DATE : 07/08/2024

Module 4 – Web Assignment

1.Mention the actions of following comments:

git remote add origin "http://github/a.git"
Git pull origin master
Git push origin dev

Answer:

1. git remote add origin "http://github/a.git"
    - **Purpose**: This command sets up a connection between your local repository and a remote repository hosted at the specified URL. This is typically used to push and pull changes to and from a remote server.
    - **Usage**: Once set, you can use commands like git push and git pull to sync changes between your local and remote repositories.
2. git pull origin master
    - **Purpose**: This command fetches changes from the remote master branch and merges them into your current local branch. It combines git fetch and git merge.
    - **Usage**: It's useful for integrating updates from the remote repository into your local branch to keep it up-to-date.
3. git push origin dev
    - **Purpose**: This command pushes your local dev branch to the remote repository, updating the dev branch on the remote named origin.
    - **Usage**: It’s used to share your local changes with others working on the same branch or to deploy code to a remote environment.

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

   Answer:

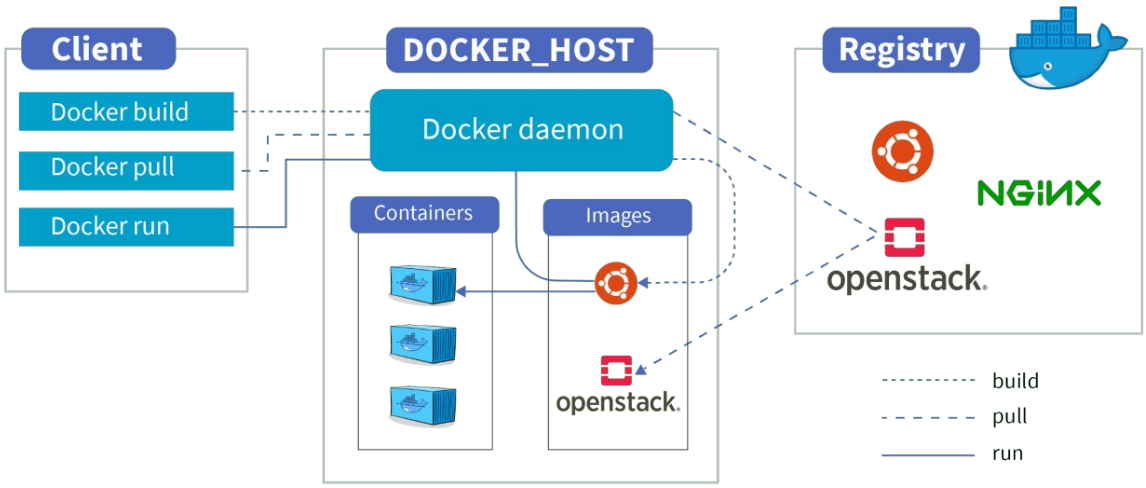
   Docker Components:

1. **dockerd (Docker Daemon)**
   - **Function**: Manages Docker containers, images, networks, and volumes. It runs as a background service and communicates with the Docker CLI and other Docker clients through a REST API or Unix socket.
   - **Configuration**: Can be configured through various options and flags to customize its behavior (e.g., --storage-driver, --insecure-registry).
2. Dockerfile
   - **Purpose**: Defines how to build a Docker image by specifying a sequence of instructions. It typically starts with a base image and then adds layers for installing applications, copying files, and configuring the environment.
   - **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.
3. docker-compose.yaml
   - **Purpose**: Defines and manages multi-container Docker applications. It allows you to configure all aspects of your application stack, including services, networks, and volumes in a single file.
   - **Key Sections**:
     - services: Defines each container in the application.
     - networks: Configures network settings and links between services.
     - volumes: Defines data storage options and mounts.
4. Docker Registries
   - **Function**: Store and distribute Docker images. Images are pushed to and pulled from registries, making them accessible for deployment and sharing.
   - **Examples**:

- **Docker Hub**: The default public registry, hosting a large number of open-source images.
- **Private Registries**: Custom or enterprise registries for internal use.

5. DockerHost

- **Role**: Provides the runtime environment for Docker containers. It includes the Docker daemon, which orchestrates container lifecycle operations.
- **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

Answer:

Docker Container Isolation:

1. **Namespaces**:

- **Types**:
  - **PID Namespace**: Isolates process IDs, ensuring processes in one container don't interfere with those in another.
  - **Network Namespace**: Provides a separate network stack, including IP addresses and routing tables.

    - **Mount Namespace**: Allows containers to have their own filesystem view, separate from the host.
    - **UTS Namespace**: Isolates hostname and domain name, making each container appear to have its own unique host.
2. **Control Groups (cgroups)**:
    - **Function**: Limits the resources available to containers (e.g., CPU, memory, disk I/O). This ensures that no single container can overwhelm the Docker host.
3. **Union Filesystems**:
    - **Function**: Allows Docker to create images with layered filesystems. Each layer can be read-only or writable, and changes are layered on top of the base image.
4. **Network Isolation**:
    - **Function**: Containers can be assigned unique IP addresses and configured with specific network rules. Docker's network driver can create isolated networks for different sets of containers, allowing controlled communication.