AUFGABENBLATT 1

Aufgabe 1.1

Docker installieren: sudo apt-get install docker.io

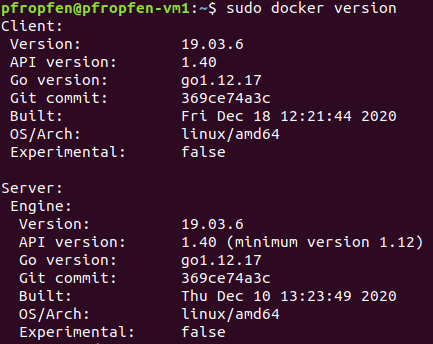
Aufgabe 1.2

1. Prüfen ob Docker installiert wurde:

sudo docker version (gibt nichts zurück wenn docker nicht installiert ist)

1. Welche Version ist installiert?

sudo docker version (19.03.6)

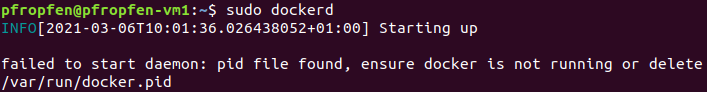


oder: sudo docker –-version (Einzeiler)



1. Feststellen ob Docker Daemon auf dem Host läuft:

dockerd starten: sudo dockerd



Oder nach Prozess suchen:

ps -e | grep dockerd

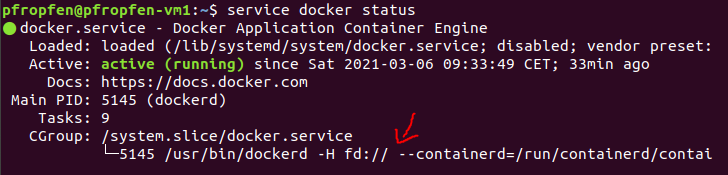


service docker status

(oder systemctl status docker)

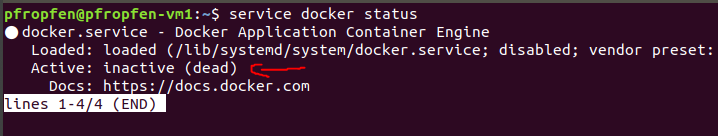
1. Parameter, mit denen Dockerd gestartet wurde:

service docker status



Parameter: -H fd:// --containerd=/run/containerd/containerd.sock

1. Läuft der Docker Daemon nach Neustart?



Nein, man sollte aber meinen, dass der Dienst beim Start auch ausgeführt wird. Möglicherweise wird er nach dem Start direkt wieder beendet wenn es nichts zu tun gibt. Existiert z.B. ein Docker Service mit „restart: always“ würde er laufen.

Aufgabe 1.3

Container auf Basis des Hello-World Images starten:

a)

sudo docker run -d --name=hello-world-1 hello-world

b)

sudo docker run -d --name=hello-world-2 hello-world

c)

sudo docker images (zeigt alle existierenden images an)

sudo docker ps (zeigt laufende container an)

sudo docker ps -a (zeigt alle container an)

Aufgabe 1.4

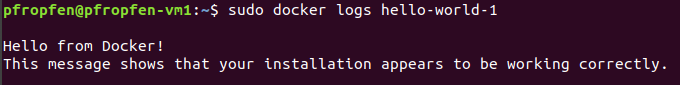
1. Genau Ein Image von Hello-world befindet sich auf dem Host:

sudo docker image ls

2. keine (sudo docker ps), hello-world beendet sich sofort nach Abarbeitung

3. hello-world-1 und hello-world-2 befinden sich auf dem Host: (sudo docker ps -a)

4. sudo docker logs hello-world-1 (zeigt Logging Informationen)



Ausgaben des Containers werden angezeigt.

