Docker concepts and Vocabulary

For understanding, motivation and discussion

4.9.2025

Docker

(PaaS) environment to build, deliver and run software as isolated and independent containers.

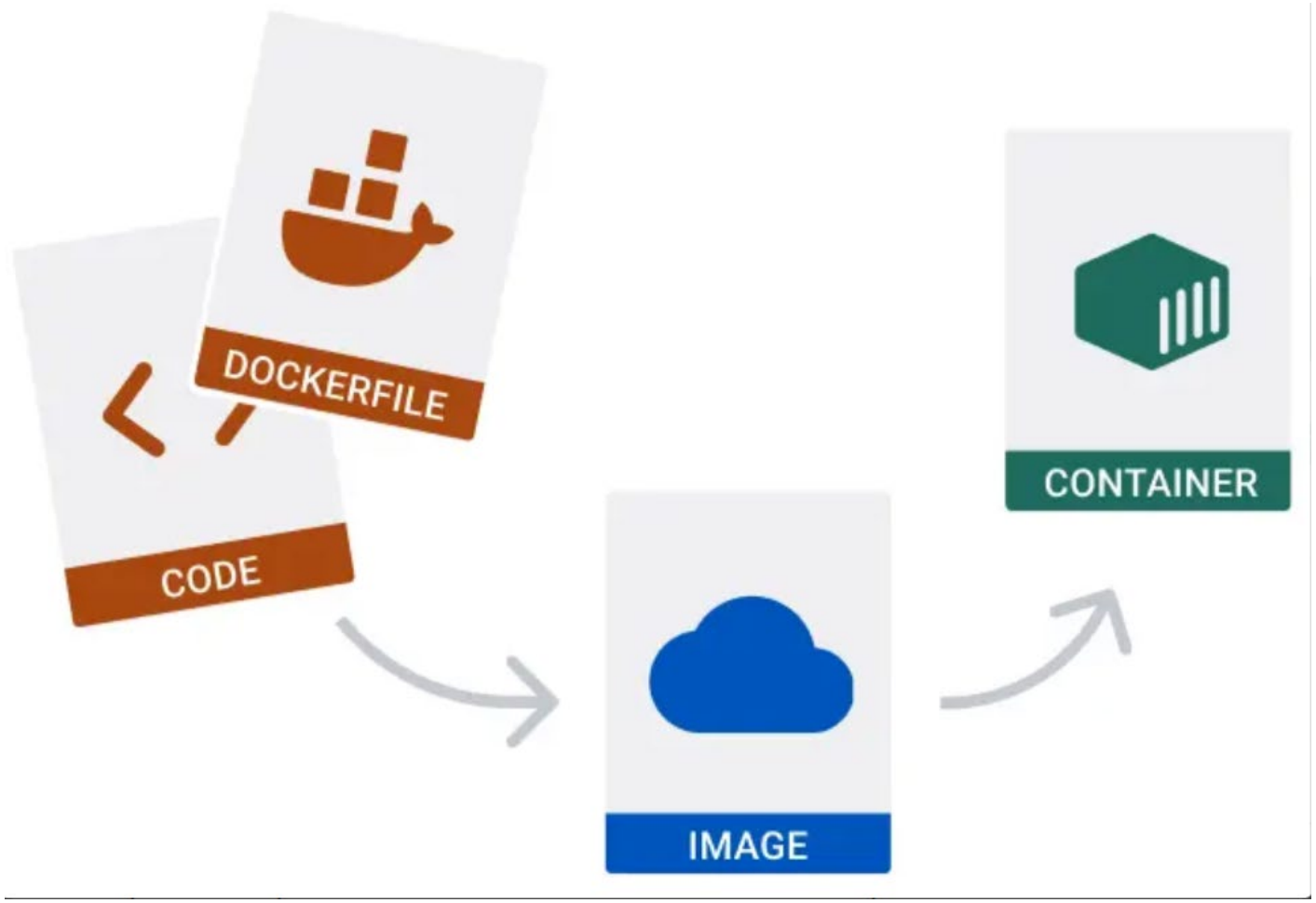
- By default, containers are sand-boxed / isolated.
- You can make container services available to others by **exposing ports**, e.g., for your laptop browser to use
- You can make containers to see others by connecting them by virtual **networks** between containers
- You can make containers to share data also by shared and **persisted volumes.** Basically, a shared folder that two or more containers can access.

