Docker & Kubernetes
Teil 3

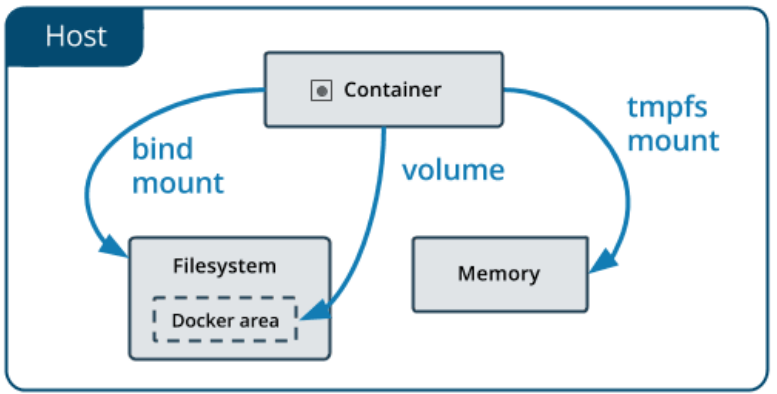
Teil 3 – Installation und Nutzung

- Docker
  - Ressourcen (Netzwerk, Volumes)
  - Logging
  - Registry
  - Beispiele
- Portainer
- Docker Compose

Docker CLI – *volume* Kommando

- Volumes (Wiederholung :-)
  - Im Host-Filesystem
  - Zugriff nur aus Containern
  - ***Verwaltung durch Docker***

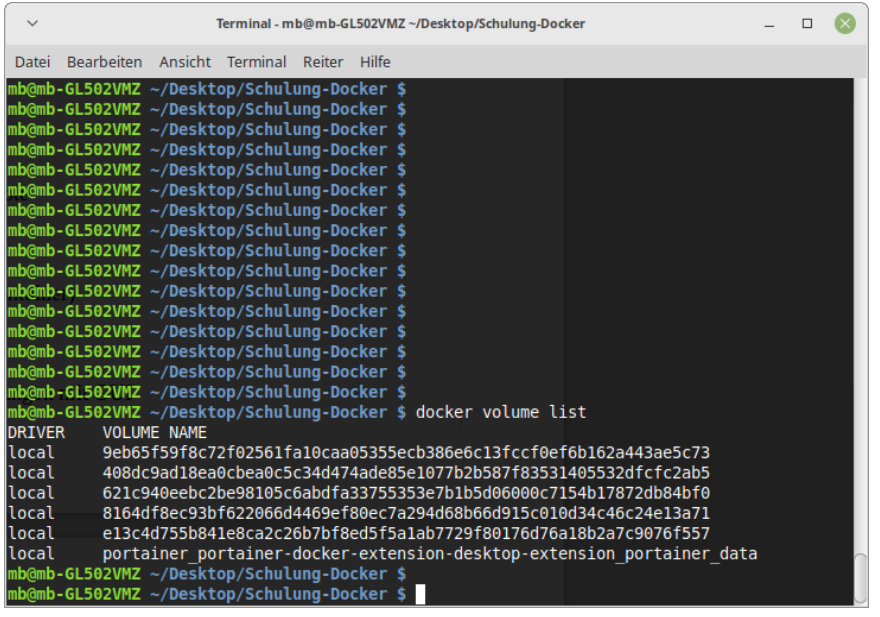
CLI-Kommandos



Docker CLI – *volume* Kommando

- Volumes auflisten

```
$ docker volume list
```



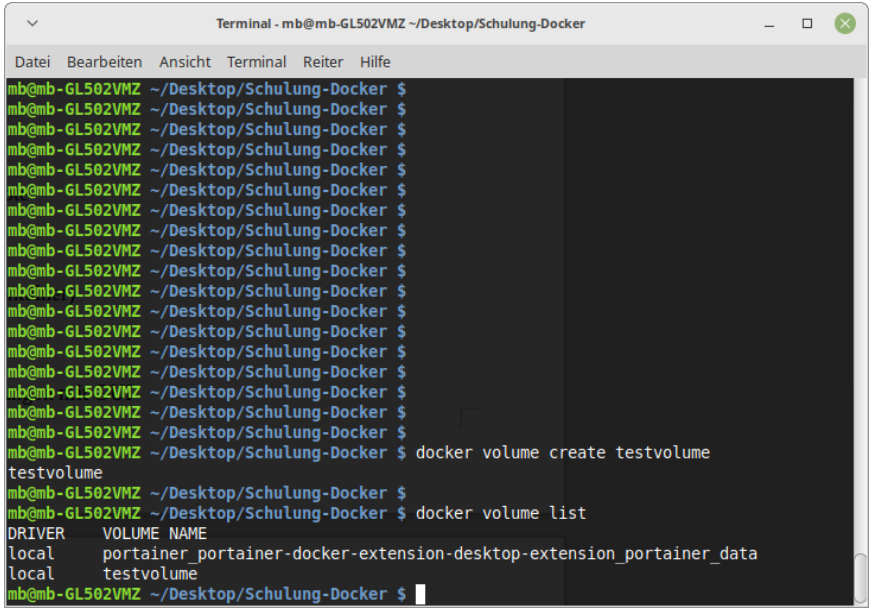
Docker CLI – *volume* Kommando

- Ein Volume erzeugen

```
$ docker volume create [OPTIONS] [<volume name>]

$ docker volume create testvolume
```

Mit der Option „--driver“ kann ein Treiber angegeben werden. Die in Docker installierten Volume-Treiber können mit „docker info“ ausgelesen werden.

