Docker

(Falls Docker per Shell Script in wsl installiert wird: Das Script benutzt Windows LineIns, bedeutet, dass Linux es nicht lesen kann. Bevor man es mit `./install-docker.sh` ausführt muss man `dos2unix install-shell.sh` verwenden um die LineIns zu machen.)

Docker Theorie

Docker ist ein essenzielles Tool in der Softwareentwicklung, dass es erlaubt, Tooling für das Ausführen von Code auch auf Maschinen zu benutzen, die diese Tools eigentlich nicht installiert haben.

Diese Werkzeuge werden in sogenannten **Dockercontainern** gespeichert.

Docker Images

Docker Images sind ein eigenes kleines Filesystem und eine Autorangabe, welche vom System interpretiert werden können. Diese Systeme enthalten Programme, Files, Befehle etc.

Docker Images sind, einmal erstellt, immutable, also nicht mehr veränderbar. Wenn man Änderungen durchführen möchte, muss man ein neues Image erstellen.

Docker Container

Container: *portable compartment in which freight is placed for convenience of movement*

Ein Docker Container ist eine **abgeschirmte Ausführungsumgebung**.

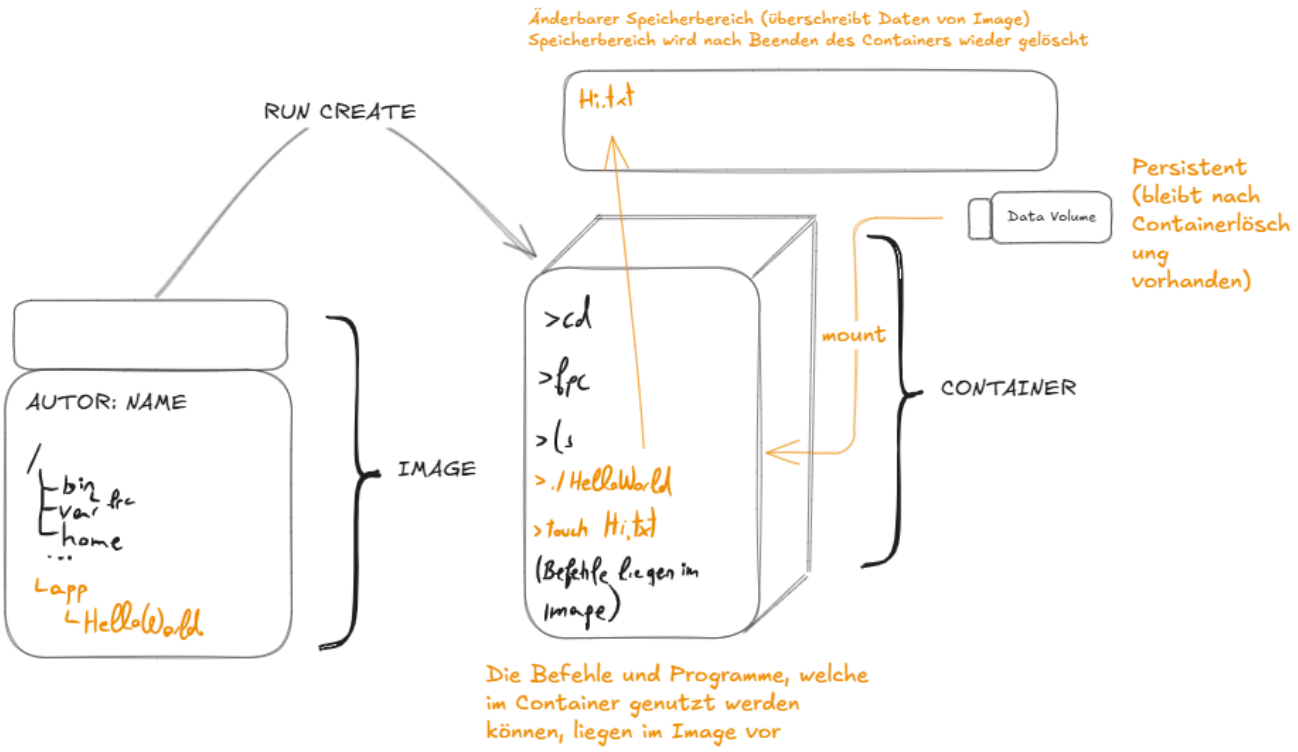
Images können mit `Run / Create` in einem **Docker-Container** ausgeführt (wie eine Shell, basically wie in wsl). An sich ist der Docker Container leer, alle Befehle, Programme, etc. liegen in den Images vor und können im Container ausgeführt werden.

