Here’s a list of some of the basic Docker commands along with explanations:

**1. docker --version**

**Description:** Displays the version of Docker installed on your system.

docker --version

**2. docker info**

**Description:** Displays detailed information about your Docker installation, including the number of containers, images, and other system-level information.

docker info

**3. docker pull <image>**

**Description:** Pulls a Docker image from Docker Hub or another registry. Example:

docker pull ubuntu

**4. docker build -t <name>:<tag> <path>**

**Description:** Builds an image from a Dockerfile. The -t flag tags the image with a name and optional tag. Example:

docker build -t myimage:latest .

**5. docker run <options> <image>**

**Description:** Runs a container from the specified image. You can also specify various options like -d for detached mode, -p to map ports, etc. Example:

docker run -d -p 8080:80 nginx

* -d: Runs the container in detached mode (in the background).
* -p: Maps ports from the container to the host (e.g., 8080:80).

**6. docker ps**

**Description:** Lists running containers.

docker ps

**7. docker ps -a**

**Description:** Lists all containers, including stopped ones.

docker ps -a

**8. docker stop <container\_id>**

**Description:** Stops a running container. Example:

docker stop mycontainer

**9. docker start <container\_id>**

**Description:** Starts a stopped container. Example:

docker start mycontainer

**10. docker restart <container\_id>**

**Description:** Restarts a running or stopped container. Example:

docker restart mycontainer

**11. docker rm <container\_id>**

**Description:** Removes a stopped container from your system. Example:

docker rm mycontainer

**12. docker rmi <image\_id>**

**Description:** Removes a Docker image from your system. Example:

docker rmi myimage

**13. docker exec -it <container\_id> <command>**

**Description:** Executes a command inside a running container. The -it flags allow you to interact with the container (i.e., run an interactive terminal session). Example:

docker exec -it mycontainer bash

**14. docker logs <container\_id>**

**Description:** Displays logs from a running or stopped container. Example:

docker logs mycontainer

**15. docker images**

**Description:** Lists all the images on your system.

docker images

**16. docker network ls**

**Description:** Lists all networks in Docker.

docker network ls

**17. docker volume ls**

**Description:** Lists all volumes in Docker.

docker volume ls

**18. docker-compose up**

**Description:** Starts services defined in a docker-compose.yml file.

docker-compose up

**19. docker-compose down**

**Description:** Stops and removes containers, networks, and volumes created by docker-compose up.

docker-compose down

**20. docker push <image>**

**Description:** Pushes a local image to a Docker registry (e.g., Docker Hub). Example:

docker push myimage:latest

These are some of the most commonly used Docker commands, and they should help you get started. Would you like me to dive deeper into any specific command or concept?

Great! Let's dive deeper into some key Docker concepts and commands to help you understand them more thoroughly.

### ****1. Docker Images and Containers****

* **Images** are the blueprints for creating containers. They are read-only, and once a container is created from an image, it becomes a running instance of that image.
* **Containers** are the running instances of Docker images. They can be started, stopped, moved, and deleted.

#### **Building Docker Images**

When you build an image with docker build, you're specifying a series of steps in a file called a **Dockerfile**. This file outlines the instructions for building the image.

Here’s a basic example of a **Dockerfile**:

# Step 1: Use an existing image as a base (e.g., Ubuntu)

FROM ubuntu:20.04

# Step 2: Set environment variables

ENV DEBIAN\_FRONTEND=noninteractive

# Step 3: Install dependencies

RUN apt-get update && apt-get install -y curl

# Step 4: Copy local files into the container

COPY ./app /usr/src/app

# Step 5: Set the working directory

WORKDIR /usr/src/app

# Step 6: Specify the command to run the app

CMD ["python", "app.py"]

* **FROM:** Defines the base image.
* **RUN:** Executes commands in the container (e.g., install software).
* **COPY:** Copies files/folders into the container.
* **WORKDIR:** Sets the working directory inside the container.
* **CMD:** Specifies the command that will run when the container starts.

To build the image from the Dockerfile:

docker build -t myapp:latest .

The -t option tags the image with a name and version (myapp:latest), and the . refers to the current directory where the Dockerfile is located.

