WSE1 – Werkzeuge im Software Engineering

SE.ba VZ

Docker Crash Course

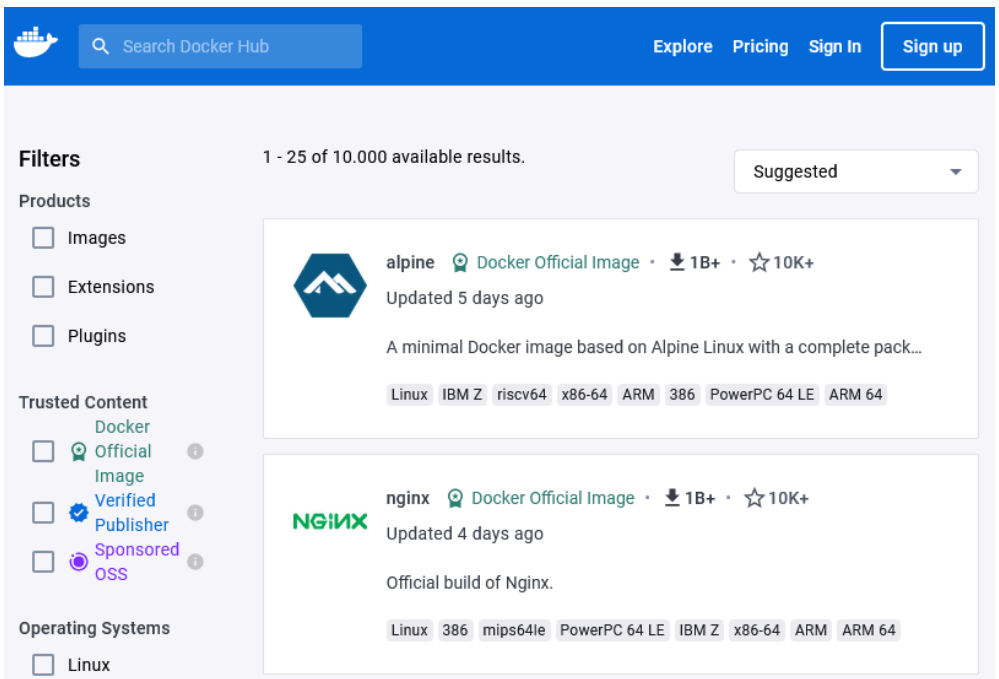
Johannes Karder, Andreas Scheibenpflug, Sebastian Pimminger

The Birth of Docker

https://imgur.com/3eTKEZp

Docker

- Set of tools to ...
  - create containers and images
  - configure them (Dockerfile)
  - execute them (Docker engine)
- Docker Hub
  - Platform for image distribution



https://hub.docker.com/search?q=

Container 1/2

Merriam-Webster

container noun

con·tain·er kən-ˈtā-nər

: one that contains: such as

a : a receptacle (such as a box or jar) for holding goods

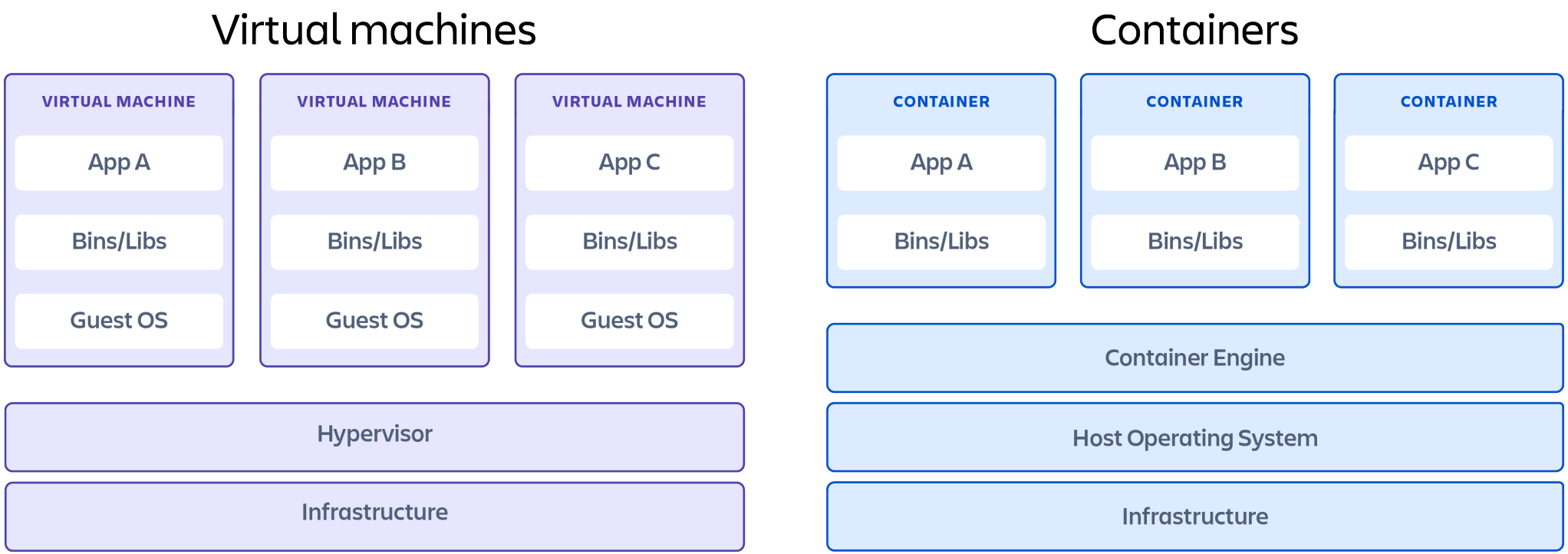
b : a portable compartment in which freight is placed (as on a train or ship) for convenience of movement