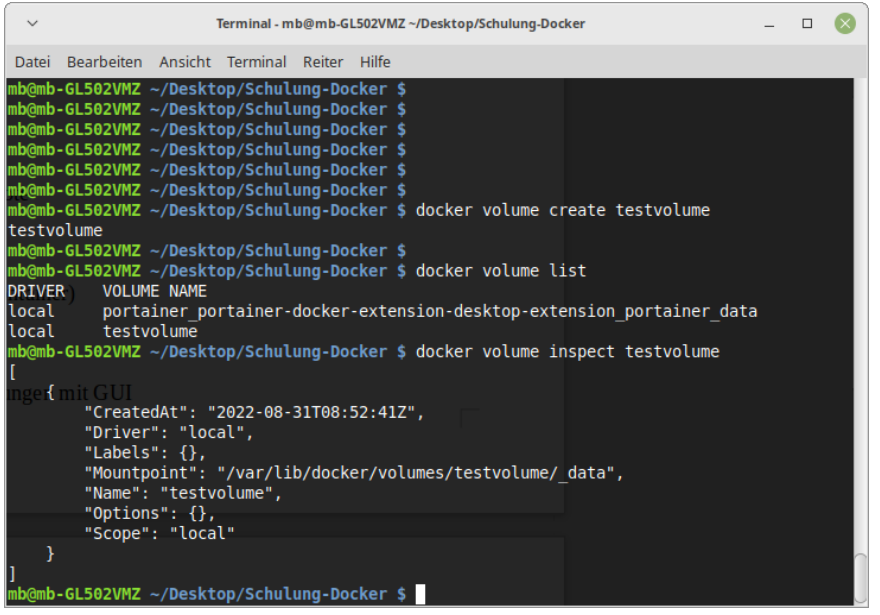


Docker CLI – *volume* Kommando

- Ein Volume untersuchen

```
$ docker volume inspect <volume name>

$ docker volume inspect testvolume
```



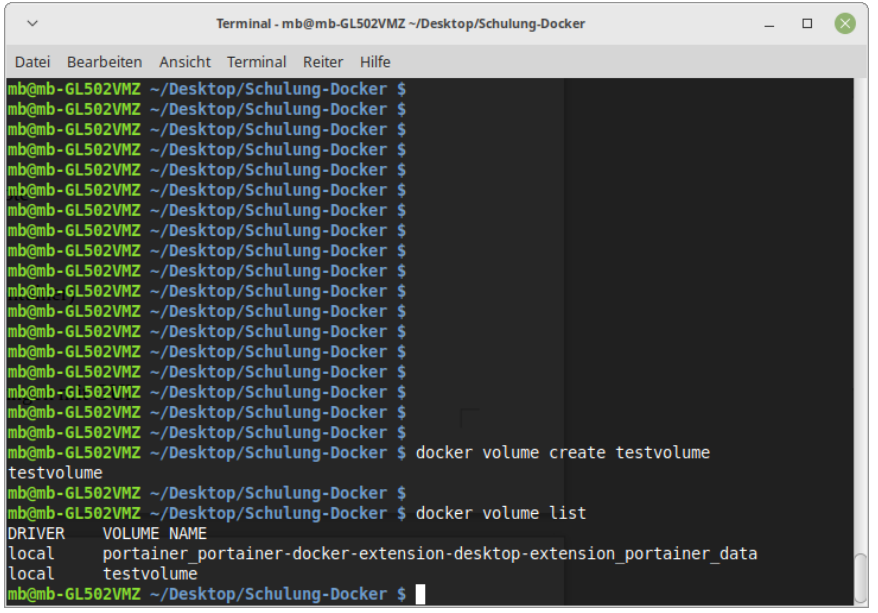
Docker CLI – *volume* Kommando

- Ein Volume löschen/entfernen

```
$ docker volume rm <volume name>

$ docker volume rm testvolume
```

Von Containern genutzte Volumes können nicht gelöscht werden.

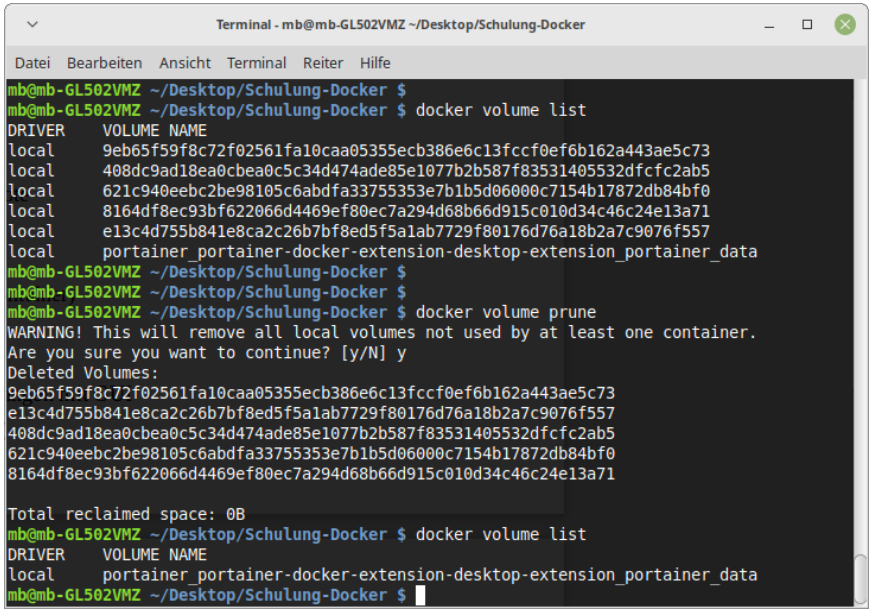


Docker CLI – *volume* Kommando

- Volumes aufräumen

```
$ docker volume prune
```

Prune entfernt alle ungenutzten, lokalen Volumes.