Using Docker containers vs using Virtual Machines

- Shortest way is maybe to say that docker containers run a bit like just apps in a computer where there is only one real operating system running.
- In that operating system, there is a Docker Engine server (dockerd, Docker daemon), that runs the docker containers in isolation, managing what services to expose to each container and from containers to the host computer.

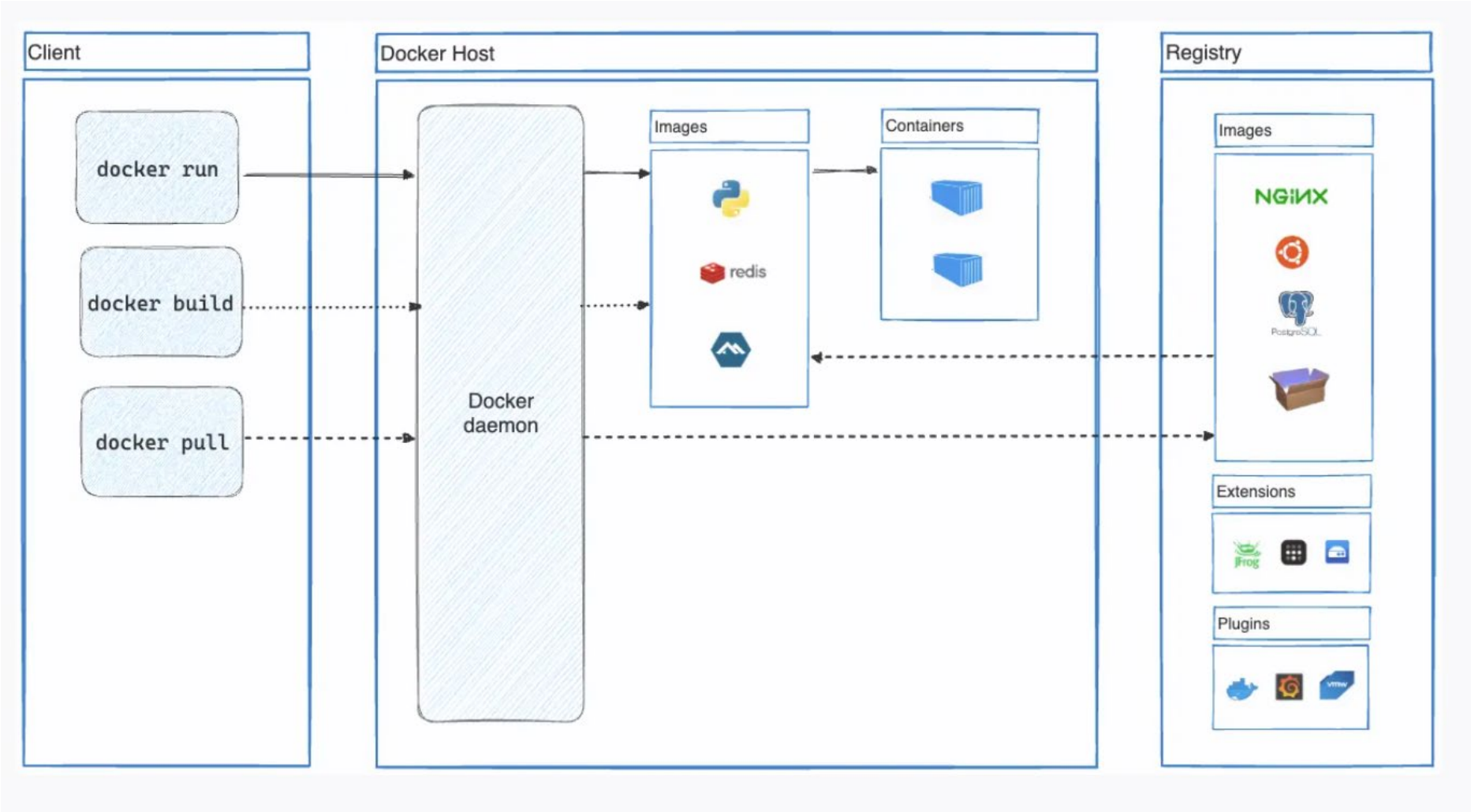
- On the other hand, inside the container, it feels that they are a real machines of their own, they feel like running e.g. light-weight Linux and its file system.

