Docker concepts and Vocabulary

For understanding, motivation and discussion

18.2.2024

Haaga-Helia

Docker

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

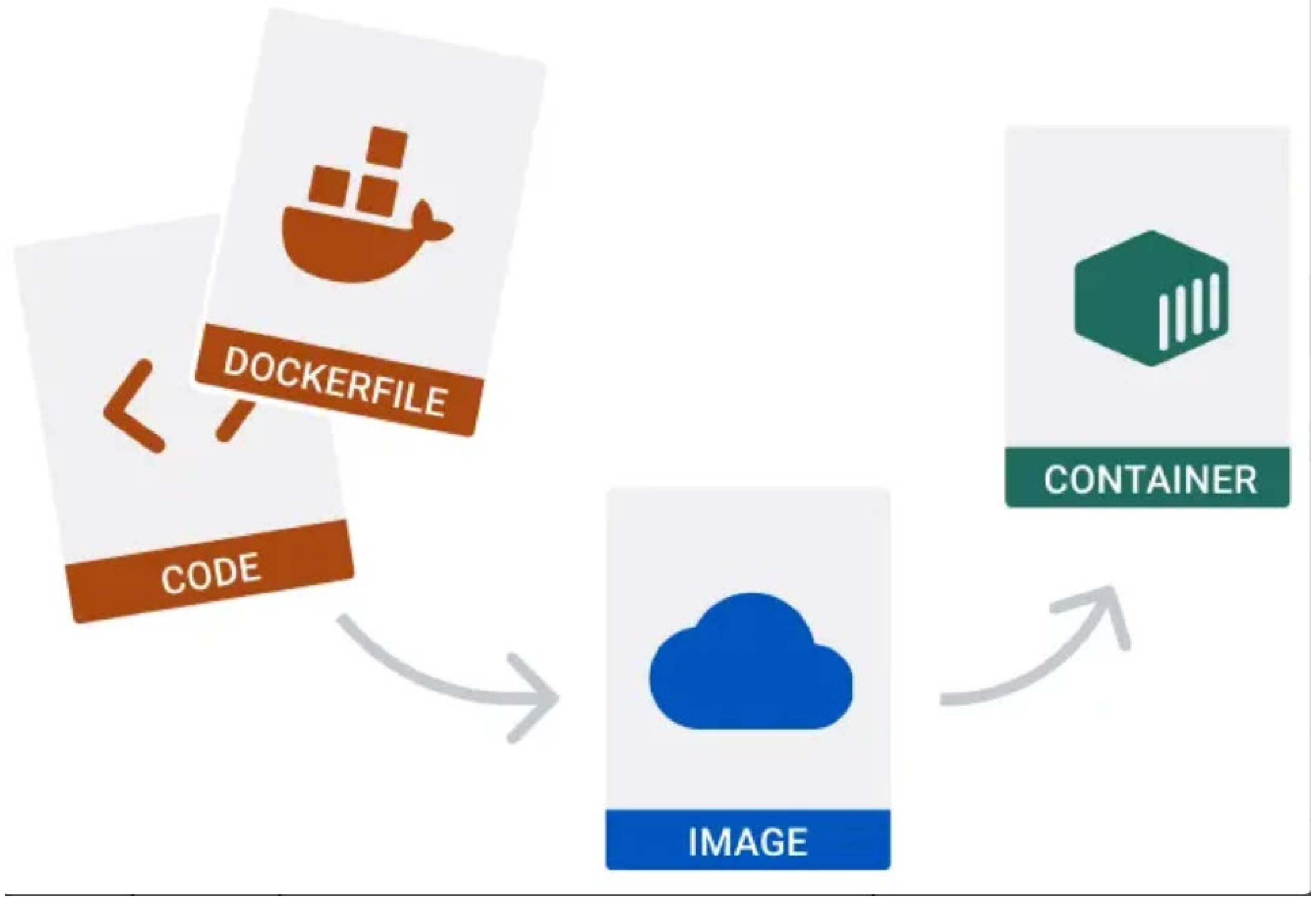
- By default, containers are sand-boxed / isolated.
- You can make container services available to others by **exposing ports**, e.g. for your browser to use
- You can make containers to see others by connecting them by virtual **networks** between containers
- You can make containers to share data also by shared **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 an docker engine server, that runs the docker containers in isolation, managing what services to give each container.