Wenn im Container ausgeführt werden, die das Dateisystem verändern (zB. `touch Filename.txt`) dann entsteht für diesen Container ein eigener Speicherbereich an dem diese Änderungen abgelegt werden.
Das Image wird davon **NICHT VERÄNDERT (immutable)**

Dieser Speicherbereich existiert nur so lange wie der Container. Wird der Container gelöscht und ein neuer Container geöffnet, wird ein neuer Speicherbereich definiert, der alte Speicher geht verloren.

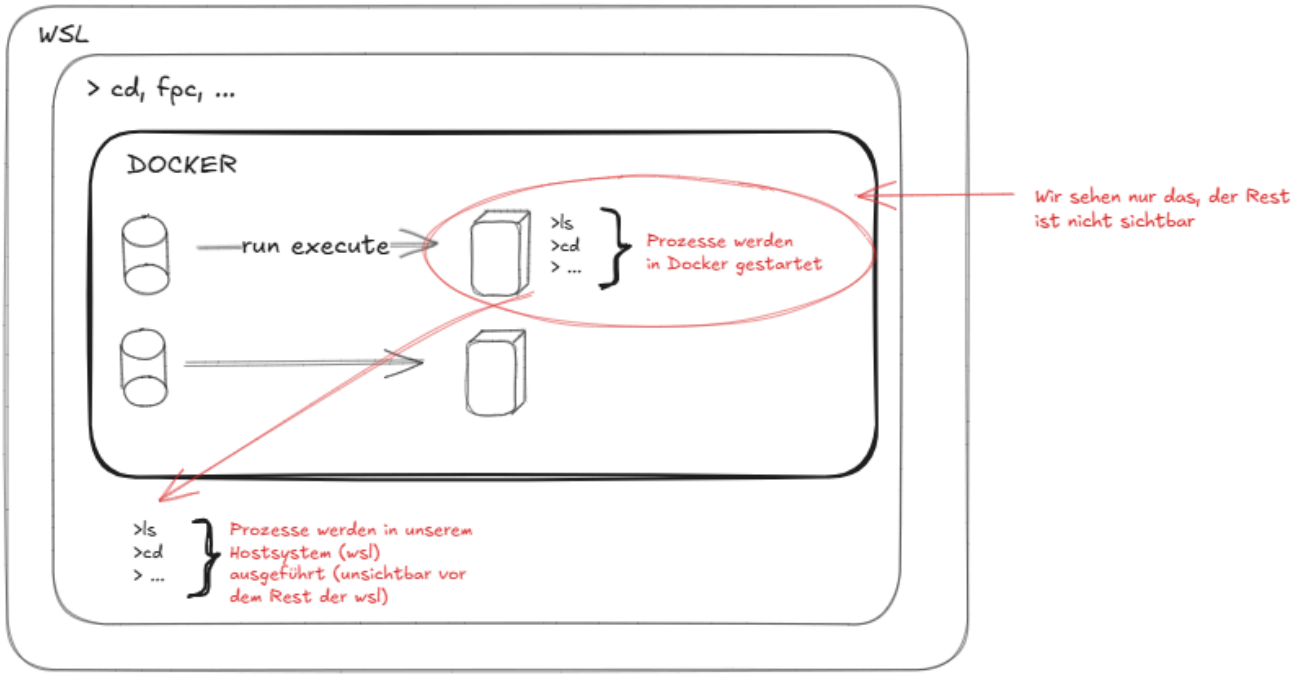
Data Volumes

Data Volumes, quasi kleine Speicherbereiche, können in einen Container **gemountet** werden (wird beim Start spezifiziert), analog zu einem USB Stick bei einem PC. Diese Dateien sind nach dem Mount im Container verwend- und veränderbar. Diese Änderungen in den Data Volumes bleiben auch nach dem Schließen des Containers vorhanden (wie bei einem USB Stick)



Vorteil: Jeder hat die exakt gleiche Architektur (dank dem Image) und identische File-System. -> works on my machine wird dadurch nicht mehr zum Problem.