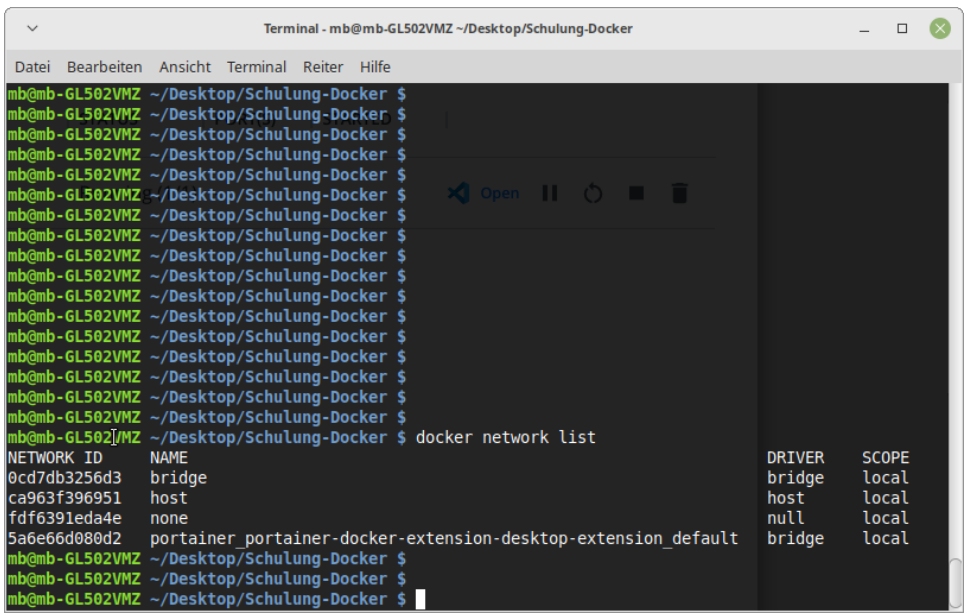
Docker CLI – *network* Kommando

- Netzwerke (Wiederholung :-)
  - Treiber
    - Bridge (default)
      - Mehrere Container sollen auf einem Docker-Host kommunizieren
    - Host – keine Isolierung zwischen Host und Container
    - Overlay – für Kommunikation in Clustern (multi-host)
      - Container auf mehreren Docker-Hosts sollen miteinander kommunizieren
    - None
    - Macvlan – Container bekommen eigene MAC-Adressen
  - Installierte Treiber mit „docker info“ auslesbar

Docker CLI – *network* Kommando

- Netzwerke auflisten

```
$ docker network list
```



Docker CLI – *network* Kommando

- Ein Netzwerk erzeugen

```
$ docker network create [OPTIONS] [<network name>]

$ docker network create --driver bridge testnet
```

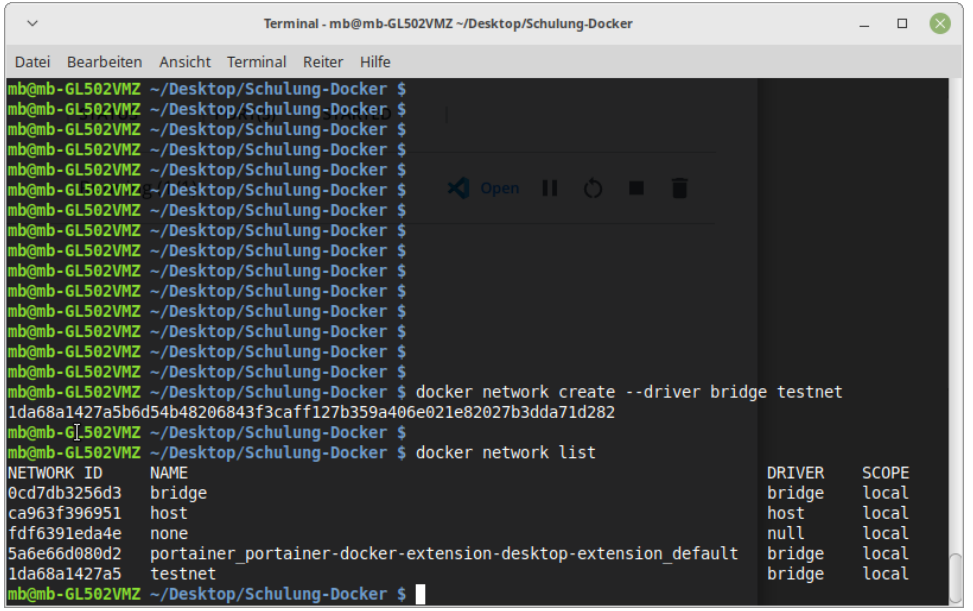
Eigene bridge-Netzwerke besitzen DNS auf Basis der Container-Namen.

Wichtige Optionen:
--driver=<name>
- Den Treiber festlegen

--gateway=<ip-number>
- Das Gateway definieren

--subnet=<subnet>
- Subnet im CIDR-Format

--ip-range=<ip-range>



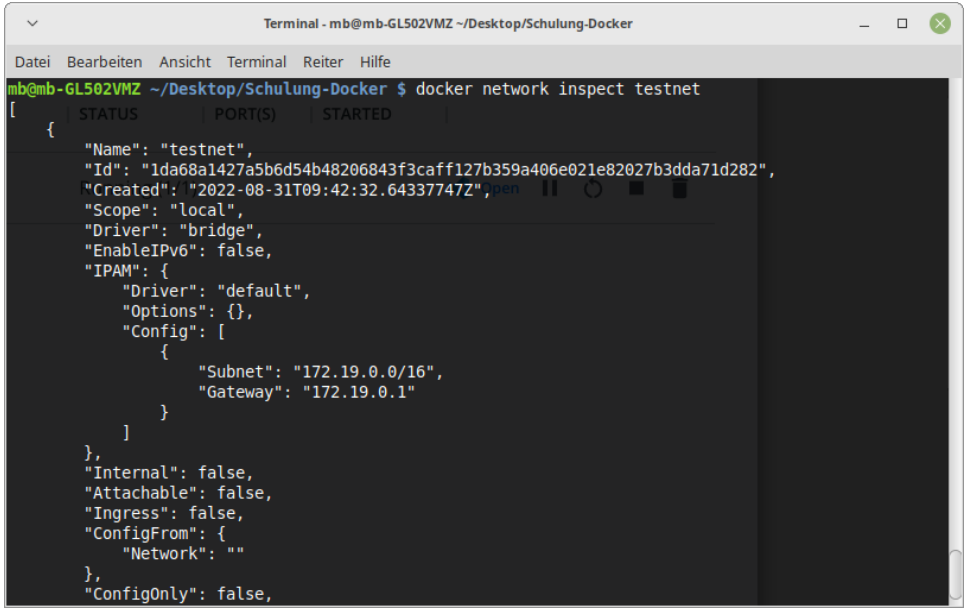
Docker CLI – *network* Kommando

- Informationen über ein Netzwerk

```
$ docker network inspect <network name>

$ docker network inspect testnet
```

Informationen über das Netzwerk, inkl. aller verbundenen Container.