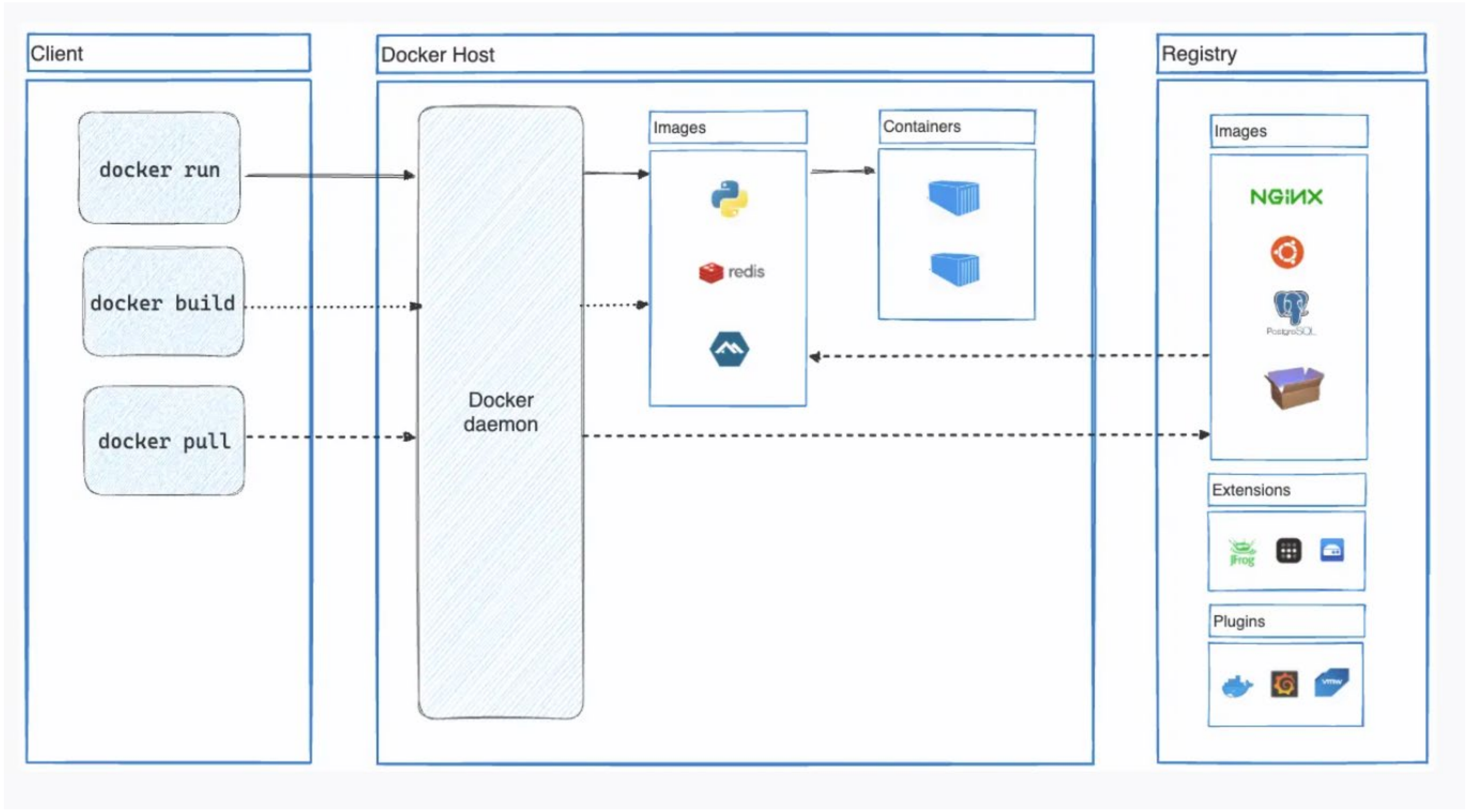
- On the otherhand, inside the container, it feels that they are a real machines of their own, they feel like running e.g. light-weight Linux and file system.

