Docker

Doku: https://docs.docker.com/

Basisinstallation

- https://docs.docker.com/engine/installation/linux/suse/
- Installation auf Ubuntuhttps://docs.docker.com/engine/installation/linux/ubuntulinux/
- Docker UI: http://linoxide.com/linux-how-to/setup-dockerui-web-interface-docker/

Docker Daten-Ablage verlagern

```
root@docker3:/etc/docker# less daemon.json

{
  "graph":  "/mnt/data/docker"
}
```

Images

Images aus Repository runterladen

`docker pull`

Alle Images im lokalen repository zeigen

```
root@develd:~# docker image ls
REPOSITORY                  TAG                 IMAGE ID       CREATED
SIZE
local_discourse/web_only    latest              21e8a905ef5c   4 weeks ago
2.65GB
grundic/jamulus             latest              9e97d3890ca8   5 weeks ago
90.3MB
discourse/base              2.0.20210217-2235   7027ba787aa6   2 months ago
2.22GB
discourse/base              2.0.20201221-2020   c0704d4ce2b4   4 months ago
2.11GB
local_discourse/data        latest              c7524a566464   5 months ago
2.44GB
discourse/base              2.0.20201004-2310   b64c37d7ab06   6 months ago
2.4GB
xbrowsersync/api            latest              a3554c99cc99   12 months ago
119MB
```

Image löschen

```
docker image rm [id]
```

Image Repo aufräumen und ungenutzte Images löschen

```
docker image prune [OPTIONS]
```

z.B: alle images löschen, die nicht von mindestens einem Container genutzt werden

```
docker image prune -a
```

Container Management

Alle Container auf einmal stoppen

```
    docker kill $(docker ps -q)
    docker rm $(docker ps -a -q)
    docker rmi $(docker images -q)
```

Container aus heruntergeladenem Image erzeugen

`docker create` erzeugt den Container und startet ihn **nicht,** `docker run` macht beides.

Alle laufenden und gestoppten Container zeigen:

`docker ps -a`

Filtern auf laufende Container

```
root@develd:~# docker ps -a -f status=running
CONTAINER ID   IMAGE                      COMMAND        CREATED
STATUS        PORTS                  NAMES
31f9a6ffcab8   grundic/jamulus            "Jamulus"      19 hours ago   Up
19 hours                          jolly_beaver
7c3cd1abf744   local_discourse/web_only   "/sbin/boot"   4 weeks ago    Up 5
days     127.0.0.1:84->80/tcp   web_only
05f7f43d0493   local_discourse/data       "/sbin/boot"   5 months ago   Up 5
days                             data
```

Laufende und gestoppte, aber nur die container id ausgeben

`docker ps -aq`

Container zeigen, die nicht laufen

```
root@develd:~# docker ps -aq -f status=exited
b39916cf84e2
e6e7c809ad34
```

Container starten/stoppen

`docker start [ID] docker stop [ID]`

Alle Container stoppen oder löschen

```
docker stop $ (docker ps -a -q) docker rm $(docker ps -a -q)
```

Diese können mit docker rm gelöscht werden

```
root@develd:~# docker rm e6e7c809ad34
e6e7c809ad34
```

Alle Container löschen, die nicht laufen

```
docker container prune
```

Restart policy ändern

```
docker update --restart=no matrix_synapse_1
```

Einen Container betreten

```
docker exec -t -i container_ID /bin/bash
```

Alternativ: stdin/out an den Container hängen:

```
docker attach [OPTIONS] CONTAINER
```

Achtung: wieder raus mit **CTRL-p CTRL-q.** Details siehe https://docs.docker.com/engine/reference/commandline/attach/

Docker Compose V2

Docker Compose erzeugt Docker Applikationen, die aus mehreren Containern bestehen. docker-compose up startet alles im Verbund.

1. Das Dockerfile definiert die Laufzeit-Umgebung
2. docker-compose.yml beschreibt die Services, die in Containern zusammen arbeiten
3. "docker compose up" erzeugt und startet die gesamte Applikation

Achtung: Die python basierten Docker Versionen (V1) in den Distro-Repos sind meist veraltet. Es empfiehlt sich, die V2 aus https://github.com/docker/compose zu benutzen, die in GOLANG neu geschrieben wurde.

Installation siehe https://github.com/docker/compose/

Um docker-compose files im Format V1 auszuführen, gibt es "compose switch": https://github.com/docker/compose-switch. Dafür muss das golang binary in /usr/local/lib/docker/cli-plugins/docker-compose vorhanden sein. composer switch bindet composer V2 so ein, dass man mit

update-alternatives zwischen V1 (python) und V2 (golang) wechseln kann.

```
root@docker2:/usr/local/lib/docker/cli-plugins# curl -fL
https://raw.githubusercontent.com/docker/compose-switch/master/install_on_li
nux.sh | sh
  % Total    % Received % Xferd  Average Speed   Time    Time     Time
Current
                                 Dload  Upload   Total   Spent    Left
Speed
100  1410  100  1410    0     0   7230      0 --:--:-- --:--:-- --:--:--
7230
  % Total    % Received % Xferd  Average Speed   Time    Time     Time
Current
                                 Dload  Upload   Total   Spent    Left
Speed
  0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--
0
100 2884k  100 2884k    0     0  5291k      0 --:--:-- --:--:-- --:--:--
5291k
/usr/bin/update-alternatives
Configuring docker-compose alternatives
update-alternatives: /usr/local/bin/docker-compose-v1 wird verwendet, um
/usr/local/bin/docker-compose (docker-compose) im automatischen Modus
bereitzustellen
update-alternatives: /usr/local/bin/compose-switch wird verwendet, um
/usr/local/bin/docker-compose (docker-compose) im automatischen Modus
bereitzustellen
'docker-compose' is now set to run Compose V2
use 'update-alternatives --config docker-compose' if you want to switch back
to Compose V1
```