#### **Running Containers**

When you run docker run <image>, Docker creates a new container from the image and starts it. Here's an example:

docker run -d -p 8080:80 nginx

* -d: Runs the container in detached mode (background).
* -p 8080:80: Maps port 80 inside the container to port 8080 on your host machine. So you can access the web server at http://localhost:8080.

To verify that the container is running, use:

docker ps

This will show you the container’s ID, status, and port mappings.

#### **Inspecting Containers**

You can inspect a container for more details:

docker inspect <container\_id>

This will return detailed information in JSON format about the container’s configuration, environment variables, and more.

### ****2. Managing Containers: Start, Stop, Restart, Remove****

#### **Stopping and Starting Containers**

If you want to stop a container that’s running:

docker stop <container\_id>

To restart a container:

docker restart <container\_id>

If you want to start a stopped container:

docker start <container\_id>

#### **Removing Containers**

Once a container is stopped, you can remove it:

docker rm <container\_id>

If you want to remove a container immediately after stopping it:

docker rm -f <container\_id>

The -f forces the removal.

### ****3. Docker Volumes****

Volumes are a way to persist data generated by and used by Docker containers. By default, data inside a container is lost when the container is removed. Volumes help with this issue by storing data outside the container’s file system.

To create a volume:

docker volume create myvolume

You can then mount the volume to a container:

docker run -d -v myvolume:/data myimage

* -v myvolume:/data: Mounts the myvolume volume to the /data directory inside the container.

To see the list of volumes:

docker volume ls

To remove a volume:

docker volume rm myvolume

### ****4. Docker Networks****

By default, Docker containers communicate over the bridge network, but you can create custom networks for isolated or more controlled communication between containers.

To create a custom network:

docker network create mynetwork

To run a container on this network:

docker run -d --network mynetwork myimage

To list networks:

docker network ls

You can inspect a specific network:

docker network inspect mynetwork

### ****5. Docker Compose****

Docker Compose is a tool for defining and running multi-container Docker applications. With Compose, you define a docker-compose.yml file to specify the services, networks, and volumes your application needs.

Here’s a simple example of a docker-compose.yml:

version: '3'

services:

web:

image: nginx

ports:

- "8080:80"

app:

image: myapp:latest

environment:

- DATABASE\_URL=postgres://user:password@db:5432

db:

image: postgres

environment:

- POSTGRES\_PASSWORD=mysecretpassword

This file defines three services:

* web: A service that runs an Nginx web server.
* app: Your application container that depends on the database.
* db: A PostgreSQL database container.

To start the application:

docker-compose up

To stop it:

docker-compose down

You can also run it in detached mode:

docker-compose up -d

Docker Compose makes it easy to manage multi-container applications without worrying about the details of how containers interact with each other.

### ****6. Docker Logs****

To see the logs for a container, use:

docker logs <container\_id>

If you want to stream the logs in real-time:

docker logs -f <container\_id>

You can also specify log options to limit the output:

docker logs --tail 10 <container\_id>

This shows the last 10 lines of logs from the container.

### ****7. Docker Networking (More Detail)****

Docker provides several default network types:

* **bridge**: The default network driver; containers are connected to a private internal network.
* **host**: Containers share the host’s network stack.
* **none**: Containers have no networking.

You can create more custom networks if necessary for your application to allow fine-grained control over communication.

### ****8. Docker Push and Pull****

* **Pulling** an image from Docker Hub:

docker pull <image\_name>:<tag>

Example:

docker pull nginx:latest

* **Pushing** an image to Docker Hub (after logging in):

docker push <username>/<image\_name>:<tag>

### ****9. Docker Exec and Interactivity****

docker exec allows you to run commands inside a running container. It’s especially useful for debugging or interacting with a container’s environment.

Example:

docker exec -it <container\_id> bash

* -i: Keeps the container’s standard input open.
* -t: Allocates a pseudo-terminal, making the shell interactive.

### ****10. Docker Container Lifecycle****

1. **Create**: When you run docker run, Docker creates a container from an image.
2. **Start**: When you start the container, it begins running and executes the command specified in the Dockerfile or the command passed in docker run.
3. **Stop**: Containers can be stopped using docker stop.
4. **Restart**: Restarting a container can be done with docker restart.
5. **Remove**: Stopped containers can be removed using docker rm.