Docker - basic concepts



Source:
https://docs.docker.com

Docker – Bigger picture



Source:
https://docs.docker.com

Previous image showed how you can e.g.

- Run images as containers
- Build images based on your files, scripts and Dockerfiles (and a base image)
- Pull images from Docker registries (for running them as containers, or basing your own images on them)

Docker Engine

- The engine (*dockerd* daemon/process and Docker Engine API)
- To manage and run the containers.
- Isolate and connect those containers based on your definitions
  - From/to each other
  - From/to your host computer

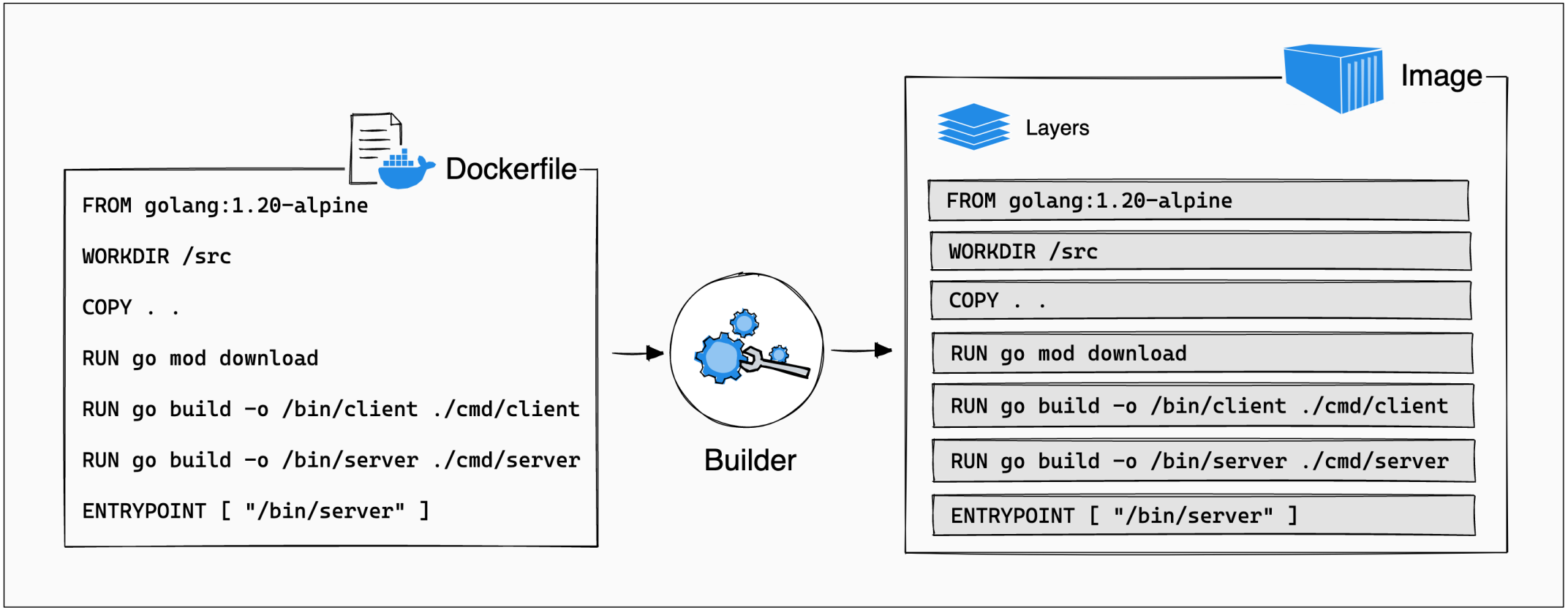
> docker (CLI command)