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
- Pull images (for running) from Docker registries

Docker Engine

- The engine (*dockerd* daemon/process and Docker Engine API)
- To manage and run the containers.
- Isolate and connect them based on definitions

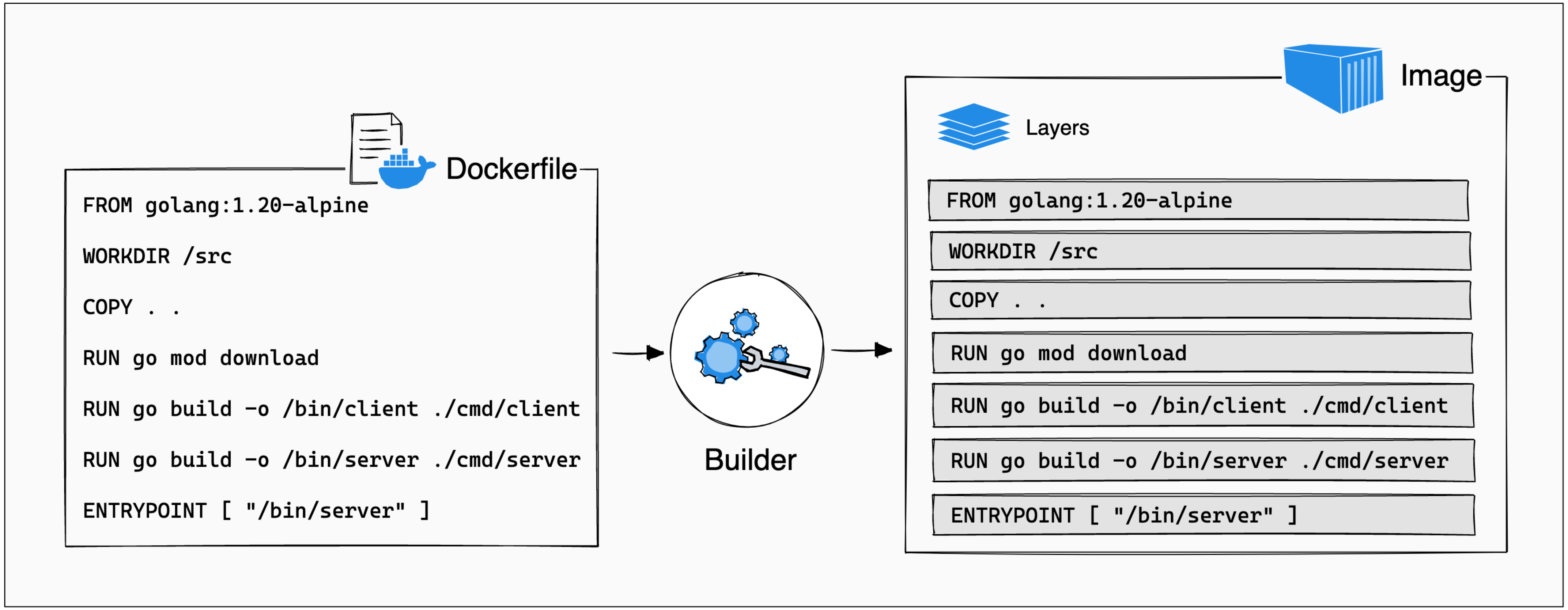
docker (CLI command)

- The *docker* command line command.
- Client for giving commands e.g. about
  - starting or stopping containers, removing containers
  - building, publishing images, removing images
  - finding 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, if needed, based on / expanding on ready-made images from *Docker Hub* or other docker image registry.

