsive.

Network Isolation:

Virtual Networks: Docker creates isolated virtual networks for containers. Containers can communicate with each other within the same network but are isolated from external networks unless explicitly configured otherwise. Docker also provides mechanisms for configuring network access controls and firewall rules.

User Namespace Isolation:

User Namespaces: Docker can map container user IDs to different host user IDs, enhancing security by isolating container users from host users. This helps in reducing the risk of privilege escalation attacks.

Security Contexts:

Security Modules: Docker can use Linux security modules (like SELinux, AppArmor, or seccomp) to apply security policies that further isolate containers and restrict their capabilities.