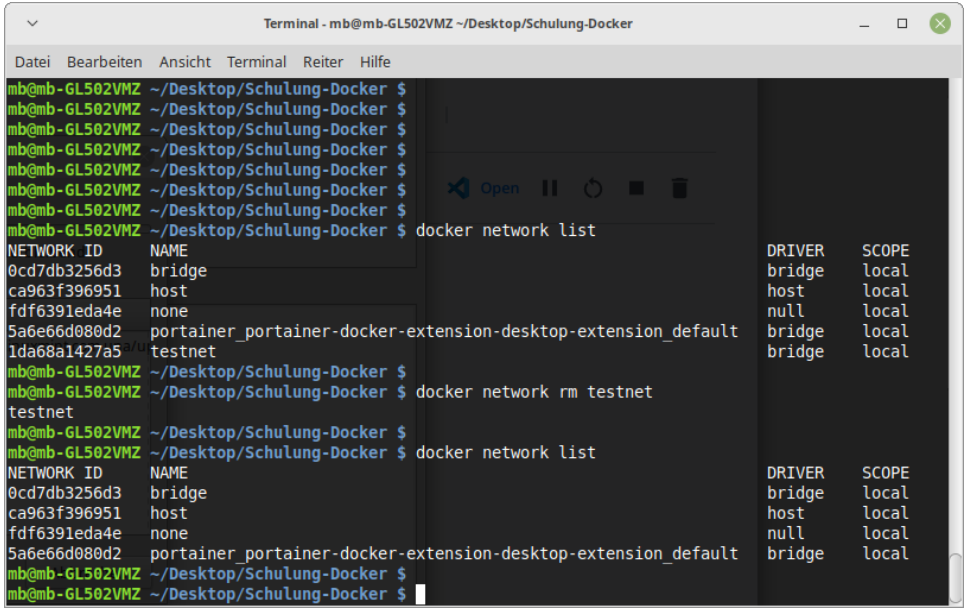
Docker CLI – *network* Kommando

- Ein Netzwerk entfernen

```
$ docker network rm <network name>

$ docker network rm testnet
```

Es können nur Netzwerke entfernt werden, mit denen keine Container mehr verbunden sind.

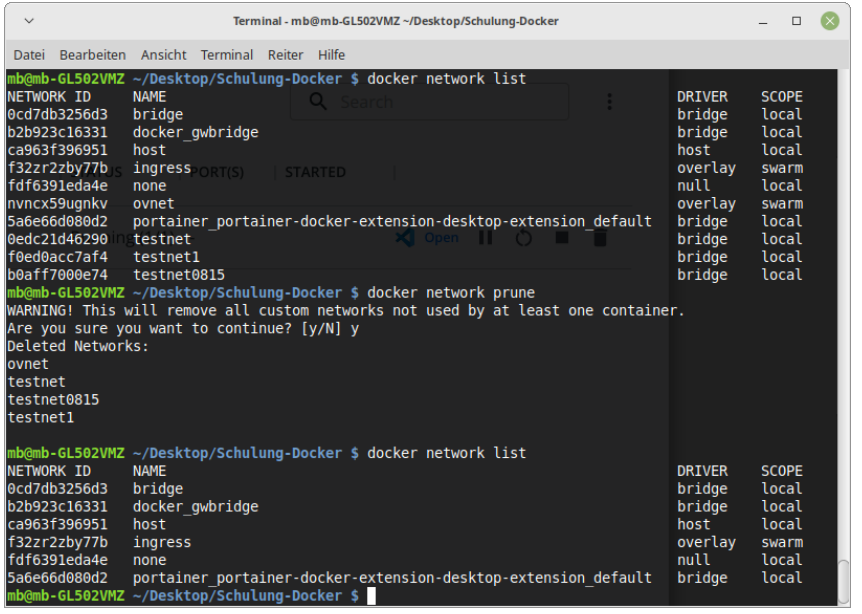


Docker CLI – *network* Kommando

- Netzwerk aufräumen

```
$ docker network prune
```

Es können nur Netzwerke entfernt werden, mit denen keine Container mehr verbunden sind.



Docker CLI – *network* Kommando

- Container mit Netzwerk verbinden/lösen

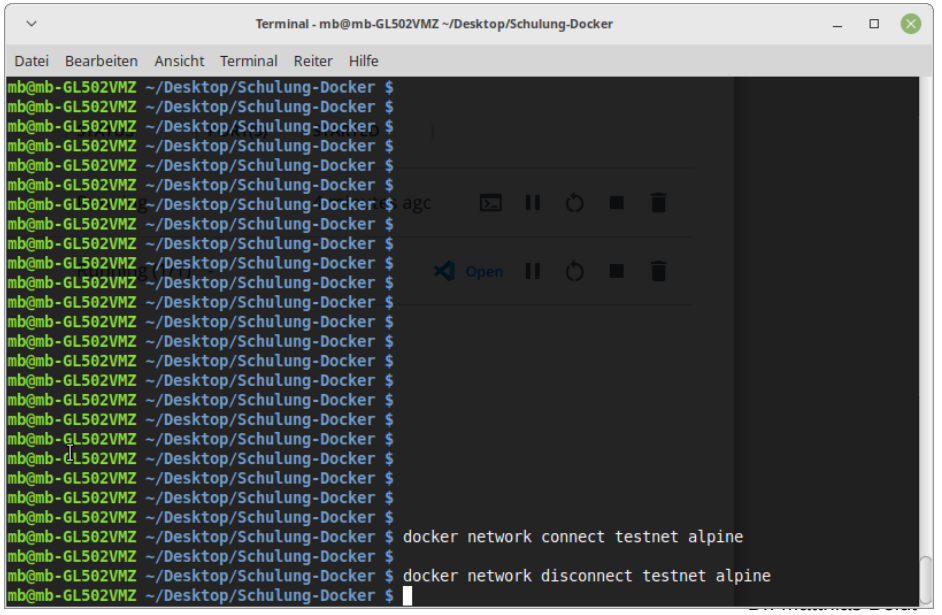
```
$ docker network connect <network-name> <container-name>
$ docker network disconnect

$ docker network create testnet
$ docker container run --name alpine --rm -i -t -d alpine sh -c "while true; do ls; sleep 3; done"
$ docker network connect testnet alpine
$ docker network disconnect testnet alpine
```

Netzwerke können nachträglich mit laufenden Containern verbunden werden und/oder von diesen gelöst werden.