https://www.merriam-webster.com/dictionary/container

Container 2/2

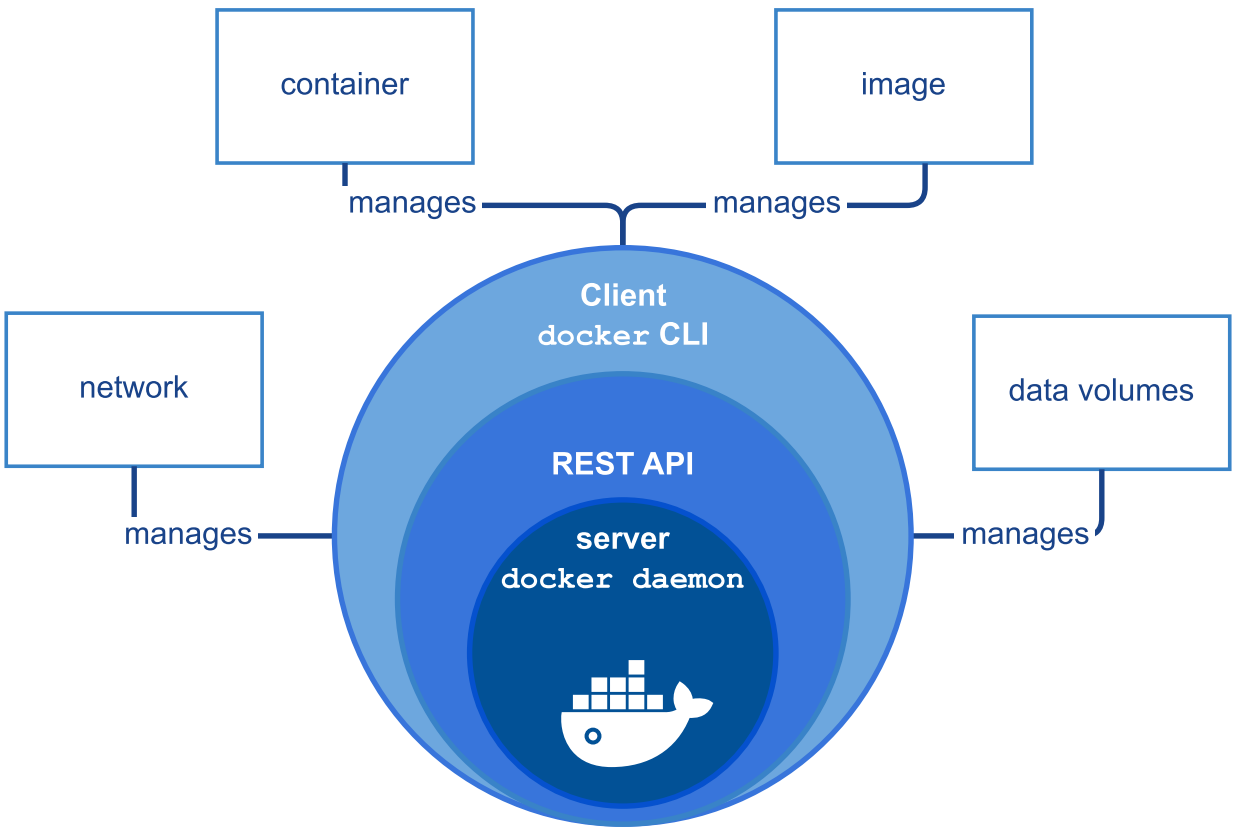
- A container ...
  - is an isolated environment
  - is managed by a host operating system (OS)
  - executes processes
- These processes ...
  - have dedicated resources (file system, files, devices, libraries)
  - cannot directly access resources of the host OS