5. docker run (pullt und buildet)

docker pull -> docker create -> docker start

Aufgabe 1.5

1. Löschen des Containers per ID:

sudo docker ps -a (zeigt IDs)

sudo docker stop 27dd5fad1b8c (falls container läuft)

sudo docker rm 27dd5fad1b8c (löscht nach ID)

2. sudo docker rm hello-world

geht nicht, weil das Image in Benutzung ist (durch existierende Container)

3. alle Container (basierend auf hello-world image) löschen

sudo docker rm hello-world

oder

sudo docker image rm –-force hello-world

Aufgabe 1.6

Container mit Apache Webserver:

1.

docker pull httpd

sudo docker run -d -p 8888:80 -h wpfdocker --name=c-meinapache\_01 httpd

2. sudo docker inspect c-meinapache\_01 | grep wpfdocker (oder grep Hostname)

3.

sudo docker inspect c-meinapache\_01 (gibt WorkDir an: /var/lib/docker/overlay2/)

/var/lib/docker/overlay2/2283a96.... /merged/usr/local/apache2/htdocs/index.html

index.html editieren.

oder

sudo docker container exec -it c-meinapache\_01 /bin/bash

ls

cd htdocs

vim index.html

4.

sudo docker stop c-meinapache\_01

sudo docker start c-meinapache\_01

die editierte Seite wird angezeigt.

AUFGABENBLATT 2

AUFGABE 2.1

sudo docker container run -d --name=c-mysqlserver-1 -e MYSQL\_ROOT\_PASSWORD=test123 mysql

sudo docker container exec -it c-mysqlserver-1 mysql -u root -p

mysql> status

\c

exit



AUFGABE 2.2

sudo docker run --name=c-myadmin-1 -d --link c-mysqlserver-1:db -p 8080:80 phpmyadmin

Mit -p Portangabe um per Browser zugreifen zu können.

(Im Browser unter localhost:8080 aufzurufen mit User=root und pw=test123)

--link verbindet zum eigenen mysql-container

AUFGABE 2.3

Mit gemeinsamen Netzwerk:

sudo docker run -d --network=n-mynet-1 --name=c-mysqlserver-1 -e MYSQL\_ROOT\_PASSWORD=test123 mysql

sudo docker run --network=n-mynet-1 --name=c-myadmin-1 -d -e PMA\_HOST=c-mysqlserver-1 -p 8080:80 phpmyadmin

AUFGABE 2.4

sudo docker inspect c-mysqlserver-1

sudo docker inspect c-myadmin-1

"Links" zeigt die Verbindung zu c-mysqlserver-1

Port, Status, Netwerkzugehörigkeit

Root Passwort

AUFGABE 2.5

Zwei MySQL Instanzen und 1 PhpMyAdmin

sudo docker network create n-mynet-1

sudo docker network create n-mynet-2

sudo docker run -d --network=n-mynet-1 --name=c-mysqlserver-1 -e MYSQL\_ROOT\_PASSWORD=test123 mysql

sudo docker run -d --network=n-mynet-2 --name=c-mysqlserver-2 -e MYSQL\_ROOT\_PASSWORD=test123 mysql

sudo docker run --network=n-mynet-1 --name=c-myadmin-1 -d -e PMA\_HOSTS=c-mysqlserver-1,c-mysqlserver-2 -p 8080:80 phpmyadmin

sudo docker network connect n-mynet-2 c-myadmin-1

AUFGABE 2.6

1.

a)

sudo docker run -p 8081:80 -h apache-vol01 -d --name=c-meinapache\_vol01 -v /usr/local/apache2/htdocs

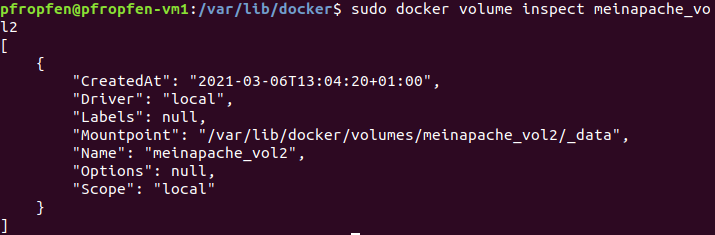
b) Speicherort des Volumes auf dem Host:

/var/lib/docker/volumes/{volume id}

2.

a)

sudo docker run -p 8082:80 -h apache-vol02 -d --name=meinapache\_vol02 -v meinapache\_vol2:/usr/local/apache2/htdocs httpd



b)

( /usr/local/apache2 im Container )

/var/lib/docker/volumes/meinapache\_vol2/\_data auf dem Host

3.

sudo docker run -p 8083:80 -h apache-vol03 -d --name=meinapache\_vol03 -v /opt/apache/docroot:/usr/local/apache2/htdocs httpd