- The *docker* command line command.
- Client for giving commands e.g., about:
  - starting or stopping containers, removing containers
  - building, publishing images, removing images
  - finding the status of containers and images
  - Or executing commands inside the container from outside: docker exec …

Dockerfile

- Your ‘script’ for making your own Docker images.
- Image could be built based on source code and other assets on your disk
  - and image is usually based on / expanding on ready-made images from *Docker Hub* or other docker image registry.
    - FROM: ubuntu:latest

Dockerfile 2/2



Source:
https://docs.docker.com

.dockerignore

- Lists which files and folders won’t be copied/packed into the Docker image.

image

- Ready-made from Docker Hub, **OR** one you have created. Template that can be used to create containers.
- E.g. some database image you want to take into use. It's a snapshot of a running/runnable DB server that starts from a certain documented state, configured in a certain documented way. See the Docker hub for the documentation. Typically, there is a root user with known or set password, a certain database/schema created, like 'test'.
- If taking a ready-made image into use, you must then configure your own image/container, e.g. with DB image
  - secure the root user by changing the password
  - create a user with less privileges with safe password or other safe access.
  - give that user access to wanted schema etc.
  - continue possibly with table creation, etc…
  - a port?
  - persist the data (metadata and actual data rows) with a docker **volume**

Docker Image registry, e.g. Docker Hub

- We can push = publish our images for others (or us) to use.
- Or pull = download images to use ourselves.