Dockerfile



```
FROM golang:1.20-alpine

WORKDIR /src

COPY . .

RUN go mod download

RUN go build -o /bin/client ./cmd/client

RUN go build -o /bin/server ./cmd/server

ENTRYPOINT [ "/bin/server" ]
```

```
FROM golang:1.20-alpine
WORKDIR /src
COPY . .
RUN go mod download
RUN go build -o /bin/client ./cmd/client
RUN go build -o /bin/server ./cmd/server
ENTRYPOINT [ "/bin/server" ]
```

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 container.
- E.g. some MariaDB image you want to take into use. It’s a snapshot of a running/runnable MariaDB or other DB server that starts from a certain documented state. Typically, there is a root user with known password (public information, everyone knows the password!), a certain database/schema created, like ‘test’. When taking that image into use, you must then immediately:
  - 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…

Docker Image registry, e.g. Docker Hub

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

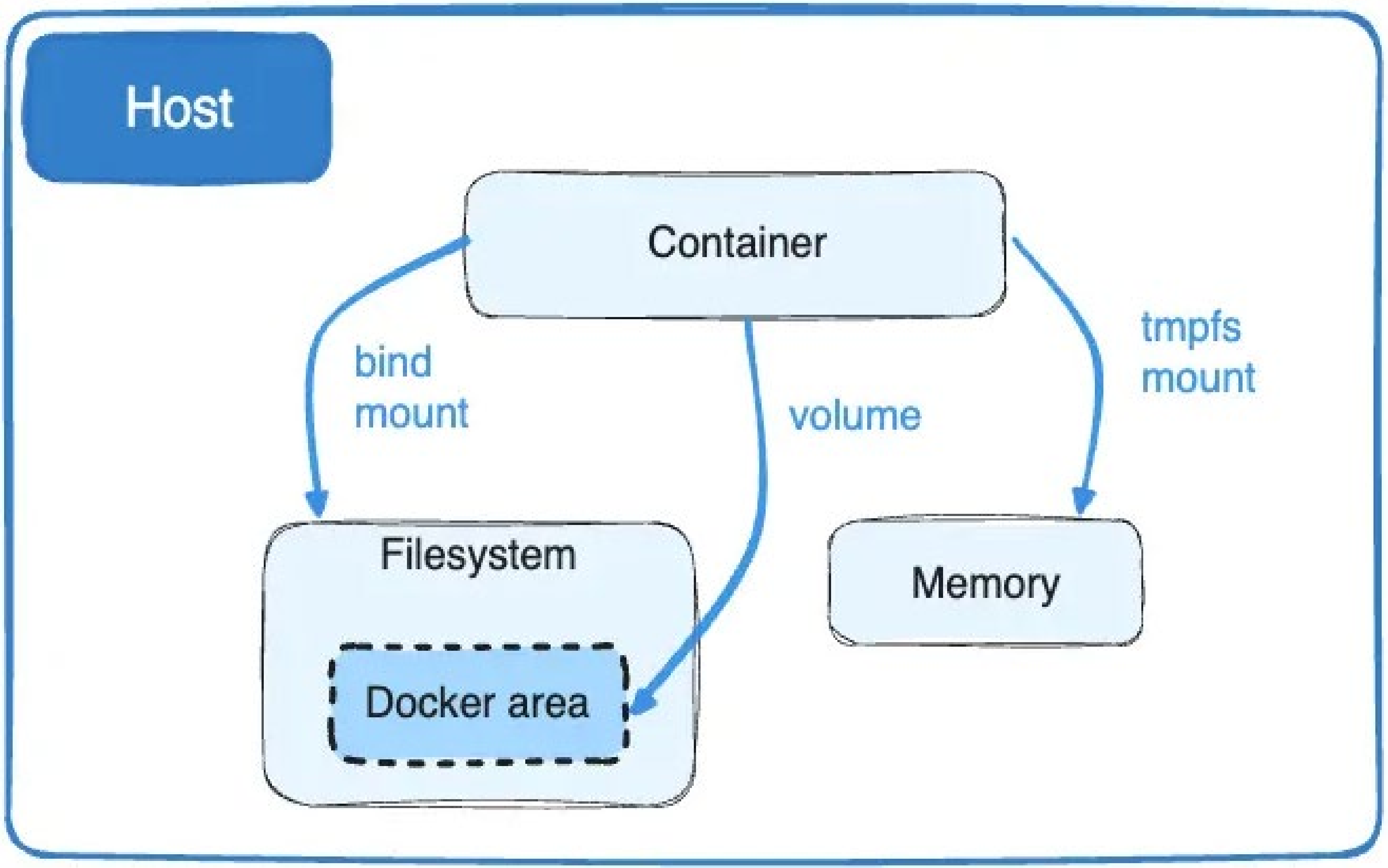
docker compose

- Tool for creating and starting multiple containers that depend on / talk to each other.
- Thus, you’ll have to:
    - define the services, plus
        - make some ports exposed
        - and/or define (virtual) networks shared by multiple containers
        - 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
- Also for keeping data between container deletion and re-creation!
    - (Container is deleted totally, but volume resides by default in the host file system)