AUFGABENBLATT 3

AUFGABE 3.1

Installation von Docker-Compose:

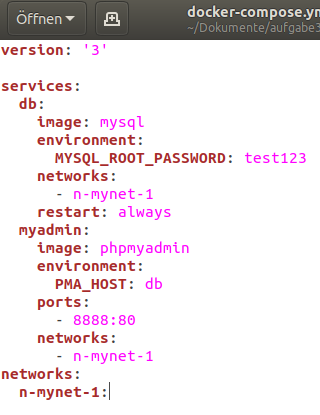
sudo apt-install docker-compose

sudo docker-compose -version



AUFGABE 3.2

YML-DATEI: docker-compose.yml



version: '3' <- Version der Compose-File

services: <- Ab hier Bereich der 'Services'

db: <- Service 'db'

image: mysql <- Name des verwendeten Images für den Service 'db'

environment: <- Abschnitt: Umgebungsvariablen für 'db'

MYSQL\_ROOT\_PASSWORD: test123 <- Umgebungsvariable 'MYSQL\_ROOT\_PASSWORD' mit dem Wert 'test123'

networks: <- Abschnitt: Netzwerkzugehörigkeiten für 'db'

- n-mynet-1 <- Netzwerkzugehörigkeit zu 'n-mynet-1'

restart: always <- Container wird beim hochfahren immer neugestartet es sei denn, er wird bewußt runtergefahren

myadmin: <- Service 'myadmin'

image: phpmyadmin <- Name des verwendeten Images für den Service 'myadmin'

environment: <- Abschnitt: Umgebungsvariablen für 'myadmin'

PMA\_HOST: db <- Umgebungsvariable 'PMA\_HOST' mit dem Wert 'db'

ports: <- Abschnitt: Verwendete Ports

- 8888:80 <- Verwendeter Port: 8888:80 ( Container mit Port 80 wird auf Port 8888 des Host geleitet)

networks: <- Abschnitt: Netzwerkzugehörigkeiten für 'myadmin'

- n-mynet-1 <- Netzwerkzugehörigkeit zu 'n-mynet-1'

networks: <- Ab hier Bereich der Netzwerke

n-mynet-1: <- Netzwerk: 'n-mynet-1' erstellen

-> 2 Dienste erstellt (db, myadmin) und in ein Netzwerk gehangen (n-mynet-1).

Bei MYADMIN wird die zu verwendene Datenbank angegeben.

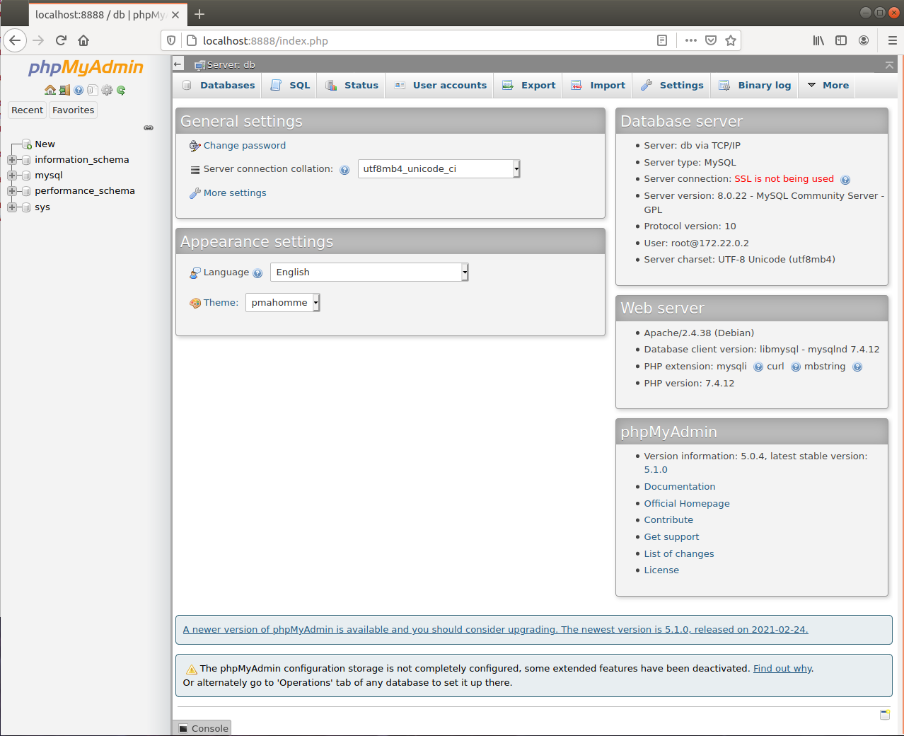
Ausführen der YML im entsprechenden Verzeichnis:

