# Docker Concepts and Commands

Docker is an open-source platform for automating the deployment, scaling, and management of applications inside lightweight containers. It allows developers to package applications with all their dependencies to ensure consistent operation across different environments.

## Key Docker Concepts

### 1. **Docker Containers**

- A container is a lightweight, standalone, executable package that includes everything needed to run a piece of software, including the code, runtime, system tools, libraries, and settings.
- Containers are isolated but can share the same OS kernel, which makes them more efficient than traditional virtual machines.

### 2. **Docker Images**

- Docker images are read-only templates used to create containers. An image contains the application code and its dependencies. Images are built using a Dockerfile.
- They are stored in Docker registries like Docker Hub, which can be pulled to create containers.

### 3. **Dockerfile**

- A Dockerfile is a text file that contains all the commands to assemble an image.
- Each instruction in a Dockerfile adds a layer to the image, making it possible to track changes and revert to previous states.

### 4. **Docker Registry**

- A Docker registry is a repository for Docker images. Docker Hub is the default registry where images are stored and shared.
- You can push your images to a registry, or pull images from it.

## 5. **Docker Compose**

- Docker Compose is a tool for defining and running multi-container Docker applications.
- A `docker-compose.yml` file allows you to configure all your services, networks, and volumes in one place.

## 6. **Docker Volumes**

- Volumes are used for persistent data storage in containers. Data in volumes persist even after the container stops or is removed.

## 7. **Docker Network**

- Docker networking allows containers to communicate with each other, the host system, or external networks.
- Common network drivers:
    - **bridge**: Default for standalone containers.
    - **host**: Shares the host's network stack.
    - **none**: No networking for the container.
    - **overlay**: Enables communication between different Docker hosts.

## 8. **Docker Swarm**

- Docker Swarm is Docker's native orchestration tool, allowing you to manage clusters of Docker Engines as a single entity.
- It enables scaling of services across multiple nodes.

---

# Essential Docker Commands

## 1. Docker Setup Commands

- **Install Docker (Linux/Ubuntu):**

```
sudo apt update
sudo apt install docker.io
sudo systemctl start docker
sudo systemctl enable docker
```

- **Check Docker Version:**

```
docker --version
```

- **Check Docker Info:**

```
docker info
```

## 2. Image Commands

- **Pull an Image from Docker Hub:**

```
docker pull <image-name>
# Example: docker pull nginx
```

- **List Docker Images:**

```
docker images
```

- **Build an Image from a Dockerfile:**

```
docker build -t <image-name> .
# Example: docker build -t my-app .
```

- **Remove an Image:**

```
docker rmi <image-id>
```

## 3. Container Commands

- **Run a Container:**

```
docker run -d -p <host-port>:<container-port> <image-name>
# Example: docker run -d -p 8080:80 nginx
```

- **List Running Containers:**

```
docker ps
```

- **List All Containers (Running & Stopped):**

```
docker ps -a
```

- **Stop a Running Container:**

```
docker stop <container-id>
```

- **Start a Stopped Container:**

```
docker start <container-id>
```

- **Remove a Container:**

```
docker rm <container-id>
```

- **Access a Running Container (Interactive Shell):**

```
docker exec -it <container-id> /bin/bash
```

## 4. **Dockerfile Commands**

- **Common Dockerfile Instructions:**
    - **FROM**: Specifies the base image to build from.
    - **RUN**: Runs a command inside the container during image build.
    - **COPY/ADD**: Copies files/directories from host to container.
    - **CMD**: Specifies the default command to run when a container starts.
    - **EXPOSE**: Exposes a container port.
    - **WORKDIR**: Sets the working directory for commands.

  Example Dockerfile:

```
FROM openjdk:11-jre-slim
WORKDIR /app
COPY target/myapp.jar /app
CMD ["java", "-jar", "myapp.jar"]
EXPOSE 8080
```

## 5. **Docker Compose Commands**

- **Start Services Defined in docker-compose.yml:**

```
docker-compose up
```

- **Start Services in Detached Mode:**

```
docker-compose up -d
```

- **Stop Running Services:**

```
docker-compose down
```

- **View Services' Status:**

```
docker-compose ps
```

## 6. Volume Commands

- **Create a Volume:**

```
docker volume create <volume-name>
```

- **List Volumes:**

```
docker volume ls
```

- **Inspect a Volume:**

```
docker volume inspect <volume-name>
```

- **Remove a Volume:**

```
docker volume rm <volume-name>
```

## 7. Networking Commands

- **Create a Network:**

```
docker network create <network-name>
```

- **List Networks:**

```
docker network ls
```

- **Inspect a Network:**

```
docker network inspect <network-name>
```

- **Connect a Container to a Network:**

  ```
  docker network connect <network-name> <container-id>
  ```

- **Disconnect a Container from a Network:**

  ```
  docker network disconnect <network-name> <container-id>
  ```

## 8. Logging and Debugging

- **View Container Logs:**

  ```
  docker logs <container-id>
  ```

- **Follow Real-Time Logs:**

  ```
  docker logs -f <container-id>
  ```

## 9. Docker Swarm Commands

- **Initialize a Docker Swarm:**

  ```
  docker swarm init
  ```

- **Join a Swarm as a Worker Node:**

  ```
  docker swarm join --token <worker-token> <manager-ip>:
  <manager-port>
  ```

- **Deploy a Stack of Services:**

  ```
  docker stack deploy -c <compose-file> <stack-name>
  ```

# Docker Tips & Best Practices

1. **Keep Images Small:** Use minimal base images to reduce image size.
2. **Use Multi-Stage Builds:** Helps create small, optimized images by building only what is necessary.
3. **Tag Your Images:** Always tag images with specific versions (e.g., `myapp:1.0.0`) instead of using `latest`.
4. **Leverage Volumes:** Use volumes for persistent data to avoid losing data when containers stop.
5. **Secure Your Containers:** Limit container privileges using the `--cap-drop` and `--security-opt` options.
6. **Clean Up Resources:** Use `docker system prune` to remove unused images, containers, networks, and volumes.