Virtual Machines vs. Containers



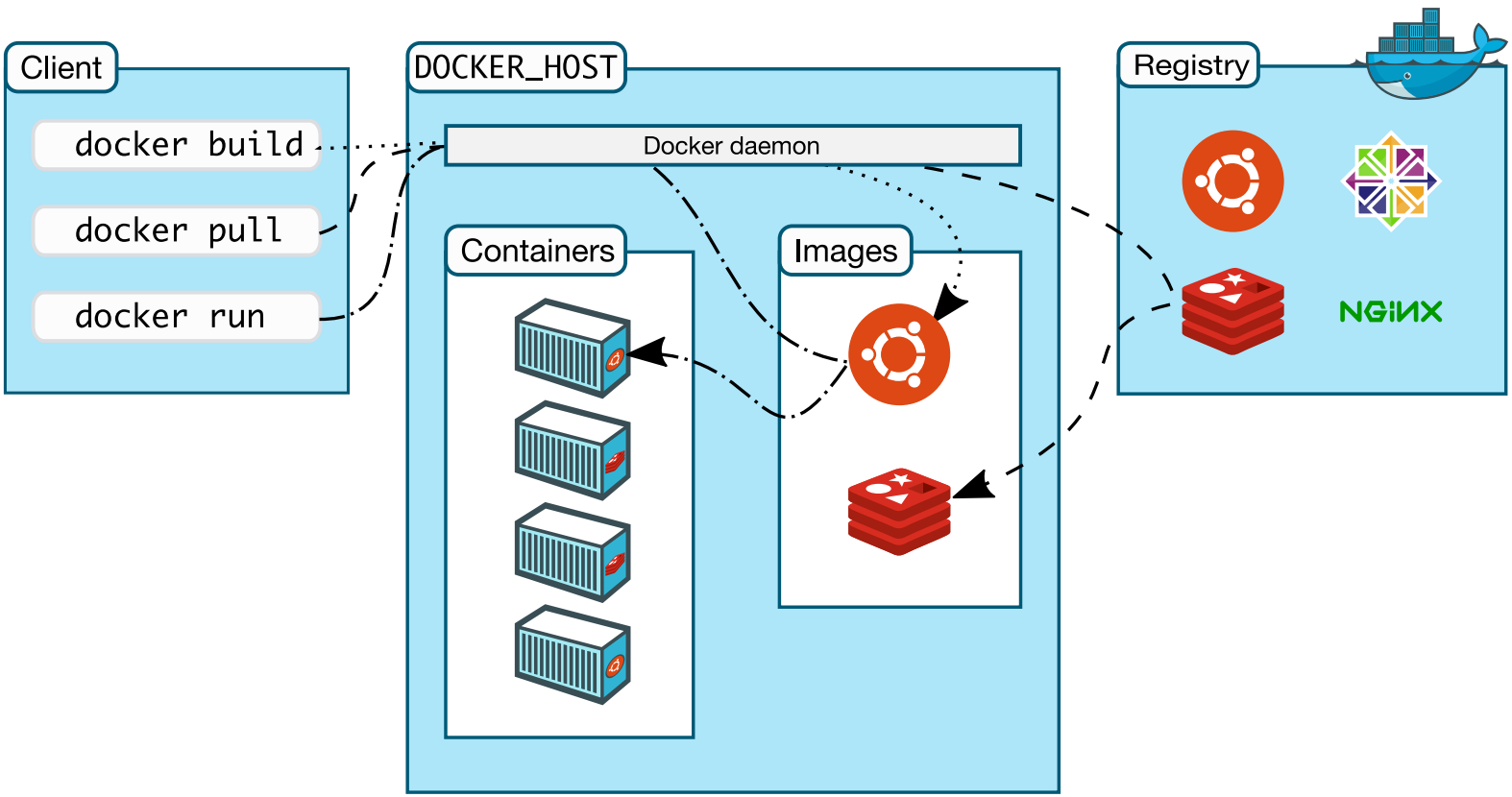
https://www.atlassian.com/microservices/cloud-computing/containers-vs-vms

Overview



https://github.com/docker/docs/blob/6a66f748791143ecfe4c321f9e11fd4b641f8948/content/engine/images/engine-components-flow.png

Docker Architecture



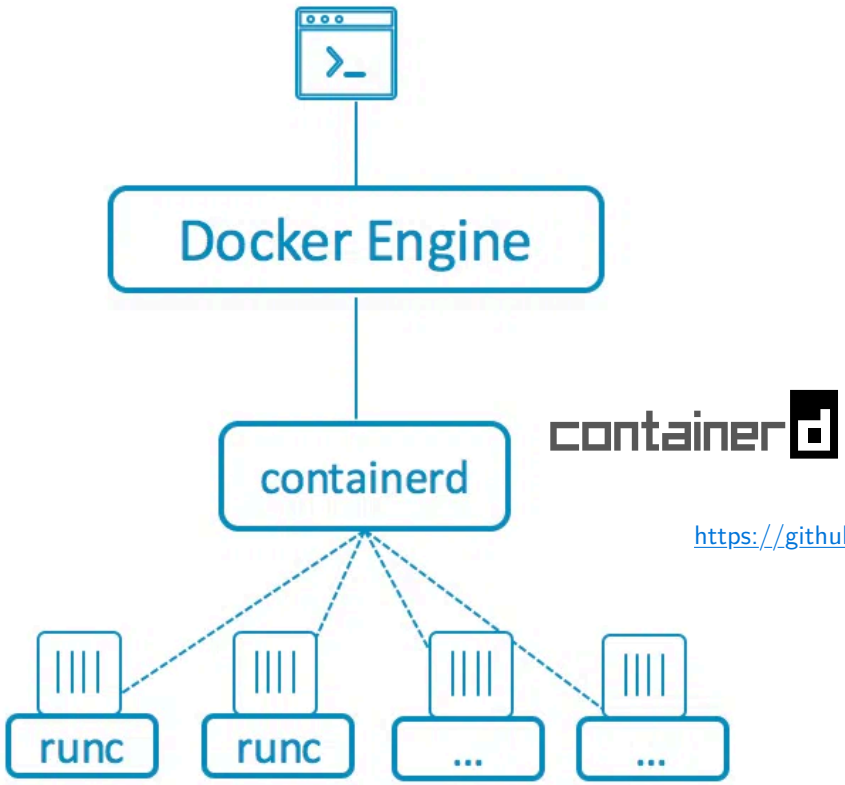
https://github.com/docker/docs/blob/6a66f748791143ecfe4c321f9e11fd4b641f8948/content/engine/images/architecture.svg