sudo docker-compose up -d

sudo docker-compose down

Über localhost:8888 die phpmyadmin-Seite aufrufen und auf MySQL zugreifen:





AUFGABE 3.3

1. Vorteile von Docker-Compose:

- Mehrere Container und Netzwerke können mit 1 Befehl erstellt/gestartet und gestoppt/gelöscht werden.

- Neuerungen in der yml-Datei anpassen.

- Umgebungsvariablen und 'secrets' können in der yml-Datei gespeichert werden und müssen nicht immer eingegeben werden.

- Netzwerkzugehörigkeiten können direkt vereinbart werden.

2.

die erstellten Container und Netzwerke heißen "aufgabe3-2\_NAME\_1"

aufgabe3-2\_ ist der Name des Verzeichnisses,

NAME ist der Name des Containers/Netzwerks,

\_1 weil es der erste (bzw. n-te) Container dieser Art ist (mit --scale CONTAINERNAME=2 mehrere möglich (REPLIKAS)).

3. In der YML-Datei:

image: xyz

container\_name: NAME

4.

Ab YML-Version 2 kann mit depends on SERVICENAME eine Reihenfolge festgelegt werden.

die Struktur der YML-Datei bestimmt die Reihenfolge der Container.

Benötigt jedoch ein Container zu lange zum Erstellen kann er 'überholt' werden.

Mit "docker-compose config" kann die Kompilierreihenfolge eingesehen werden.

5.

Falls bei der Konfiguration kein Netzwerk angegeben wird, wird

automatisch ein VERZEICHNISNAME\_DEFAULT Netzwerk erstellt und der Container eingehangen.

6.

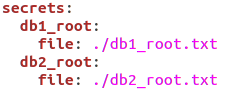
Daten die geschützt sein sollen können mit 'secrets' über den Inhalt einer

Textdatei übergeben werden welche an einem Ort mit eigenen Zugangsrechten liegt.

im Service:

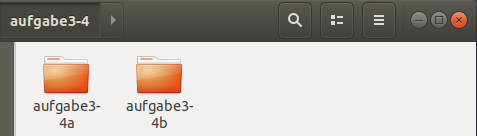


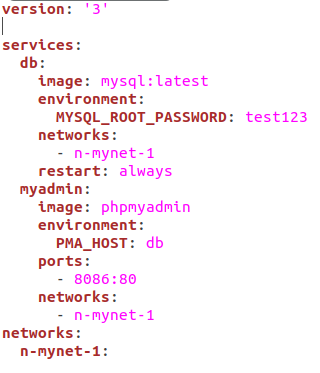
Global:



AUFGABE 3.4

2 VERZEICHNISSE JEWEILS MIT YML-DATEI:





version: '3' <- Version der Compose-File

services: <- Ab hier Bereich der 'Services'

