Module 4

1.Mention the actions of following comments:

- **git remote add origin "http://github/a.git":-**

  Answer:-
  This command adds a new remote repository to your local Git repository and names its origin. The URL of the remote repository is `"http://github/a.git"`. This allows you to interact with the remote repository using the name `origin`.

- **Git pull origin masterAnswer:-**

  Answer:-
  This command fetches the latest changes from the `master` branch of the remote repository named `origin` and merges those changes into the current branch of your local repository. If there are any updates on the `master` branch in the remote repository, they will be incorporated into your current branch.

- **Git push origin dev**

  Answer:-
  This command pushes the commits from your local `dev` branch to the remote repository named `origin`, updating the `dev` branch in the remote repository. If the `dev` branch does not exist on the remote, this command will create it and upload your commits.

2. What are the functions of following Docker objects and key components:
Dockerd:

Dockerfile

Docker-compose.yaml
Docker Registries
DockerHost?

Answer:-

Dockerd:

- **`dockerd`** is the Docker daemon, which is the background service running on the host that manages Docker containers. It listens for Docker API requests and manages Docker objects such as images, containers, networks, and volumes. The Docker daemon can communicate with other daemons to manage Docker services.

Dockerfile:

- A **`Dockerfile`** is a text file that contains a series of instructions on how to build a Docker image. Each instruction in a Dockerfile creates a layer in the image. Dockerfiles are used to automate the process of creating Docker images, ensuring consistency and repeatability.

docker-compose.yaml:

- This is a configuration file for **Docker Compose**, which is a tool for defining and running multi-container Docker applications. In the `docker-compose.yaml` file, you define the services, networks, and volumes that your application needs. Docker Compose then uses this file to create and start all the services defined in it with a single command.

Docker Registries:

- **Docker registries** are repositories where Docker images are stored and distributed. The most commonly known registry is Docker Hub, but you can also use private registries. Registries allow you to share images with others and are essential for deploying Docker images across different environments.

DockerHost:

- The Docker Host is the physical or virtual machine where the Docker daemon (`dockerd`) runs. It provides the resources (CPU, memory, storage) needed to run Docker containers. The Docker Host can be a local machine, a server in a data center, or a cloud-based virtual machine.

3.What's the isolation in Docker container?Isolation in Docker containers is achieved through several key technologies:

Answer:-

1. Namespaces:
    - Namespaces provide the first and most important isolation layer in the Linux kernel. They ensure that a container's processes, network interfaces, and filesystem mounts are isolated from those of other containers and the host system. Different types of namespaces include:
        - PID namespace: Isolates process IDs, ensuring that a container's processes cannot see or interact with processes outside of the container.
        - NET namespace: Provides a separate network stack for each container, allowing containers to have their own IP addresses, network interfaces, and routing tables.
        - IPC namespace: Isolates inter-process communication resources, like shared memory segments.
        - MNT namespace: Isolates filesystem mount points, ensuring a container cannot see or affect the filesystem of the host or other containers.
        - UTS namespace: Isolates hostname and domain name, allowing containers to have independent hostnames.
2. Control Groups (cgroups):
    - Cgroups limit, account for, and isolate the resource usage (CPU, memory, disk I/O, network, etc.) of a collection of processes. This ensures that containers can only use the resources that have been allocated to them, preventing a single container from exhausting the host's resources.
3. Union File Systems (UnionFS):
    - UnionFS allows files and directories of separate file systems (known as layers) to be transparently overlaid, forming a single coherent file system. This is used in Docker to create images where each layer represents a different filesystem state. Containers use these layers to ensure isolation and efficiency.
4. Seccomp, Capabilities, and AppArmor/SELinux:
    - Seccomp (Secure Computing Mode) restricts the system calls a container can make, reducing the kernel's attack surface.
    - Capabilities fine-tune the permissions of root within containers, granting only specific privileges required by the application.
    - AppArmor and SELinux provide additional mandatory access controls, further isolating containers from each other and the host.