Docker Client

- Frontend for users
  - Manage containers, volumes, networks, ...
  - Create images using Dockerfiles
  - Download images from registry
  - ...
  - `docker --help`
- Uses REST API to communicate with Docker daemon
- `docker run debian /bin/echo "Hello, Docker world!"`

Docker Daemon

- Creates and manages containers and images
- Provides REST API
  - Programmatically use Docker
  - Used by Docker CLI
- Container runtime: `containerd` + `runc`



https://github.com/cncf

https://www.docker.com/blog/docker-engine-1-11-runc/

- Alternative container runtimes:

crun

https://github.com/containers/crun

youki

https://github.com/containers/youki

Docker Images and Containers

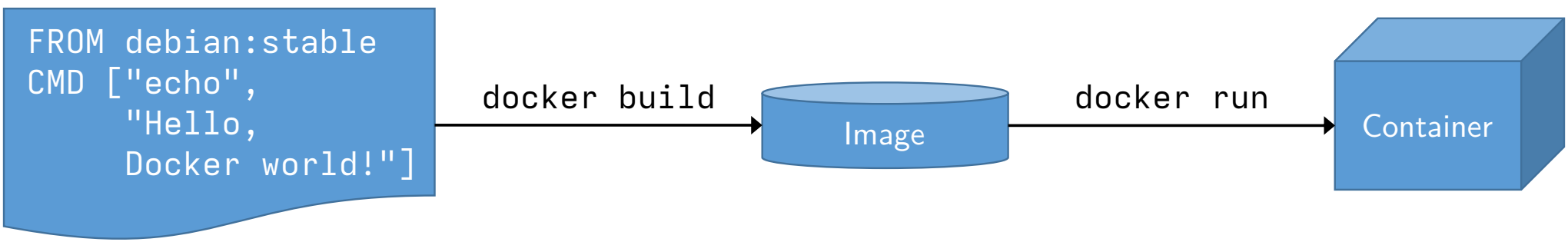
- Preconfigured runtime environment
- Technically, an archive that contains ...
  - one or more file systems with files and
  - metadata
- `docker run` starts containers from images
- A container is a runnable instance of an image
- Images should be considered immutable

Docker Registry

- Service to provide Docker images
- Official Docker registry: https://hub.docker.com/
- Self-hosted registry: https://hub.docker.com/_/registry
- Image references
  - `NAME[:TAG|@DIGEST]`
  - `name` → `debian:latest`
  - `name:tag` → `debian:bullseye`
  - `name@digest` → `debian@sha256:e538a2f...`

Dockerfiles

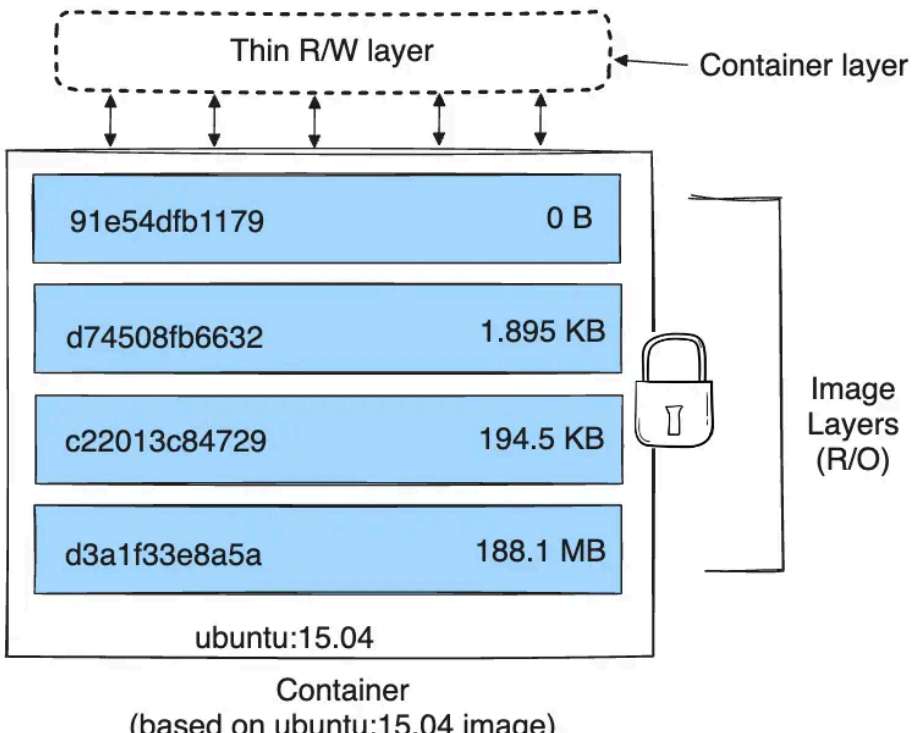
- Contain instructions for creating images
- `docker build` creates an image from a Dockerfile