Container beim Start verbinden:

```
$ docker container run –network=testnet alpine
```



Docker – Overlay-Netzwerk

- Nutzung mit Docker-Swarm
  - `docker swarm init` auf Master
  - `docker swarm join` auf anderen Docker Servern
  - `docker network create --attachable --driver overlay ...` auf Master
- Nutzung der Overlay-Netzwerke mit den Containern/Services der Docker-Server im Swarm
- https://docs.docker.com/network/network-tutorial-overlay/

Docker – Logging

- „12 Factors“ => XI. Logs
  - Anwendung/Service kümmert sich nicht darum
  - Strom der Ereignisse wird auf *stdout* und *stderr* geschrieben
  - Laufzeitumgebung kümmert sich um das Sammeln
- `docker logs <container>` liest von stdout und stderr
- Logging-Driver können das Verhalten ändern
- Sammlung durch verschiedene Log-/Event-Collectors möglich (Fluentd, Splunk, …)
- https://docs.docker.com/config/containers/logging/local/

Docker – Container Registry

- Kümmert sich um die Verwaltung von Images
- Private & public Repositories sind möglich
- Versionierung / Speicherung mit Tags
- Anmeldung mit

```
docker login -p <passwd> -u <user> <server>
```

- Abmeldung mit

```
docker logout <server>
```

Docker – Registry Möglichkeiten

- Docker Hub – Angebot von Docker Inc.
- Docker Registry … self hosted … open source
  https://docs.docker.com/registry/
- Kommerzielle Angebote:
  - GitLab, DigitalOcean, AWS, GCP, Azure, ...
  - Canister ... https://canister.io/
- Harbor … https://goharbor.io/
  - Open source, mit UI
  - Für self hosting
  - Aus dem CNCF-Stack

Docker – Registry Möglichkeiten

- Docker Hub – Angebot von Docker Inc.
- Docker Registry … self hosted … open source
  https://docs.docker.com/registry/
- Kommerzielle Angebote:
  - GitLab, DigitalOcean, AWS, GCP, Azure, ...
  - Canister ... https://canister.io/
- Harbor … https://goharbor.io/
  - Open source, mit UI
  - Für self hosting
  - Aus dem CNCF-Stack

Docker – Docker Registry

- Installation als Docker-Container

```
$ docker network create registry
$ docker container run -d -p 5000:5000 --network registry --restart=always --name registry registry:2
```

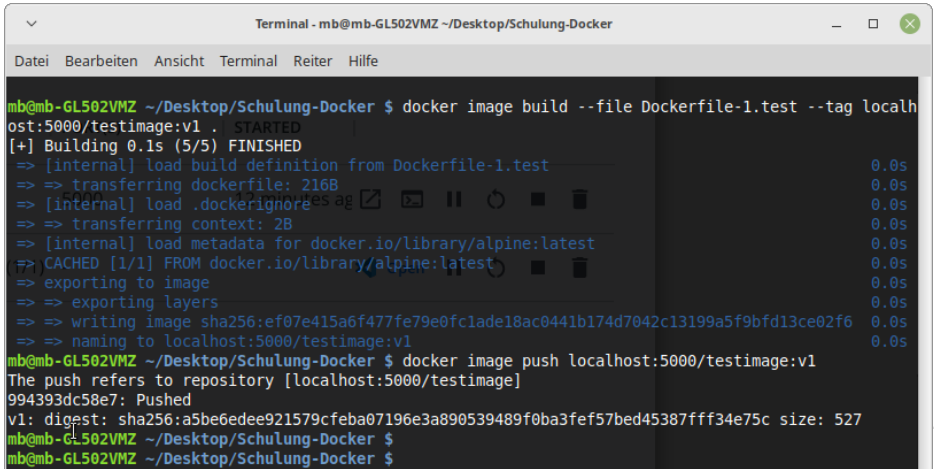
- Ein Image bauen und in die Registry senden

```
$ docker image build --file Dockerfile-1.test --tag localhost:5000/testimage:v1 .
$ docker image push localhost:5000/testimage:v1
```

Aufbau einer eigenen, einfachen Registry.

Eigenes Netzwerk für Nutzung von DNS.

