Docker commands→day1

If you're just getting started with Docker and want to know the basic commands to use from scratch, here's a simple guide to help you understand and work with Docker containers and images.

### Prerequisites:

1. \*\*Install Docker\*\*: Make sure Docker is installed on your machine. You can check this by running:

```bash

docker --version

```

2. \*\*Start Docker Daemon\*\*: Ensure that the Docker service is running. On most systems, Docker starts automatically, but you can manually start it with:

```bash

sudo systemctl start docker

```

### Docker Basic Commands

#### 1. \*\*Check Docker Version\*\*

To verify that Docker is installed and running correctly:

```bash

docker --version

```

#### 2. \*\*Pull an Image from Docker Hub\*\*

To download a pre-built image from Docker Hub (like an official Ubuntu image):

```bash

docker pull ubuntu

```

You can specify a version (tag) as well:

```bash

docker pull ubuntu:20.04

```

#### 3. \*\*List Docker Images\*\*

To list all the images currently available on your local machine:

```bash

docker images

```

This will show the `REPOSITORY`, `TAG`, `IMAGE ID`, `CREATED`, and `SIZE`.

#### 4. \*\*Run a Container\*\*

To start a new container based on an image, and open an interactive terminal session inside the container:

```bash

docker run -it ubuntu

```

This command:

- `run`: Starts a new container.

- `-it`: Combines two options:

- `-i`: Runs the container in interactive mode (keeps the STDIN open).

- `-t`: Allocates a pseudo-TTY (gives you a terminal).

If you want to run a command in a container, for example, a bash shell:

```bash

docker run -it ubuntu bash

```

#### 5. \*\*Run a Container in the Background (Detached Mode)\*\*

To run a container in the background (detached mode), use the `-d` flag:

```bash

docker run -d ubuntu

```

You can check the running container's logs using:

```bash

docker logs <container\_id>

```

#### 6. \*\*List Running Containers\*\*

To see the containers that are currently running:

```bash

docker ps

```

To see all containers (including stopped ones):

```bash

docker ps -a

```

#### 7. \*\*Stop a Running Container\*\*

To stop a running container:

```bash

docker stop <container\_id\_or\_name>

```

For example:

```bash

docker stop 123456789abc

```

#### 8. \*\*Remove a Container\*\*

To remove a container (make sure it's stopped first):

```bash

docker rm <container\_id\_or\_name>

```

#### 9. \*\*Remove an Image\*\*

To delete a Docker image from your local machine:

```bash

docker rmi <image\_name\_or\_id>

```

For example:

```bash

docker rmi ubuntu

```

#### 10. \*\*Build a Docker Image from a Dockerfile\*\*

A Dockerfile is a script containing a series of instructions to build a custom Docker image. Here’s how you can build an image:

1. \*\*Create a `Dockerfile`\*\* (in the same directory):

Example Dockerfile:

```dockerfile

# Use an official base image

FROM ubuntu:20.04

# Set environment variables

ENV DEBIAN\_FRONTEND=noninteractive

# Update and install software

RUN apt-get update && apt-get install -y python3 python3-pip

# Set the working directory in the container

WORKDIR /app

# Copy local files into the container

COPY . /app

# Run a command to install dependencies (if any)

RUN pip3 install -r requirements.txt

# Set the command to run on container startup

CMD ["python3", "app.py"]

```

2. \*\*Build the image\*\*:

```bash

docker build -t my-python-app .

```

This builds the image and tags it as `my-python-app`.

#### 11. \*\*Run a Container from Your Custom Image\*\*

After building an image, you can run a container from it:

```bash

docker run -it my-python-app

```

#### 12. \*\*Docker Exec (Access a Running Container)\*\*

To run a command inside a running container (like accessing its shell):

```bash

docker exec -it <container\_id\_or\_name> bash

```

This opens an interactive bash shell inside the running container.

#### 13. \*\*Docker Network\*\*

To view all Docker networks:

```bash

docker network ls

```

To create a custom network:

```bash

docker network create my\_network

```

#### 14. \*\*Docker Volumes (Persisting Data)\*\*

If you want to mount a volume (for persistent data) from the host to the container:

```bash

docker run -v /path/on/host:/path/in/container -it ubuntu

```

To list all volumes:

```bash

docker volume ls

```

#### 15. \*\*Docker Compose\*\* (for Multi-Container Applications)

If you need to manage multi-container applications (e.g., an app with both a backend and database), you can use \*\*Docker Compose\*\*. Here’s a quick example:

1. \*\*Create a `docker-compose.yml`\*\* file:

Example:

```yaml

version: '3'

services:

web:

image: nginx

ports:

- "80:80"

db:

image: postgres

environment:

POSTGRES\_PASSWORD: example

```

2. \*\*Start the services\*\*:

```bash

docker-compose up

```

To run it in detached mode:

```bash

docker-compose up -d

```

To stop the services:

```bash

docker-compose down

```

### Additional Helpful Commands:

- \*\*Show detailed info about a container\*\*:

```bash

docker inspect <container\_id\_or\_name>

```

- \*\*Search for Docker images\*\*:

```bash

docker search <image\_name>

```

### Summary:

With these commands, you'll be able to start from scratch, pull Docker images, run containers, build custom images, manage networks and volumes, and handle multi-container environments using Docker Compose. As you get more comfortable, you can dive deeper into Docker's capabilities.

Command for saving the image as

The docker save command is used to **export a Docker image** into a tarball archive file. This allows you to save an image locally on your filesystem, which can later be transferred, backed up, or loaded into another Docker instance (possibly on a different machine) using the docker load command.

→ docker save image\_name > image\_name.tar

for loading the image again

→ docker load < image\_name.tar