Docker Container laufen in Docker **abgeschottet** von dem Rest des OS. Die Befehle werden zwar im OS ausgeführt, aber wir können das nicht sehen und nicht bearbeiten.



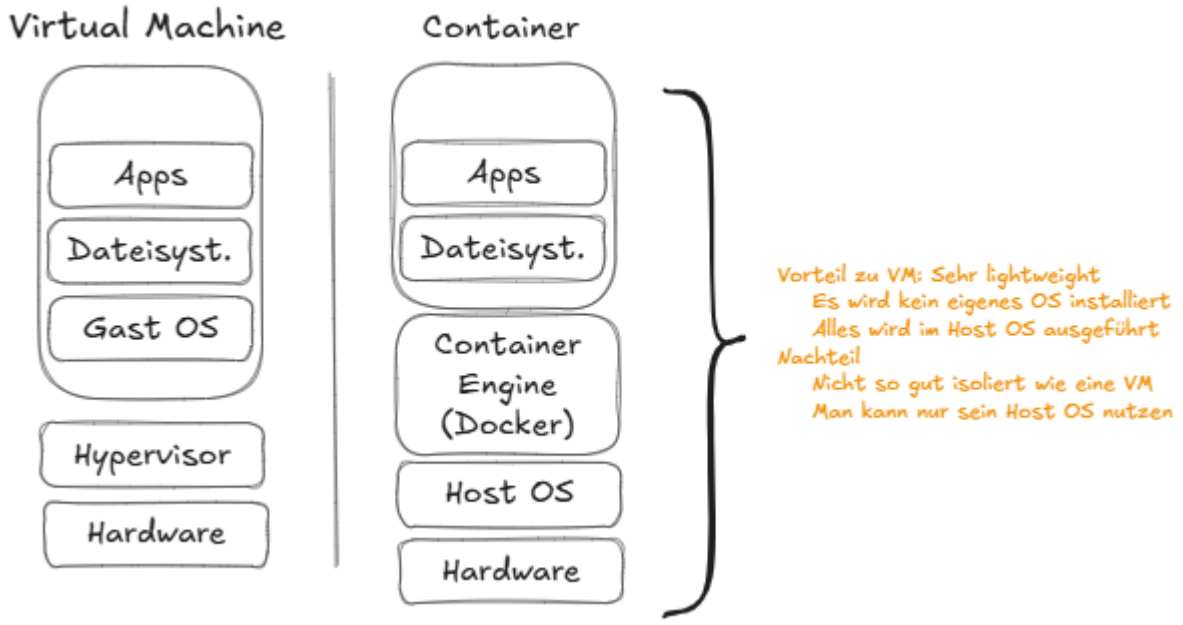
Container Registries / Docker HUB

Die Docker Registries / Docker Hub sind online Server, auf welchen eigene Images hoch- und Images von anderen heruntergeladen werden können.

Virtual Machine / Container

Bei einer Virtual Machine wird die reale Hardware abstrahiert. Es werden virtuelle Versionen der Hardware zur Verfügung gestellt, auf welchem dann ein eigenes 'Guest OS' laufen kann. Dadurch kann ich zB Linux auf einem Windows PC laufen lassen. Für das Guest OS ist nur der Hardwareanteil sichtbar, den ich bei der Erstellung einer VM zuweise.
Der **Hypervisor** ist für die Abstraktion dieser realen Hardwareteile zuständig.

Container sind um einiges mehr leightweight als VMs. Sie haben kein eigenes OS, sondern laufen komplett auf dem bereits installierten Betriebssystem. Die Isolation ist allerdings nicht so hoch wie auf einer VM.



Docker CLI

Über die Docker CLI kann ich Images und Container verwalten, Netzwerk und Datenmengen verwalten und über die **REST API** mit dem **Docker Deamon** (Server) kommunizieren. Dieser gibt uns Antwort auf unsere Befehle. Docker ist eine **Serveranwendung**. Meist werden Images auf einem Server gehostet um dann von vielen PCs verwendet werden zu können.

Bind Mounts

Ermöglicht das mounten von absoluten Pfaden hinaus aus dem Container in das Betriebssystem, was es ermöglicht Änderungen in einem OS Ordner live im Container zu haben und zu bearbeiten.

