Docker

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

Basisinstallation

- Doku zu SUSE bei Docker: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/
- Opensuse Library auf Docker Hub: https://hub.docker.com/_/opensuse/

Standard Befehle

https://www.linux.com/blog/learn/chapter/Containers-Devs/2017/6/basic-commands-performing-docker-container-operations

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]
```

Container

Container aus heruntergeladenem Image erzeugen

`docker create`

Details siehe https://docs.docker.com/engine/reference/commandline/create/

`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
```

Diese können mit docker rm gelöscht werden

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

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

Docker Compose erzeugt Docker Applikationen, die aus mehreren Containern bestehen. Das Dockerfile definiert die Betriebsumgebung. *docker-compose.yml* definiert die Services und Container, die Teil der Applikation sind. docker-compose up startet alles im Verbund.

Docker Netzwerk

Standardmässig werden drei Netze angelegt:

```
root@nas:~# docker network ls NETWORK ID NAME DRIVER SCOPE c14b1455af82
bridge bridge local d80bf7ff4673 host host local 269bb4261727 none null
local cd1ba9150e2f owncloud_dockerserver_default bridge local
```

Container werden an die bridge (entspricht 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 der Docker Bridge

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 volumes

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

Kodi headless

Image vom Docker Hub holen

docker pull linuxserver/kodi-headless

Container aus Image erzeugen

- `p 8080` - webui port
- `-p 9777/udp` - esall interface port
- `-v /config/.kodi` - path for kodi configuration files
- `-e PGID` GroupID docker group
- `-e PUID` UserID docker user
- `-e TZ` - for timezone information *eg Europe/London, etc*
- –restart="always" - Restart policy

```
docker create —-name=kodi-headless -v /var/lib/kodi/.config/.kodi
—restart=always -v /var/lib/kodi/.kodi -e PGID=474 -e PUID=1001 -e
TZ=Europe/Berlin -p 8080:8080 -p 9777:9777/udp linuxserver/kodi-headless
```

Kodi Konfiguration

http://kodi.wiki/view/Advancedsettings.xml

Kodi mit mariadb Container

https://github.com/milaq/kodi-headless

http://kodi.wiki/view/MySQL/Setting_up_Kodi

https://mariadb.com/kb/en/mariadb/installing-and-using-mariadb-via-docker/#using-mariadb-images

```
docker run –name mariadb –restart=always -e MYSQL_ROOT_PASSWORD=root mariadb
```

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/