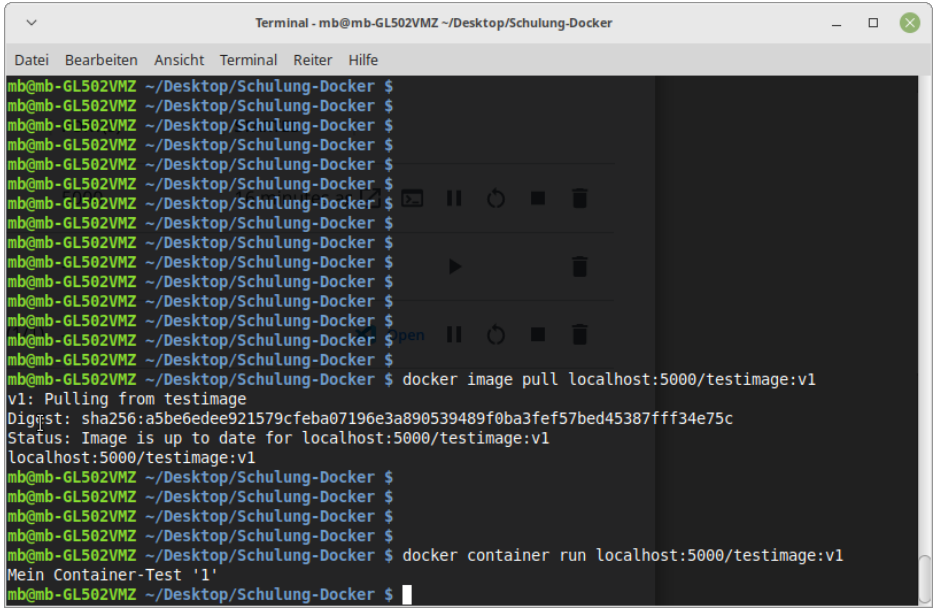
Kommandos in Datei „Registry.sh“



Docker – Docker Registry

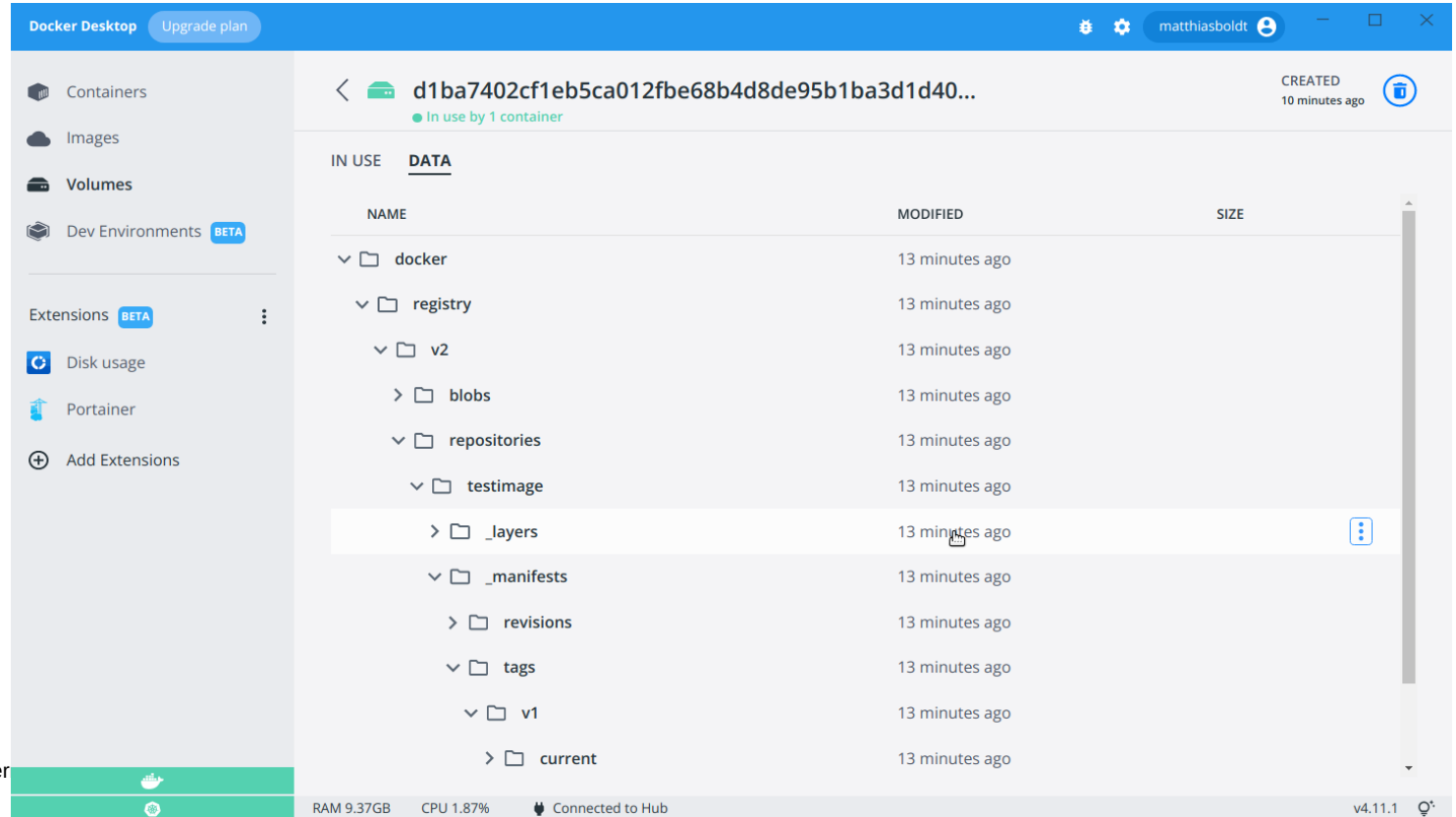
- Image aus der Registry laden

```
$ docker image pull localhost:5000/testimage:v1
$ docker container run localhost:5000/testimage:v1
```



Docker – Docker Registry

- Daten werden in einem Volume gehalten



Docker – Docker Registry

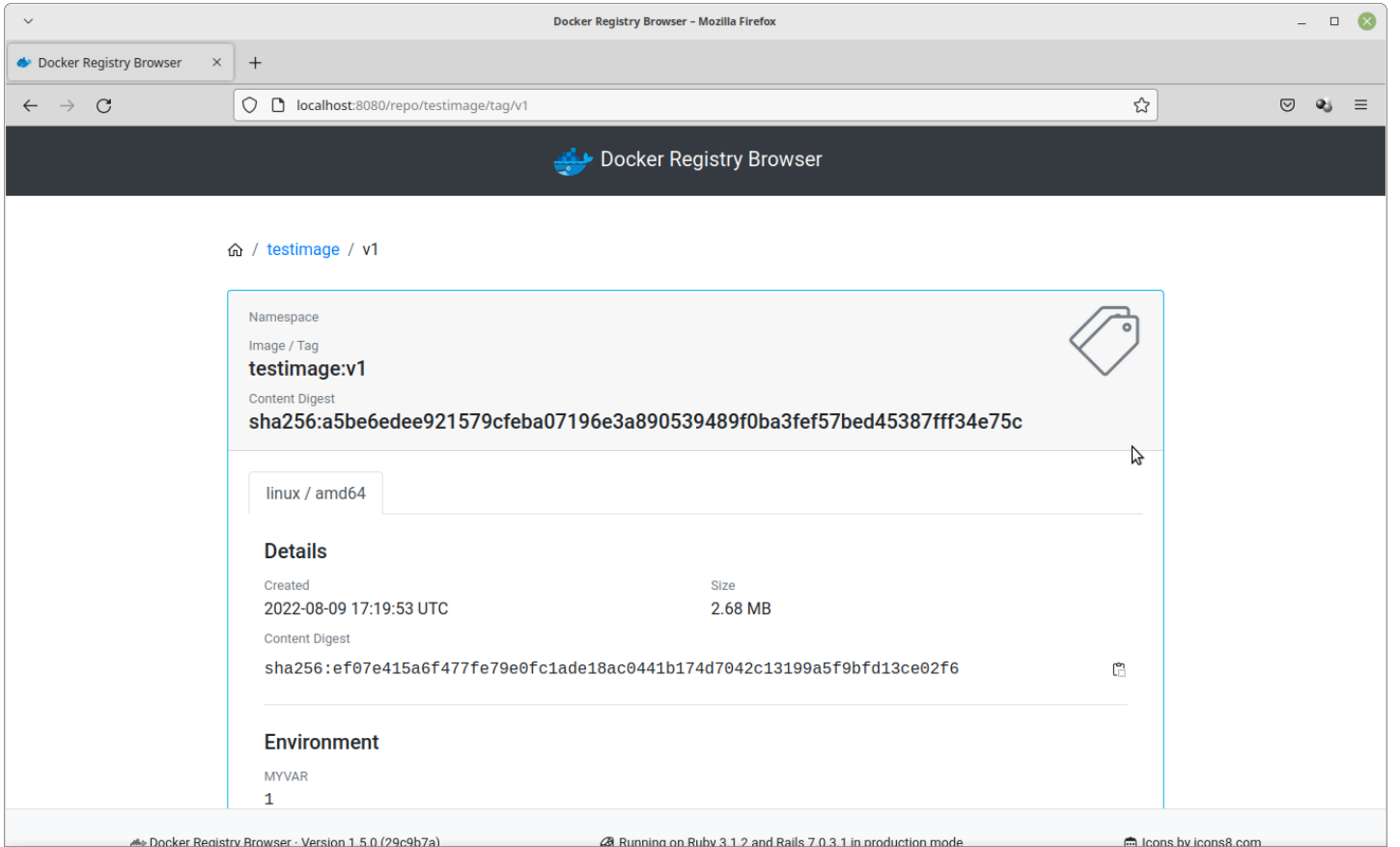
- WEB-Viewer für die eigene Docker Registry

```
$ docker container run -d --name registry-browser --network registry --env
DOCKER_REGISTRY_URL=http://registry:5000 -p 8080:8080 klausmeyer/docker-registry-browser
```