Dockerfiles

Dockerfiles enthalten Instruktionen für die Erstellung von Images. Diese werden auf der obersten Ebene der gewünschten Datenstruktur abgelegt, und mit `docker build -t ImageName:Version .` gebaut. (Der Punkt referenziert das derzeitige Arbeitsverzeichnis als 'höchstes' Verzeichnis -> hier liegt Dockerfile, wird zu root)

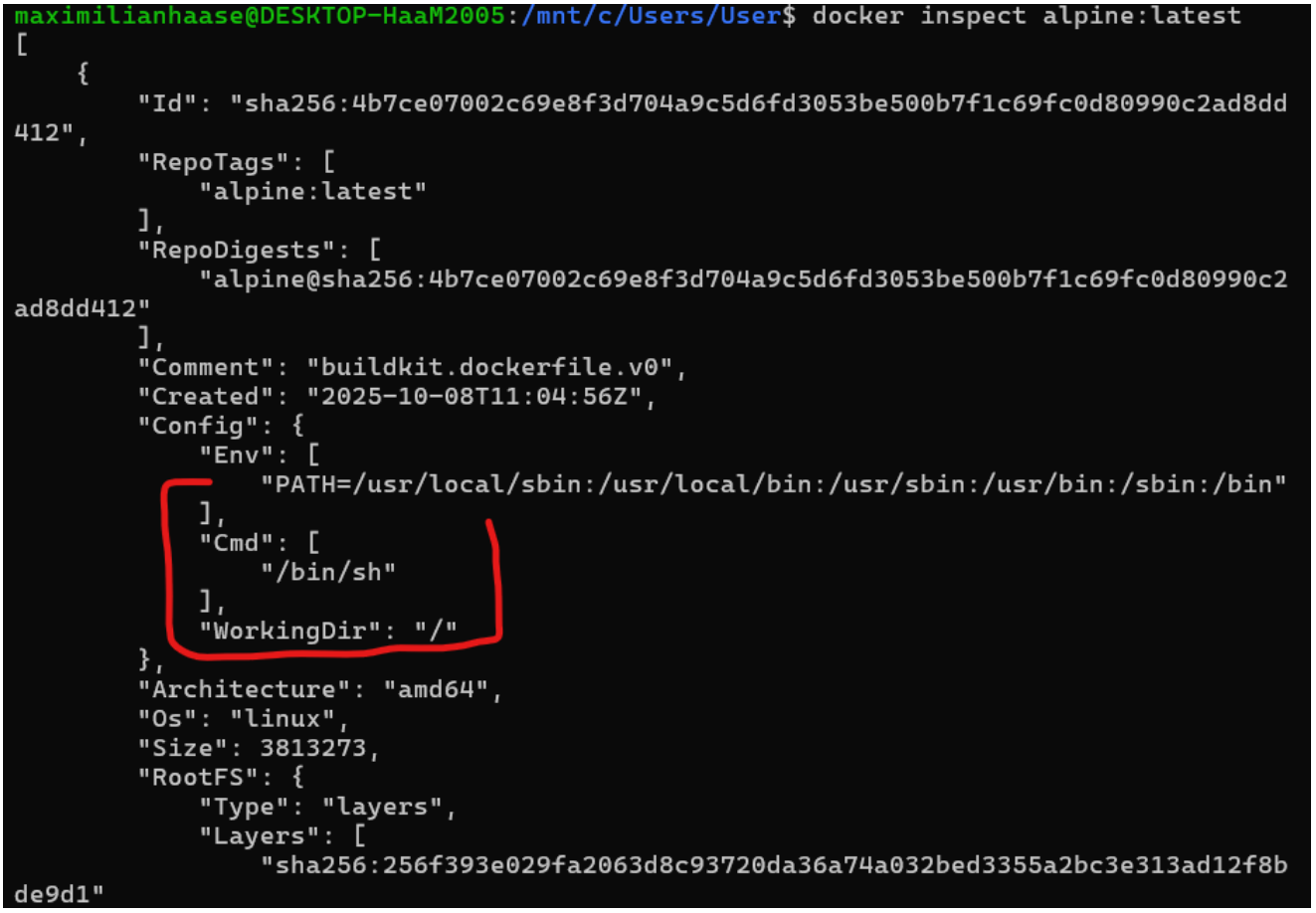
Docker Praxis

Wichtige Docker Commands (Docker CLI)