Copy-on-write Storage

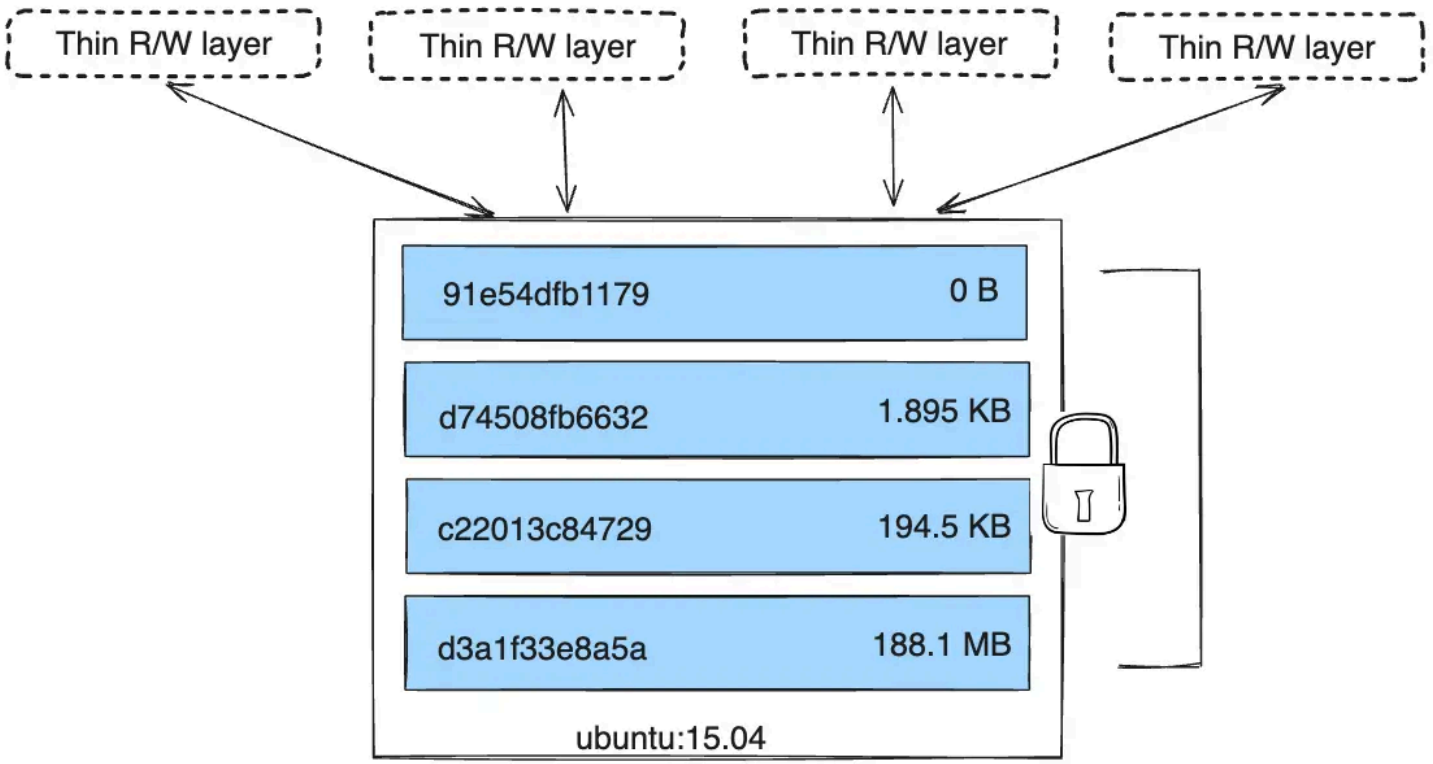
- Union filesystem: Allows to mount and overlay multiple file systems
- Docker: Overlay multiple layers of a docker image
- Read operations executed on existing layers
- Write operations executed on new layer
  - File to be written to is copied first $\rightarrow$ Copy-on-write
- Docker provies multiple storage drivers
  - OverlayFS is default on currently supported Linux distributions

Image Layers



https://docs.docker.com/storage/storagedriver/

Container Layers



https://docs.docker.com/storage/storagedriver/

Shared Layers

debian:latest

```
FROM scratch
ADD rootfs.tar.xz /
CMD ["bash"]
```

Layers (2)

0	ADD file:b4987bca8c4c4c640d6b71dcccfd7172b44771e0f851a47d05c00c2bdcd204f6 in /	116.51 MB
1	CMD ["bash"]	0 B

custom:latest

```
FROM debian:latest
LABEL author='johannes.karder@fh-ooe.at'
COPY ./test.txt /root/test.txt
RUN echo 'hello' > /root/hello.txt
CMD ["date"]
EXPOSE 80
```

Layers (7)

0	ADD file:b4987bca8c4c4c640d6b71dcccfd7172b44771e0f851a47d05c00c2bdcd204f6 in /	116.51 MB
1	CMD ["bash"]	0 B
2	LABEL author=johannes.karder@fh-ooe.at	0 B
3	COPY ./test.txt /root/test.txt # buildkit	16 B
4	RUN /bin/sh -c echo 'hello' > /root/hello.txt # buildkit	6 B
5	CMD ["date"]	0 B
6	EXPOSE map[80/tcp:{}]	0 B