db: <-

image: mysql:5.7 <- :latest bei der anderen yml-DAtei

environment: <-

MYSQL\_ROOT\_PASSWORD: test123 <-

networks: <-

- n-mynet-1 <-

restart: always <-

myadmin: <-

image: phpmyadmin <-

environment: <-

PMA\_HOST: db <-

ports: <-

- 8085:80 <- 8086:80 bei der anderen yml-DAtei

networks: <-

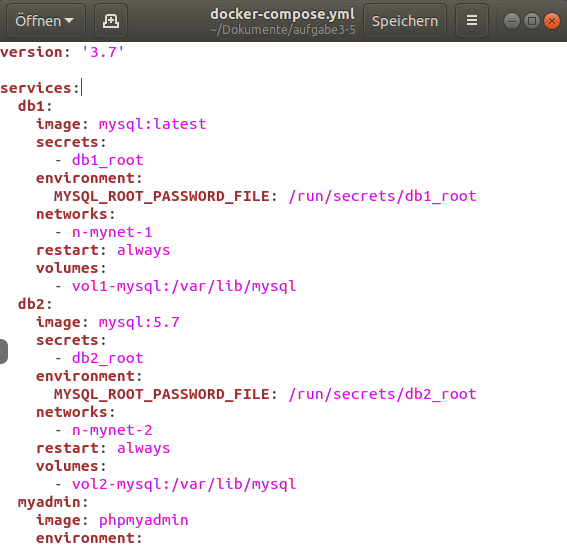
- n-mynet-1 <-

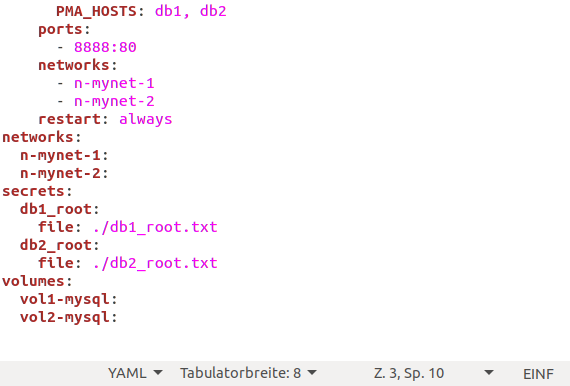
networks: <-

n-mynet-1:

sudo docker-compose up -d <- in beiden Verzeichnissen

AUFGABE 3.5





Zunächst werden die Services db1, db2 und myadmin angelegt.

db1, db2 haben je ein eigenes netz, myadmin ist in beiden.

Für db1 und db2 werden volumes angelegt.

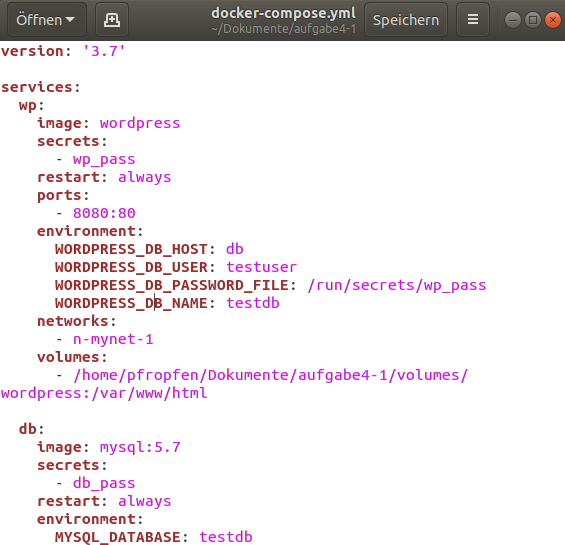
Die Passwörter werden in entsprechenden text-dateien gespeichert

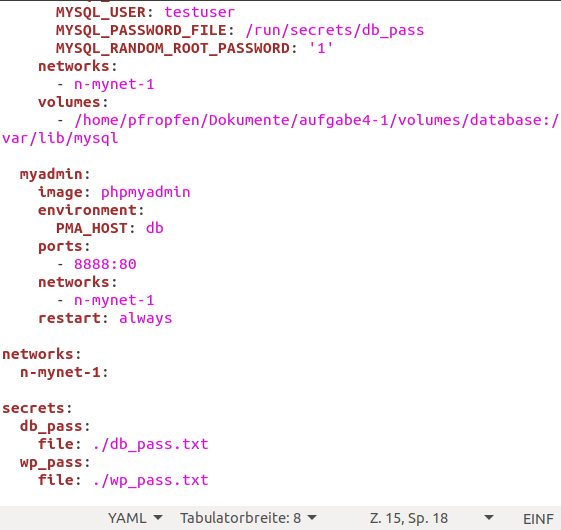
Mit 'restart always' werden die Container automatisch wieder gestartet.

Um das Passwort zu ändern muss das Volume gelöscht werden (down -v).

