Images & Containers

Images

Images are **one of the two core building blocks** Docker is all about (the other one is "Containers").

Images are **blueprints / templates** for containers. They are **read-only** and contain the application as well as the necessary application environment (operating system, runtimes, tools, ...).

Images **do not run themselves**, instead, they can be executed as containers.

Images are **either pre-built** (e.g. official Images you find on DockerHub) or you build **your own** Images by defining a **Dockerfile**.

Dockerfiles contain **instructions** which are executed when an image is built (`docker build .`), every instruction then creates a **layer** in the image. Layers are used to **efficiently rebuild** and share images.

The `CMD` instruction is special: It's **not executed when the image is built** but when a **container is created and started** based on that image.

Containers

Containers are the **other key building block** Docker is all about.

Containers are **running instances of Images**. When you create a container (via `docker run`), a thin **read-write layer** is added on top of the Image.

Multiple Containers can therefore be started based on one and the same Image. All Containers run in **isolation**, i.e. they don't share any application state or written data.

You need to create and start a Container to start the application which is inside of a Container. So it's Containers which are in the end executed - both in **development and production**.

Key Docker Commands

For a full list of all commands, add `--help` after a command - e.g. `docker --help`, `docker run --help` etc.

Also view the official docs for a full, detailed documentation of ALL commands and features: https://docs.docker.com/engine/reference/run/

Important: This can be overwhelming! You'll only need a fraction of those features and commands in reality!

- `docker build .`: Build a Dockerfile and create your own Image based on the file

docker stop <container>: Stop the container with its ID or name.

docker image inspcct <image>: See the detailed information for the image.

docker attach <container>: Switch the container to the attached mode.

docker tag <old_image> <new_image>: Rename the image name and tag. When we rename an image, we actually create a clone of the old image, a clone with the new name, and the old image still exists.

    - `-t NAME:TAG`: Assign a `NAME` and a `TAG` to an image
- `docker run IMAGE_NAME`: Create and start a new container based on image `IMAGENAME` (or use the image id)
    - `--name NAME`: Assign a `NAME` to the container. The name can be used for stopping and removing etc.
    - `-d`: Run the container in **detached** mode - i.e. output printed by the container is not visible, the command prompt / terminal does NOT wait for the container to stop
    - `-it`: Run the container in "**interactive**" mode - the container / application is then prepared to receive input via the command prompt / terminal. You can stop the container with `CTRL + C` when using the `-it` flag
    - `--rm`: Automatically **remove** the container when it's stopped
- `docker ps`: **List** all **running** containers
    - `-a`: **List** all **containers** - including **stopped** ones
- `docker images`: **List** all **locally stored images**
- `docker rm CONTAINER`: **Remove** a container with name `CONTAINER` (you can also use the container id)
- `docker rmi IMAGE`: **Remove** an image by name / id
- `docker container prune`: **Remove** all **stopped** containers
- `docker image prune`: **Remove** all **dangling** images (untagged images)
    - `-a`: **Remove** all **locally stored** images
- `docker push IMAGE`: **Push** an image **to DockerHub** (or another registry) - the image name/ tag must include the repository name/ url Make sure to log in using "docker login" before pushing. Use "docker logout" to log out.
- `docker pull IMAGE`: **Pull** (download) an image **from DockerHub** (or another registry) - *this is done automatically if you just `docker run IMAGE` and the image wasn't pulled before*

docker run -p <local_port>:<container_port> <image>: The -p flag means publish a container's port(s) to the host and it allow us to specify the local port and the internal Docker container exposed port. Note that the "-p <local_port>:<container_port>" part cannot be seperated.

docker start <container>: Restart running the stopped container, but this container is running in the background (detached mode) by default, instead of running the container in the terminal (attached mode) like "docker run..." by default.
"-a" attach the container when restarting. "-i" attach container's STDIN, and we don't need "-t" when restarting because that was still memorized if we ran the container originally with "-t".

docker create <option> <image>: Create a container but not run it. Use "docker start" to run it instead of " docker run". <option> includes "--name", "--rm", "-p", "-it" ..., but doesn't include "-d" because this command doesn't run the container.

docker exec <option> <container> <command>: Execute a command in a running container (the container should be started first). For example: "docker exec <container> ls ." is to see the files in the working directory inside of the container.

docker cp <source_dir> <destionation_dir>: Copy the file or folder into a running container or out of a running container. If the directory is in the container, the container namd or ID should be specified like "<container>: <directory>". But the working directory cannot be written data from host by this command.

For all docker commands where an ID can be used, you don't always have to write out the full id. And we can just use the first (few) character(s) - just enough to have a unique identifier.
If the ID on the image is "abcdefg". So instead of: docker run abcdefg
you could also run: docker run abc
or, if there's no other image ID starting with "a", you could even run just: docker run a
This applies to ALL Docker commands where IDs are needed.