Infobefehle

- `docker version`
  - gibt die Versionsinformationen der Docker Installation zurück
- `docker info`
  - gibt wichtige Infos (zB laufende Container, Imageanzahl, Plugin infos, REST API URL etc.)
- `docker ps`
  - gibt eine Liste aller laufenden Container im System
  - `-a`
    - Zeigt eine Liste aller Container (auch ausgeschalten)
  - `-l`
    - Zeigt letzten erstellten container
- `docker images`
  - Zeigt eine Liste aller Images an
- `docker inspect ImageName:Version`
  - Gibt Einstellungsinfos, Metainformationen, Architekturinfos und Konfigurationen
    - Config beinhaltet: Cmd & Entrypoint
      - cmd sagt uns welcher Prozess im Container gestartet wird (/bin/sh -> shell (PID 1 -> ist immer __der erste Befehl der ausgef. wird))

Output von `docker inspect alpine:latest`



```
maximilianhaase@DESKTOP-HaaM2005:/mnt/c/Users/User$ docker inspect alpine:latest
[
    {
        "Id": "sha256:4b7ce07002c69e8f3d704a9c5d6fd3053be500b7f1c69fc0d80990c2ad8dd
412",
        "RepoTags": [
            "alpine:latest"
        ],
        "RepoDigests": [
            "alpine@sha256:4b7ce07002c69e8f3d704a9c5d6fd3053be500b7f1c69fc0d80990c2
ad8dd412"
        ],
        "Comment": "buildkit.dockerfile.v0",
        "Created": "2025-10-08T11:04:56Z",
        "Config": {
            "Env": [
                "PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin"
            ],
            "Cmd": [
                "/bin/sh"
            ],
            "WorkingDir": "/"
        },
        "Architecture": "amd64",
        "Os": "linux",
        "Size": 3813273,
        "RootFS": {
            "Type": "layers",
            "Layers": [
                "sha256:256f393e029fa2063d8c93720da36a74a032bed3355a2bc3e313ad12f8b
de9d1"
```

Arbeiten mit IMAGE / CONTAINER / VOLUMES

- `docker pull ImageName`
  - Zieht ein gegebenes Image vom Docker Daemon Server
- `docker run` / `docker run -ti ImageName:Version` (run = `docker create + docker run`)
  - Startet ein Image in einem Container (endet sofort, wenn kein Befehl kommt. Container wechselt in stop status, wenn PID 1 beendet wird (cmd: /bin/bash))
  - `--rm`
    - Wenn der container beendet wird, dann wird er automatisch gelöscht
  - ``--name Name
    - gibt dem Container einen Namen
  - `-i`
    - "interactive": sorgt dafür, dass Container auf Eingaben wartet (stdStream)
  - `-t`
    - "tty": gibt uns ein 'Pseudo Terminal' in das wir Befehle für den Container schreiben können.
  - -v
    - Mountet ein Volume an den Container
    - Name:/MountDirectory (MD -> in welchem Ordner das Volume eingebunden wird)

- `docker run --rm -ti -v wse-volume:/data alpine:latest`

Wichtig! Die Starteigenschaften werden bei der Erstellung im Container gespeichert -> Muss nur beim erstellt werden angegeben werden, danach nicht mehr!

- `docker rm Name`
  - Löscht einen Container
  - `docker rm ID`
    - löscht container mit der ID
- `docker start Name / ID`
  - Startet einen existierenden Docker Container, aber verbindet sich nicht automatisch mit meiner Shell
- `docker attach Name / ID`
  - Verbindet sich mit der Inputmethode im Container

- `docker volume create Name`
  - Erstellt ein neues Volume
- `docker run -v VName:/Dir Name Image`
  - Erstellt ein neues Volume VName und mounted dieses in das Verzeichnis /Dir im Container Name
-

Arbeiten in der Pseudo Shell

Analog zu Linux, je nachdem, welche Befehle im Image vorliegen. TTY kann mit Exit verlassen werden, dann schließt sich die Shell und wir kommen zurück in die WSL Shell