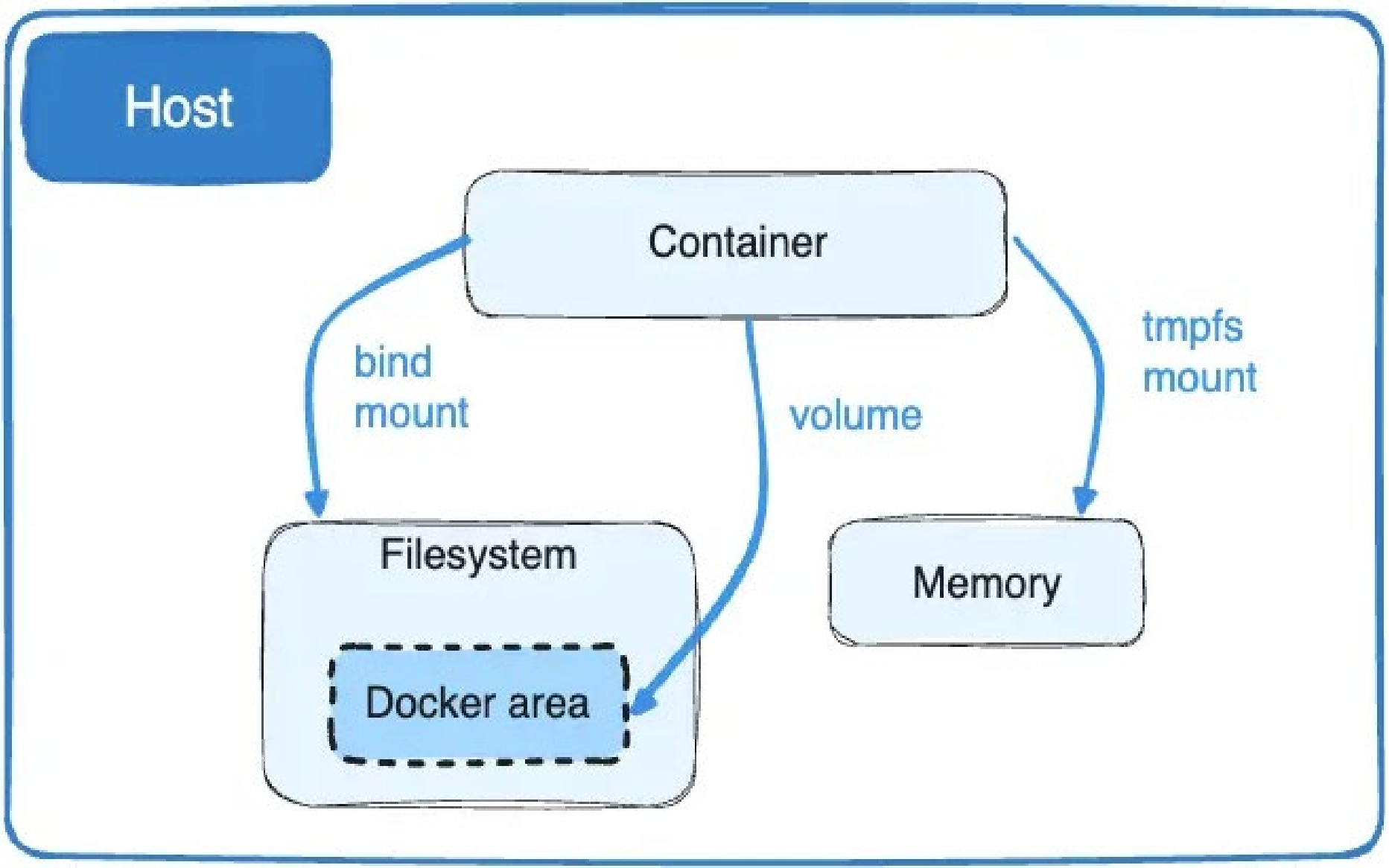
> docker-compose (CLI command)

- Tool for creating and starting multiple containers that depend on / talk to each other.
- Thus, you'll have to:
  - define one or more **services**, plus
    - make some **ports** published = opened
    - and/or define (virtual) **networks** shared by multiple containers
    - and/or share **volumes**.
- You can define those in a **docker-compose.yml** file
- https://docs.docker.com/compose/compose-application-model/

volume

- Persisted folders and files on disk.
- Allows sharing between containers, if they are configured to see that volume
- Also for keeping data between container deletion and re-creation!
  - (Container is deleted totally, but volume resides by default in the host file system)
- Three different main types of volumes exist, for different needs. Discussed in other material.