Test

```
root@docker2:/usr/local/lib/docker/cli-plugins# update-alternatives --
display docker-compose
docker-compose - automatischer Modus
  beste Version des Links ist /usr/local/bin/compose-switch
  Link verweist zur Zeit auf /usr/local/bin/compose-switch
  Link docker-compose ist /usr/local/bin/docker-compose
/usr/local/bin/compose-switch - Priorität 99
/usr/local/bin/docker-compose-v1 - Priorität 1
```

Update per docker-compose

Update der Images, die im compose file referenziert sind

```
docker-compose pull
```

Daraus die Container neu bauen und startenb

```
docker-compose up --build
```

```
docker-compose up --force-recreate --build -d
```

Docker Netzwerke

Standardmässig werden drei Netze bridge, host, none angelegt. Alle anderen sind custom Networks, die z.B: über compose angelegt wurden:

```
root@docker1:~# docker network ls
NETWORK ID     NAME           DRIVER    SCOPE
54e670dd998d   bridge         bridge    local
5b1cd745d5a3   docker_back    bridge    local
b56f108784b3   docker_dbnet   bridge    local
f3619965eb7b   docker_front   bridge    local
c91196bf89eb   host           host      local
b4f69adafbb3   none           null      local
```

Container werden an die bridge "docker0" auf dem Host gebunden, solange nicht beim docker create ein anderes Netzwerk gewählt wurde (`docker create –network=<NETWORK>`). Mit

```
docker network inspect bridge
```

sieht man den Zustand eines Docker networks

Custom networks

docker network create erzeugt ein eigenes Netzwerk:

```
docker network create –subnet 192.168.82.0/24 –driver bridge bridge2
```

```
locutus:/home/thommie # docker network inspect bridge2 [ { "Name": "bridge2", "Id": "9c353bcf0c2c6ccee0b821e1ff4d1740a074bdea94e93959c522d46a4e6fde8e", "Scope": "local", "Driver": "bridge", "EnableIPv6": false, "IPAM": { "Driver": "default", "Options": {}, "Config": [ { "Subnet": "192.168.82.0/24" } ] }, "Internal": false, "Containers": {}, "Options": {}, "Labels": {} } ]
```

Mit

```
docker attach container1
```

sieht man das Netzwerk von innen

Docker logs

Analog zu tail -f:

```
docker logs --follow
```

Docker volumes

https://docs.docker.com/engine/admin/volumes/volumes/

Kubernetes

- Auf Ubuntu: http://thedevopsblog.com/containers/kubernetes-1-4-setup-in-ubuntu-16-04/
- Offizielle Tutorials: https://kubernetes.io/docs/tutorials/kubernetes-basics/
- weitere: https://marc.wäckerlin.ch/computer/kubernetes-on-ubuntu-16-04

Begrifflichkeiten

- Master = koordiniert den Cluster über die Kubernetes API - auf dem Master laufen keine Pods
- Node = Maschine, auf der Cluster (Pod) laufen (kann eine oder mehrere phys. Maschine oder VMs sein)
- Pod = einer oder mehrere Container, die gemeinsam Ressourcen nutzen (z.B. gemeinsamer Speicherplatz, gemeinsame IP Adresse, Informationen, wie der Container zu betreiben ist). Pod = Container + gemeinsame Ressourcen (Speicher, RAM, CPU, Netzwerk usw.)
- Service: Funktion, die von einem oder mehreren Pods bereitgestellt wird

Minikube - zum Üben

Minikube is a tool that makes it easy to run Kubernetes locally. Minikube runs a single-node Kubernetes cluster inside a VM on your laptop for users looking to try out Kubernetes or develop with it day-to-day.

https://github.com/kubernetes/minikube

```
curl -Lo minikube
https://storage.googleapis.com/minikube/releases/latest/minikube-linux-amd64
chmod +x minikube mv minikube /usr/local/bin/
```

Linux CI Installation Which Supports Running in a VM (example w/ kubectl installation)

```
curl -Lo minikube
https://storage.googleapis.com/minikube/releases/latest/minikube-linux-amd64
chmod +x minikube
```

dann

```
curl -Lo kubectl https://storage.googleapis.com/kubernetes-release/release/
$( curl -s
https://storage.googleapis.com/kubernetes-release/release/stable.txt )
/bin/linux/amd64/kubectl chmod +x kubectl
```

und

```
export MINIKUBE_WANTUPDATENOTIFICATION=false export
MINIKUBE_WANTREPORTERRORPROMPT=false export MINIKUBE_HOME= $HOME export
CHANGE_MINIKUBE_NONE_USER=true mkdir $HOME /.kube || true touch $HOME
/.kube/config export KUBECONFIG= $HOME /.kube/config sudo -E ./minikube start
—vm-driver=none # this for loop waits until kubectl can access the api server
that minikube has created for i in {1..150} # timeout for 5 minutes do
./kubectl get po &> /dev/null if [ $? -ne 1 ] ; then break fi sleep 2 done
```

Minikube mit node.js hello world applikation:

https://kubernetes.io/docs/tutorials/stateless-application/hello-minikube/