`docker build` and Caching

- `docker build` creates a new layer for various commands within a Dockerfile
- For file operations (e.g. `COPY`) the checksum of the files is saved
- All layers are cached
  - Speeds up build process
  - Allows sharing of layers between different images
  - Speeds up image downloads
  - Reduces size of containers and images

Docker Commands

Build Images

- `docker build <path>`
- Builds an image from a Dockerfile and saves it locally
- Options
  - `-f` / `--file`: Specify Dockerfile (default is “PATH/Dockerfile”)
  - `-t` / `--tag`: Name and optionally tag an image
  - `docker build -t myimage:latest .`

List Images

- `docker images`
- `docker image ls`
- Lists all local Docker images, e.g. saved in /var/lib/docker/image/
- Options
  - `-a` / `--all`: Also list intermediate images

Save / Load Images

- `docker [image] save <image> > image_file.tar`
- `docker [image] save <image> -o image_file.tar`
  - Saves image as tar archive

- `docker [image] load < image_file.tar`
- `docker [image] load -i image_file.tar`
  - Loads image from tar archive

Remove Images

- `docker rmi <image>`
- `docker image rm <image>`
  - Deletes an image

Pull / Push Images

- `docker [image] pull <[url/]tag>`
  - Loads an image from a repository

- `docker [image] push <[url/]tag>`
  - Uploads an image to a repository

Run Containers 1/2

- `docker run <name[:tag]> [command] [args]`
- Starts an image, and optionally a command within the resulting container
- Creates a writable layer
- Options
  - `-i` / `--interactive`: Leaves open STDIN even if not attached
  - `-a` / `--attach`: Attach to STDIN/STDOUT/STDERR
  - `-t` / `--tty`: Opens a shell for input/output
  - `-v` / `--volume`: Mounts directories/files/volumes

Run Containers 2/2

- Options
  - `-p` / `--publish`: Opens ports of the container to the host OS
  - `-d` / `--detach`: Starts container in the background
  - `--rm`: Automatically removes container when it exits
  - `-e` / `--env`: Sets environmental variables within the container
  - `--name`: Sets a name (can be used in other commands, e.g. start/stop/etc.)
  - Many more options to constrain resources
    - `-c` / `--cpu-shares`
    - `--cpus`
    - `-m` / `--memory`

Create Containers

- `docker container create [options] <image> [command] [args]`
- Creates a writable layer for a container on top of an image
- Can be started with `docker start`
- Options: see `docker run`

Start / Stop

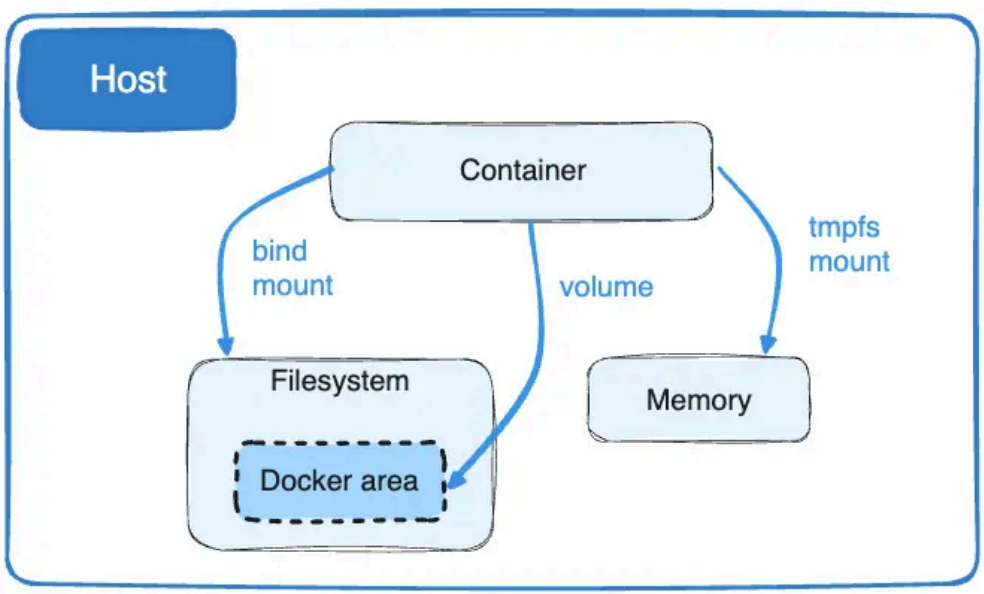
- `docker stop <container>`
  - Stops a running container
  - Sends SIGTERM to all container processes and SIGKILL after timeout
- `docker kill <container>`
  - Sends SIGKILL to all container processes
- `docker start <container>`
  - Starts a stopped container
- `docker pause <container>`
  - Pauses processes within the container (cgroups freezer)
- `docker unpause <container>`

Copy To / From Containers

- Copy directories/files between container and host
  - `docker [container] cp <container>:src dest`
  - `docker [container] cp src <container>:dest`
- Example
  - `docker cp container1:/root/notes.txt notes.txt`