  - Open-Source-Projekt von Klaus Meyer
  - https://github.com/klausmeyer/docker-registry-browser
  - https://github.com/klausmeyer/docker-registry-browser/blob/master/docs/README.md
  - Der Viewer ist über http://localhost:8080 zu erreichen

Docker – Docker Registry

Ansicht der Information zum erzeugten Image im Browser über http://localhost:8080



Docker – Shell-Login

- Shell-login in einen laufenden Linux-Container
- Für Debugging, Tests und Inspektion
- Nutzung von „docker container exec …“

```
$ docker container exec -it <container name> <shell>
$ docker container exec -it alpine /bin/sh
```

- Welche Shell gestartet wird, hängt vom Basis-System ab

Docker – Portainer

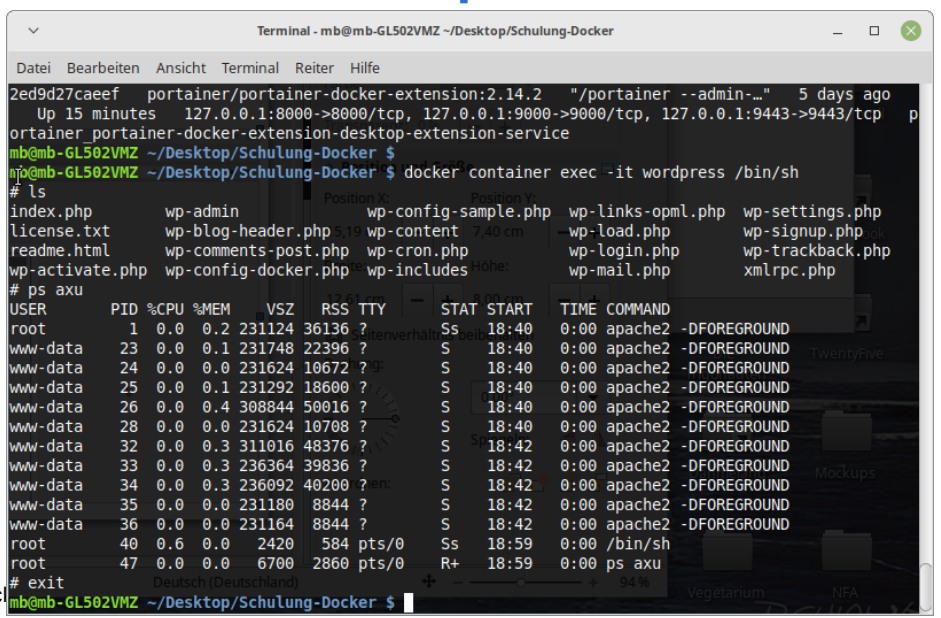
- Nahezu alle Docker-CLI Kommandos aus UI erreichbar
- Komplette Übersicht im Gegensatz zum UI von Docker Desktop
- Als Extension in Docker Desktop installierbar
- Auf Linux Server unter Docker Engine separat installierbar
- In Datei „CMD-Docker-auf-Linux-Server.txt“ ist die Installation beschrieben
- … sehen wir uns Portainer einmal an …
  - Docker Objekte verwalten
  - Image aus TAR-Archiv importieren

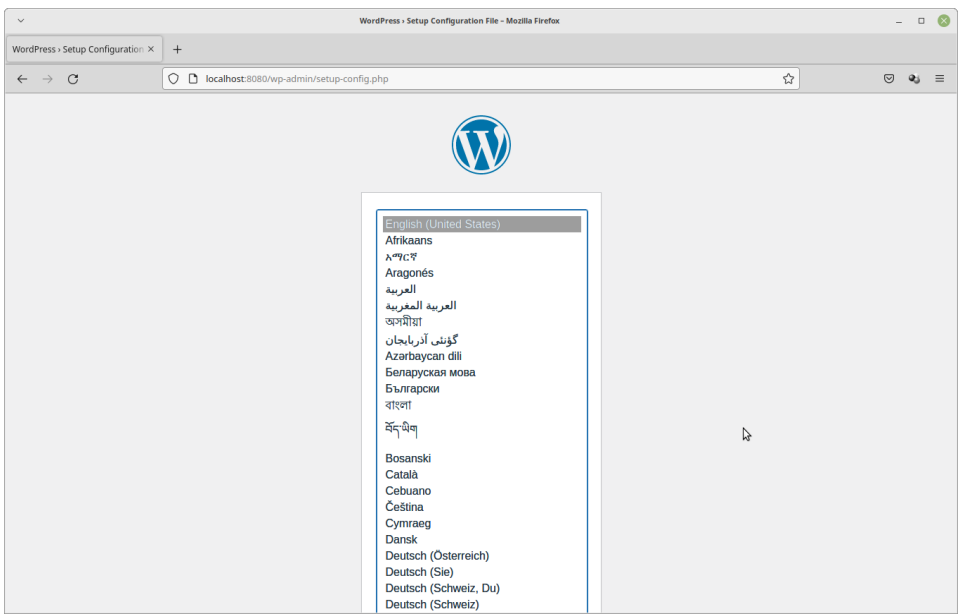
Docker – einfache Beispiele

- Wordpress

```
$ docker network create wordpress
$ docker container run --name wordpress --network wordpress -p 8080:80 -d wordpress
```