volume

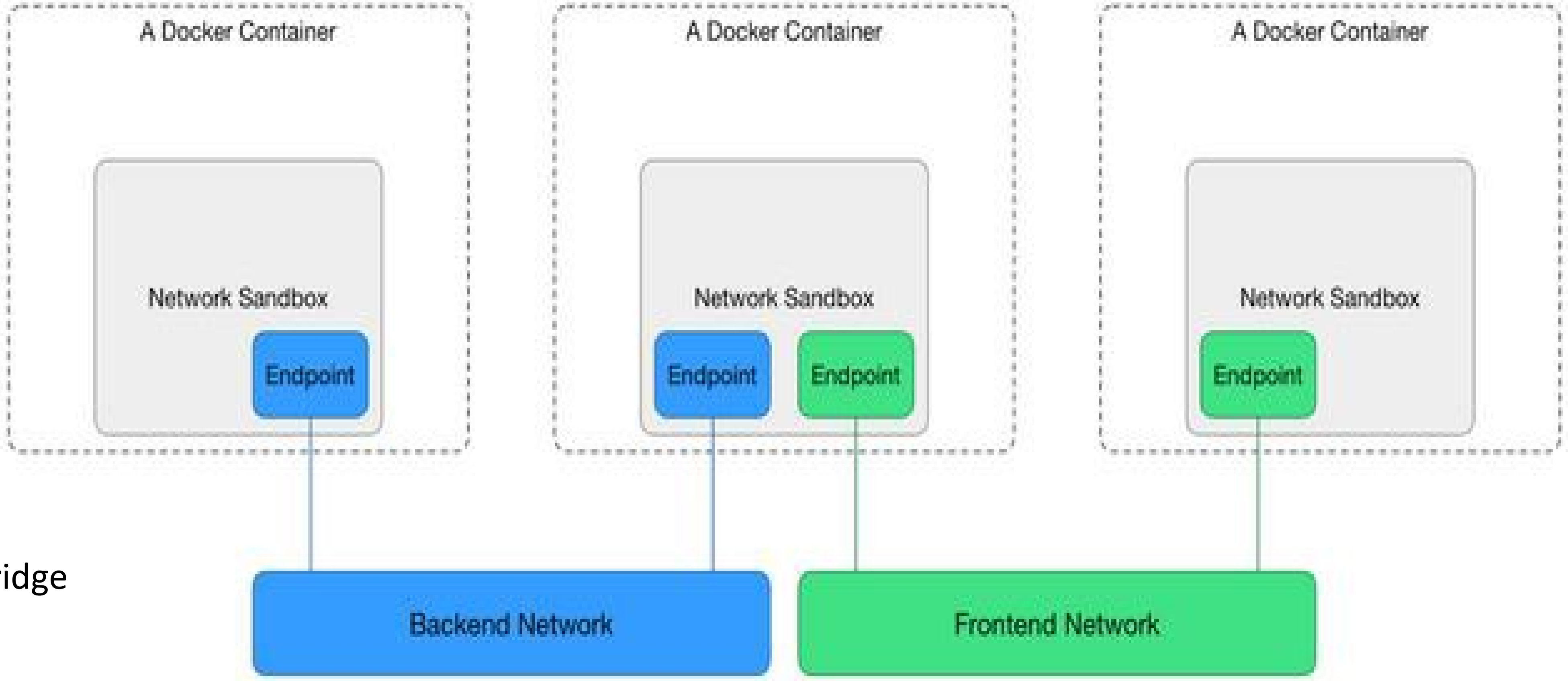


Source: https://docs.docker.com

network

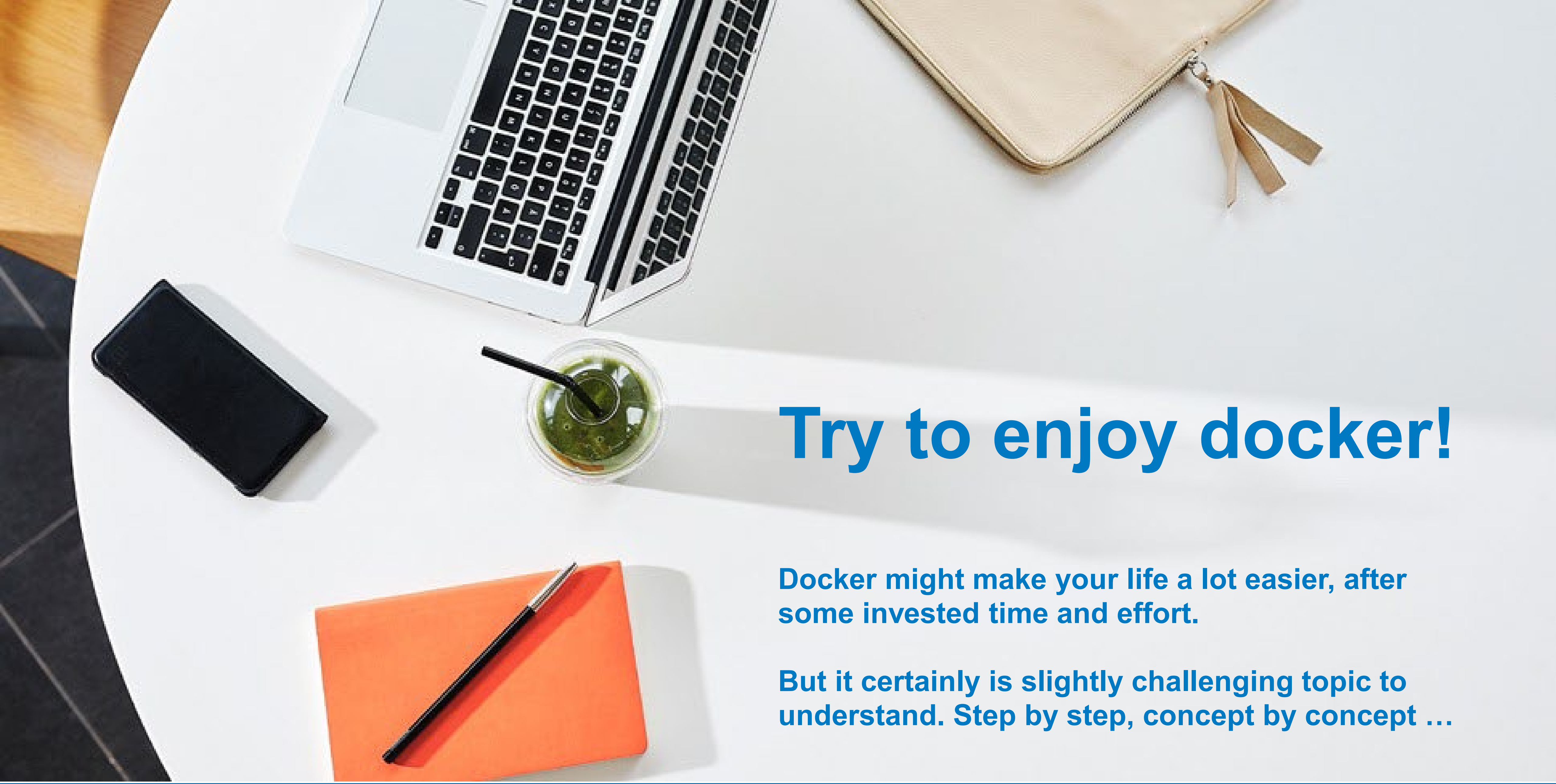
- Allows (by default isolated) containers to communicate with each other, or the host computer
- https://www.docker.com/blog/docker-compose-networking/
- There are multiple network configurations available, e.g. Bridge, Host, none, (Overlay, etc.)
- https://docs.docker.com/network/drivers/bridge/ **Bridge** 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
- Etc. (There are others, but then we start to go to so complicated 'server farms' that better add Kubernetes too)

network



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 …

Haaga-Helia