Mounts

- Allow to
  - ... persist data (volumes)
  - ... access/exchange data with host system (bind mounts)
  - ... temporary store data (tmpfs)



https://docs.docker.com/storage/volumes/

Volumes

- Managed by Docker
- Mounted in containers
- Data remains persisted across container restarts
- Format: `volume_name:container_dir[:options]`
- Example:
  - `docker volume create my_volume`
  - `docker run -v my_volume:/root`

Volume

- `docker volume`
  - `create <name>`: Creates a named volume
  - `rm <name>`: Deletes a volume
  - `ls`: Lists created volumes
  - `prune`: Deletes all unused volumes

Bind Mounts

- Mount files of folders of host system in container
- Format:
  - `host_dir:container_dir[:options]`
  - `host_file:container_file[:options]`
- Example:
  - `docker run -v $PWD/notes.txt:/root/note.txt ...`
  - ```
    docker run --mount \
      type=bind,\
      source=$PWD/notes.txt,\
      target=/root/note.txt ...
    ```
    ```

Tmpfs

- Allows mounting in-memory file system
  - which is outside the writable layer of a container
- Tmpfs data is lost upon stopping the container
- Tmpfs mounts
  - ... cannot be shared between containers
  - ... only work on Linux
- Example:
  - `docker run -tmpfs /logs ...`

Ports

- Open container ports to host system
- Access applications bound to ports within container
- Format:
  - `host_ports:container_ports`
- Example:
  - `docker run -p 1234:80 ...`
  - `docker run -p 1234-1236:1234-1236 ...`

Ps

- `docker ps` lists all running containers
- Options:
  - `-a` / `--all`: list all containers, including the ones not running
  - `-l` / `--latest`: show last started container
  - `-s` / `--size`: show size of containers
    - Size: Size of writable layer
    - Virtual: Size of read-only image + writable layer

Attach / Exec

- `docker attach <container>`
  - Connects to STDIN, STDOUT and STDERR of container
- `docker exec <container> command [args]`
  - Executes command in running container

Top / Stats

- `docker top <container>`
  - Lists running processes of container
- `docker stats [<container>]`
  - Lists resource usage statistics of container(s)
  - CPU, memory, network I/O, disk I/O

Networking

- Multiple networking options
  - Extendable via plugins
- Default: bridge
  - Default network for containers – if not specified otherwise
  - No open ports
    - Between containers
    - Between container and host system
  - No name resolution between containers
    - Access only possible via IP addresses

User-Defined Bridges

- Assignment of containers to network
- Ports between containers are open
- Ports to host system must be opened explicitly
- DNS available
  - Container name == DNS name
- Container can be added/removed to/from user-defined bridges while running

Network

- `docker network create <network>`
  - Creates a new network (default: bridge)
- `docker network [dis]connect <network> <container>`
  - Adds/removes container to/from network
- `docker network rm|ls|prune`
  - See docker volume
- `docker run --net[work] <network> ...`
  - Starts a container within network
  - `docker run --network none ...` disables networking

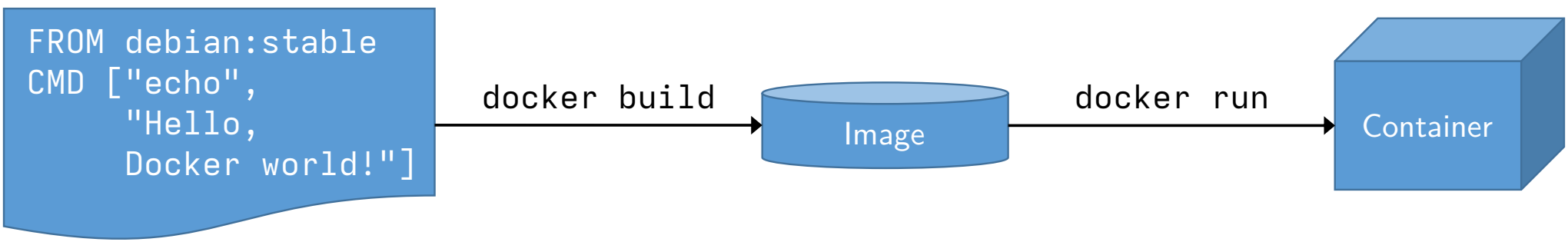
Dockerfiles

Build Images

- `docker build <path>`
- Builds an image from a Dockerfile and saves it locally
- Options
  - `-f` / `--file`: Specify Dockerfile (default is “PATH/Dockerfile”)
  - `-t` / `--tag`: Name and optionally tag an image
  - `docker build -t myimage:latest .`

Dockerfiles

- Contain instructions for creating images
- `docker build` creates an image from a Dockerfile



FROM

- Defines the base image
  - `FROM <image[:tag]> [AS <name>]`
- Example:
  - `FROM debian:latest`
- Name is required for multi-stage builds, e.g.
  - `COPY --from=<name> ...`
- Multi-stage builds ($>$ v17.05)
  - Multiple `FROM`s possible
  - Commands after last `FROM` make up final image

RUN

- Executes a command within image
- Used to configure image
  - Install software
  - Configure software
  - ...
- Examples:
  - `RUN apt-get install nginx`
  - `RUN dotnet build`

WORKDIR

- `WORKDIR <path>`
- Defines working directory for commands
  - Used to expand relative paths in `COPY`, `RUN`, `CMD`, `ADD`, `ENTRYPOINT`
- Example:
  - `WORKDIR /root`
  - `COPY /root/file.txt .`

COPY

- `COPY <src> <dest>`
- Copies files/directories from host system to image
  - Makes files available within container
  - Directories are created if non-existent
  - Multiple `<src>` may be defined
    - `<dest>` must then be a directory and end with a slash "/"
- Examples:
  - `COPY ./notes.txt /root/`
  - `COPY ./NotesAPI/ .`
  - `COPY ./NotesAPI/ /root/NotesAPI/`
  - `COPY files* /root/`
  - `COPY ["a.txt", "b.txt", "/root/"]`

ADD

- Like COPY, but
  - … if `<src>` is an URL, downloads the file and saves it to `<dest>`
  - … if `<src>` is a tar archive, extracts it in `<dest>`
- Examples:
  - `ADD https://planet.osm.org/planet/planet-latest.osm.bz2 /root/`
  - `ADD rootfs.tar.xz /`

ENTRYPOINT

- Defines the executable to start once the container is started
- Formats:
  - Shell form: `ENTRYPOINT command param1 param2 ...`
  - Exec form: `ENTRYPOINT ["executable", "param1", "param2", ...]`
- Arguments to `docker run <image> ...`
  - ... are appended after all arguments in exec form
  - ... override all elements specified using CMD
- Override `ENTRYPOINT` using `docker run --entrypoint ...`

CMD

- Defines the program to be executed
- Defines default parameters for `ENTRYPOINT`
- Format:
  - Exec form: `CMD ["executable", "param1", "param2", ...]`
  - Shell form: `CMD command param1 param2 ...`
  - Parameter for `ENTRYPOINT`: `CMD ["param1", ...]`
- Exec form does not start a shell
  - No substitution, e.g. no shell variables are evaluated

ENTRYPOINT, docker run and CMD

- `ENTRYPOINT` in exec form:
  - Parameters of `CMD` are appended to `ENTRYPOINT`
  - Parameters of docker run are appended to `ENTRYPOINT`
- `ENTRYPOINT` in shell form:
  - Parameters of `CMD` / `docker run` are not appended
  - `ENTRYPOINT` is started as sub-process of /bin/sh
    - Application will not receive SIGTERM from docker stop

ENTRYPOINT - Example