AUFGABENBLATT 4

AUFGABE 4.1



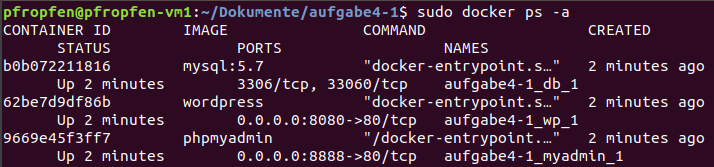


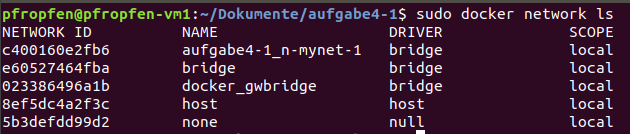
1. sudo docker-compose up -d

(2) localhost:8080 aufrufen und wordpress installieren

1. sudo docker ps -a

sudo docker network ls





Container: Networks:

aufgabe4-1\_wp\_1 aufgabe4-1\_n-mynet-1

aufgabe4-1\_db\_1

aufgabe4-1\_myadmin\_1

(4)

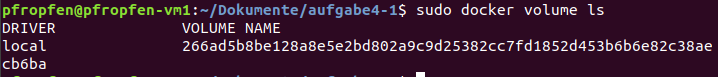
sudo docker-compose down

(5)

sudo docker ps -a

sudo docker network ls

(alle Container und erstelltes Netz weg, persistente daten befindet sich aber trotzdem im Volume)



(6)

compose runterfahren, 'volumes'-Ordner manuel löschen

compose wieder starten (sudo docker-compose up -d)

ö

AUFGABE 4.2

Die Volumes müssen auf "named Volumes" geändert werden!

dadurch werden sie mit "sudo docker-compose down -v" mit gelöscht.

yml-datei ändern:

services:

wp:

volumes:

- vol1-wp:/var/www/html (<- in services, linke seite ändern)

db:

volumes:

- vol2-db:/var/lib/mysql

volumes: (<- auf Top-Level deklarieren)

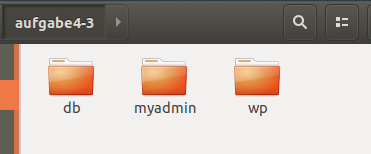
vol1-wp

vol2-db

AUFGABE 4.3

-----------

Je Service 1 Unterordner mit eigener YML-Datei.



Netzwerk je auf Toplevel Ebene benennen:

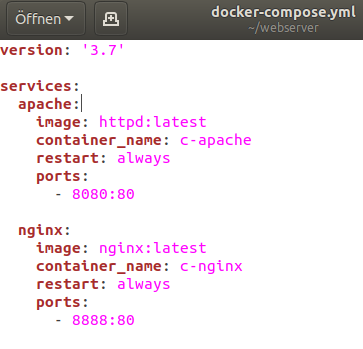
networks:

n-mynet-1:

name: meinnetz

AUFGABE 4.4

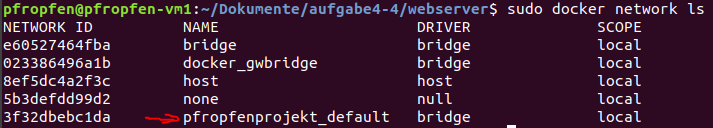
(yml-datei in /home/pfropfen/webserver/)



Projektname selbst festlegen:

sudo docker-compose -p PROJEKTNAME up -d

sudo docker-compose –-projectname PROJEKTNAME up -d



AUFGABE 4.5

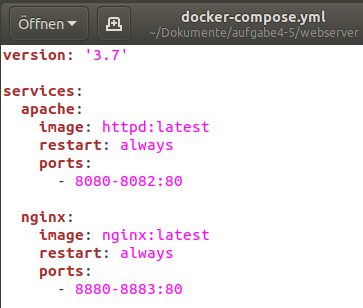
1. mit "--scale dienstname=X" kann ein Dienst mehrfach gestartet werden

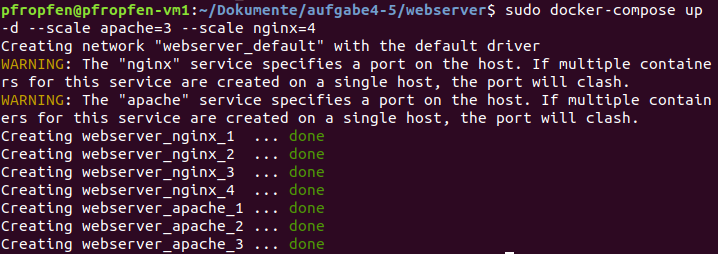
sudo docker-compose up -d --scale apache=3 --scale nginx=4

in der yml-datei müssen je 3 bzw. 4 ports angegeben werden. (- 8080-8082:80 für apache)

(- 8880-8883:80 für nginx)

Keine container\_name damit docker selbst benennen kann.