- Ist über http://localhost:8080 erreichbar

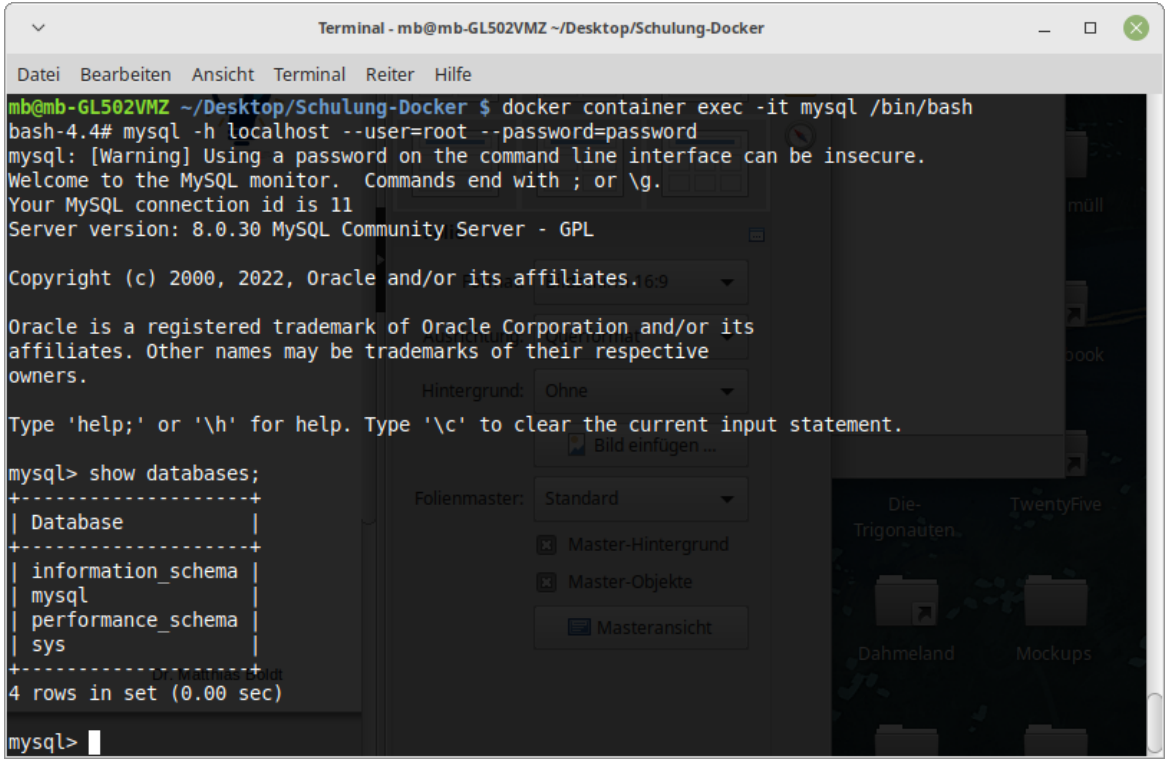




Docker – einfache Beispiele

- MySQL-DB-Server

```
$ docker container run --name mysql --network wordpress -e MYSQL_ROOT_PASSWORD=mypwd -d mysql:latest
```



Docker – einfache Beispiele

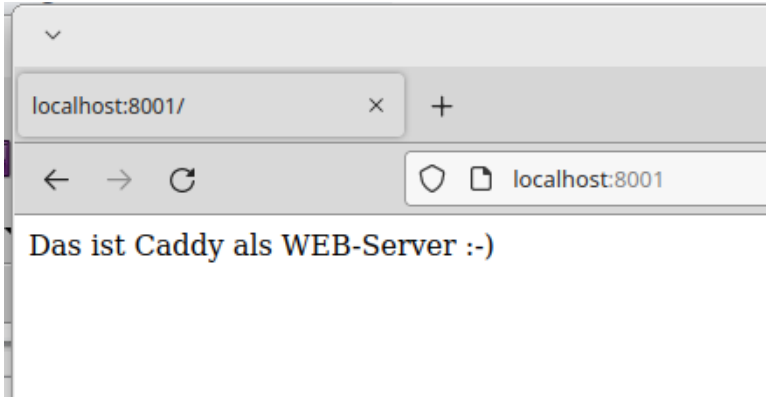
- Wordpress & MySQL-DB-Server verbinden
  - Beide Container im gleichen bridge Netzwerk
  - In Wordpress-Konfiguration den MySQL-Server angeben

Docker – einfache Beispiele

- WEB-Server mit Caddy

```
$ echo "Das ist Caddy als WEB-Server :-)" > index.html
$ docker run -d -p 8001:80 -v $PWD/index.html:/usr/share/caddy/index.html -v caddy_data:/data caddy
```

  - Erreichbar über http://localhost:8001
  - Kann sehr einfach als reverse proxy genutzt werden



Docker - Sicherheit

- Mögliche Angriffsvektoren
  - Betriebssystem
  - Netzwerk
  - Docker Daemon
  - Container
  - Prozesse in Containern
  - ...

Docker - Sicherheit

- Container nutzen das Host-OS (Kernel, Betriebssystem, Netzwerk, ...)
  - Kernel-Security-Features
  - Firewall
- Docker Daemon
  - REST-API … Zugriffsbeschränkung
- Images
  - signed/trusted images
  - docker scan
- Container
  - Zugriffsbeschränkungen für Prozesse
  - Volumes sichern
  - Netzwerke mit minimaler Sichtbarkeit konfigurieren

Docker Compose

- Erweiterung für Docker
- Verwaltung von Docker-Objekten für vollständige Anwendungen (z.B. mit mehreren Containern)
  - Container
  - Netzwerke
  - Volumes
  - ...
- YAML-Files für die Konfiguration
- CLI-Programm „docker-compose“ mit vielen Kommandos *oder*
- Kommando „compose“ im Docker Daemon

Docker Compose

- Nutzung in drei wesentlichen Schritten:
  - Festlegung der Umgebung der Anwendung in Dockerfiles
  - Konfiguration der Services der Anwendung in docker-compose.yml
  - `docker compose up` zum Start der Anwendung (oder `docker-compose up` mit dem extra Binary)

… Bildquellen

- RedHat
- IDG
- Docker
- Earthly
- Aquasec
- dev.to