```
FROM debian:stable
ENTRYPOINT ["/bin/echo"]
CMD ["hello, cmd"]
```

```
$ docker run ...                    # prints "hello, cmd"
$ docker run ... "hello, docker" # prints "hello, docker"
```

ENV

- Sets environmental variables
- `ENV <key>=<value>`
- `ENV <key> <value>`
- Environmental variables are available in running container
- `docker run --env <key>=<value>`
- Examples:
  - `ENV DOTNET_SDK_VERSION=1.0.4`
  - `ENV DOTNET_SDK_VERSION 1.0.4`

ARG

- Defines variables that can be used during build time
- `ARG <name>[=<default value>]`
- Variables are not available in running container
- For multi-stage builds, they must be defined in every stage for them to be used
- User can specify values:
  - `docker build --build-arg <key>=<value>`
- Environmental variables override build variables

EXPOSE

- `EXPOSE <port> [<port>/<protocol> ...]`
- Only informs that container will listen on specified ports at runtime
- Does not publish port
- Must be done manually via `docker run -p ...`

Docker Compose

Docker Compose

- Definition, creation and execution of multi-container applications
- Definition via YAML file
- Specify required resources via configuration vs. `docker run` arguments
  - Port mappings
  - Networks
  - Volumes
  - ...

Structure

```
services:
  service1:
    ...
  service2:
    ...

volumes:
  ...
networks:
  ...
```

- One service corresponds to one Dockerfile/Image
- Configuration of each service entry corresponds to docker run/create/start arguments
- Sections for volumes and networks

`docker compose`

- `docker compose build`
  - Builds all images (executes build steps in YAML)
- `docker compose up`
  - Creates and starts all containers (services)
  - `-d`: Executes containers in background
- `docker compose down`
  - Stops all containers

`docker compose up`

- Creates containers at first start (`docker run`)
- Starts containers if they exist and are unchanged (`docker start`)
  - Writable layer remains as long as container exists
  - Can lead to hard to diagnose problems in docker compose files with many services
  - `--force-recreate`: always recreates containers

Example

```yaml
services:
  db:
    container_name: mydb
    image: postgres
    volumes:
      - ./tmp/db:/var/lib/postgresql/data
  web:
    build: .
    command: bundle exec rails s -p 3000 -b '0.0.0.0'
    volumes:
      - .:/myapp
    ports:
      - "3000:3000"
    depends_on:
      - db
```

Further Reading

- Docker documentation
  - https://docs.docker.com/get-started/docker-overview/
- Dockerfile reference
  - https://docs.docker.com/reference/dockerfile/
- Compose file reference
  - https://docs.docker.com/reference/compose-file/