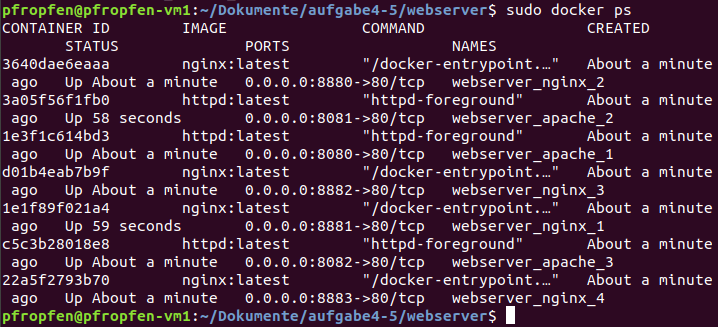




2.

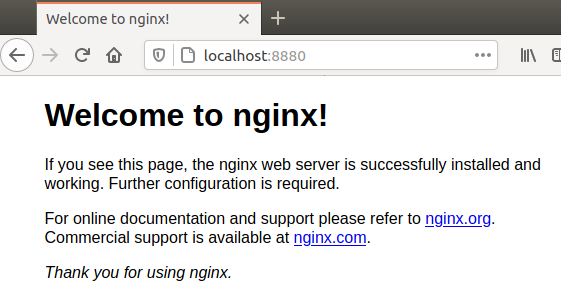
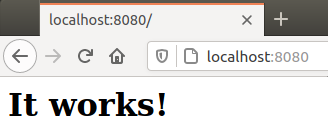
sudo docker ps <- zeigt container die laufen

sudo docker network ls <- zeigt netze die angelegt wurden



Im Browser "localhost:8080" (bzw ..8081,8880,usw) aufrufen. Wird die Seite angezeigt,

hat der Container auf die Anfrage reagiert.



3.

mit "sudo docker-compose down -v" werden alle Container, Netzwerke (und Volumes) gelöscht.

AUFGABENBLATT 5

AUFGABE 5.1

1.

https://hub.docker.com/\_/httpd?tab=tags (<- available tags)



Liste von verfügbaren Images mit Tags und pull-Befehl

(latest, 2.4.46 , 2.4 , 2.4-alpine , ...)

Varianten:

Allgemeine Linux Variante

Variante auf Basis von Alpine Linux

2.

Quick Reference (Problembehandlung, unterstützte Architekturen, ...)

Supportet tags and Dockerfile Links (<- zeigt Ihnhalt der Dockerfiles)

Beschreibung

How to use this Image

Image Varianten

License

https://hub.docker.com/\_/httpd (<- der \_ gibt an dass es sich um ein offizielles Image handelt)

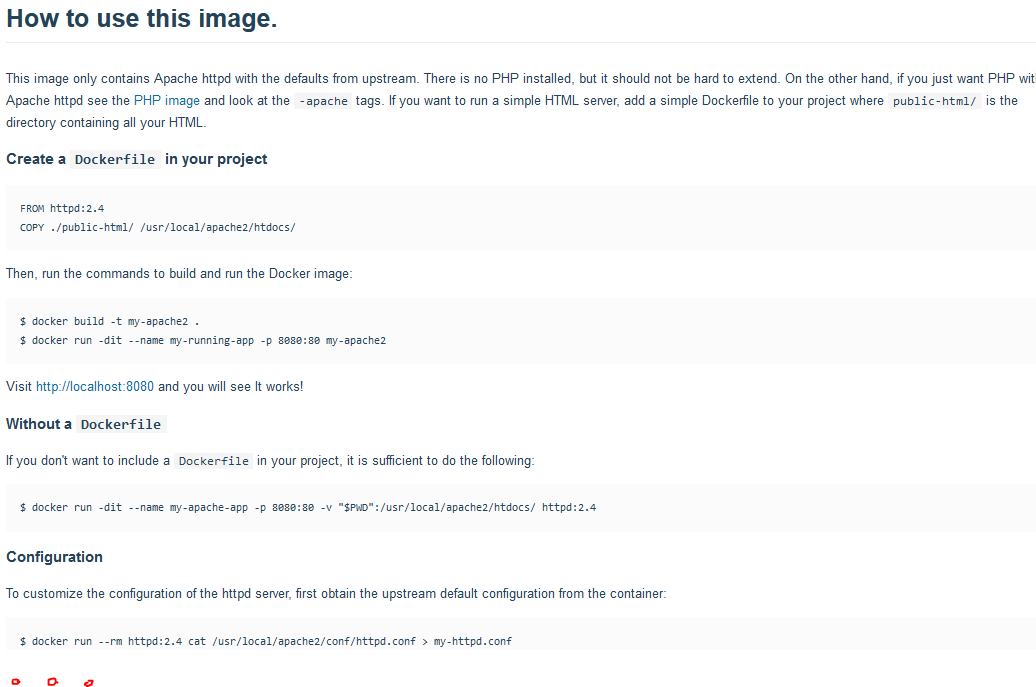
3.