volume 2/2

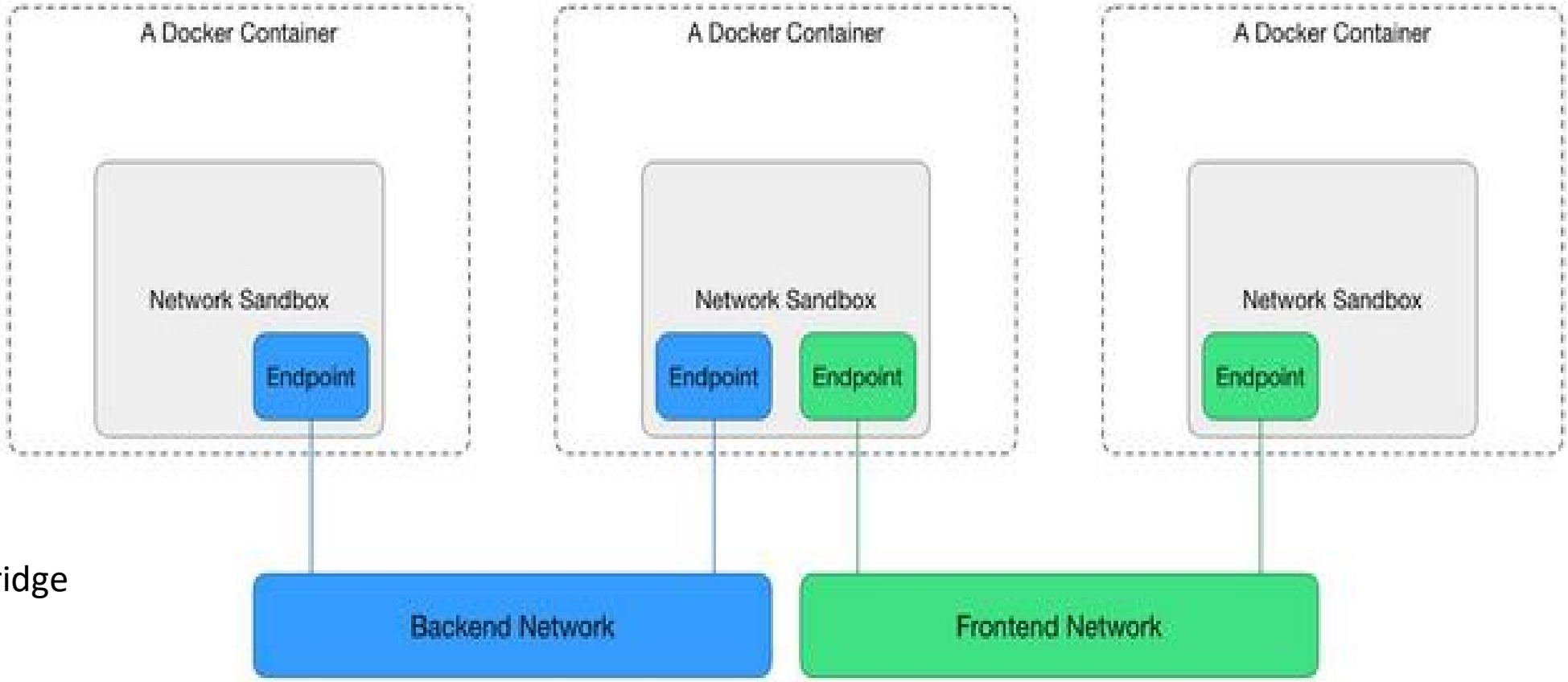


Source:
https://docs.docker.com

network

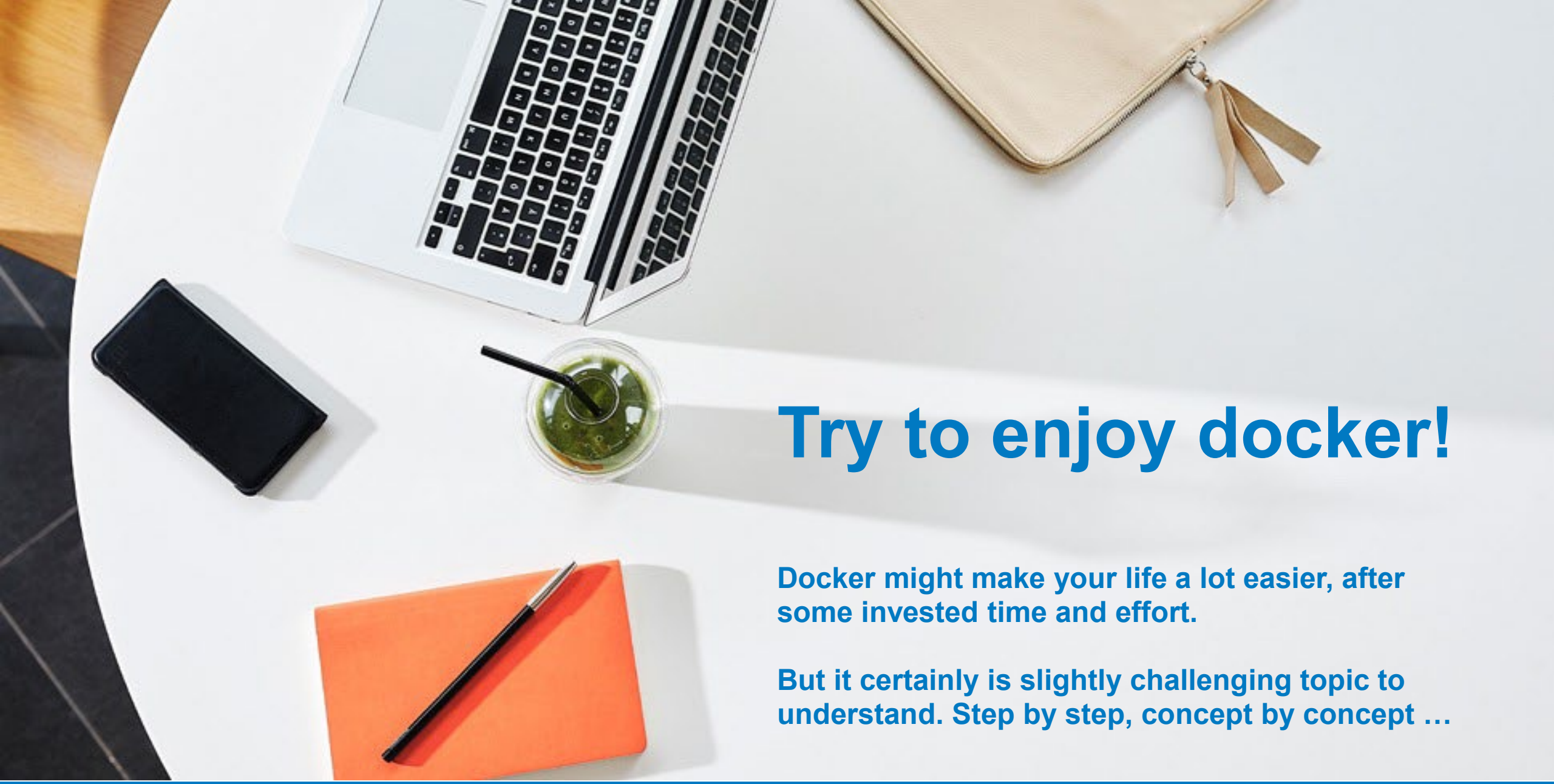
- Allows (by default isolated) containers to communicate with each other, or the host computer
- https://www.docker.com/blog/docker-compose-networking/
- There are multiple network configurations available, e.g. Bridge, Host, none, (Overlay, etc.)
- https://docs.docker.com/network/drivers/bridge/ **Bridge** network is the normal one, letting included containers communicate with each other.
  - **i) default** bridge, automatic when you don't specify any network. If you expose ports, they are visible in the host computer.
  - **ii) User-defined Bridge**, now you can use DNS name resolution, containers keep their DNS name. Usually the recommended network in docker, especially when e.g. having backend container communicating internally with database container!
- https://docs.docker.com/network/drivers/host/ **iii) Host** network refers to the host computer where Docker engine is run. Do you want to e.g. let the containers run like your host computer service? Sometimes, often not! No need to expose the ports, they are visible in the host like any service on host computer.
- **iv) None** = total isolation, the container is not connected to other containers or host computer with networks. Volume then?
- Etc. (There are others, but then we start to go to so complicated 'server farms' that better add Kubernetes too)

network 2/2



Here two user-defined bridge networks

Source:
https://docs.docker.com



Try to enjoy docker!

Docker might make your life a lot easier, after some invested time and effort.

But it certainly is slightly challenging topic to understand. Step by step, concept by concept …