Abschnitt:

Ho to use this Image?

Befehle um den Container mit oder ohne Dockerfile zu erstellen.

Konfiguration durch .conf Datei



4.

Link zu Apache Lizenzinformation.

Hinweis auf wahrscheinliche andere Lizenzen (z.B. Bash)

Link zu Some additional license information

5.

Variante und Version:

normal-variante: 2.4.46 , 2.4 , 2 , latest

alpine-Variante: 2.4.46-alpine , 2.4-alpine , 2-alpine , alpine

(link zeigt den kompletten inhalt der dockerfile)

AUFGABE 5.2

sudo docker image ls (zeigt vorhandene Images)

1.

sudo docker pull httpd:2.4.46

2.

sudo docker pull httpd:2.4.46-alpine

3.

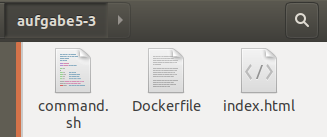
sudo docker image inspect httpd:2.4.46

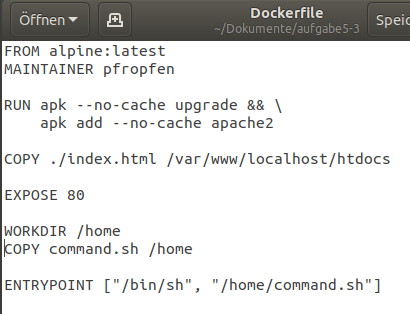
sudo docker image inspect httpd:2.4.46-alpine

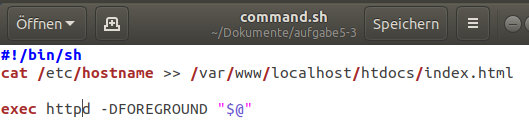
sudo docker image history httpd ... (zeigt Schichten des Images)

AUFGABE 5.3

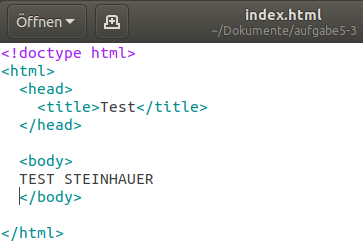
1.

dockerfile, command.sh und index.html:





cat gibt Inhalt aus >> hinten anhängen > überschreiben



2. Image mit Dockerfile erzeugen:

sudo docker build -t apacheimage . (baut das image "." bedeutet es wird die dockerfile aus dem aktuellen Verzeichnis verwendet)

(-t eigenen namen für image angeben)

3. Image klein halten:

Alpine (statt z.B. Ubuntu) als Basisimage verwenden.

--no-cache damit installer-dateien nicht mit gespeichert werden

(Nur die notwendigen updates und services im Image enthalten)

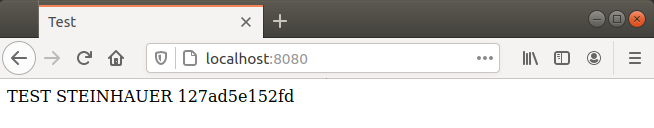
Eine RUN Direktive (mit && verknüpft) statt 2 RUN-Befehle.

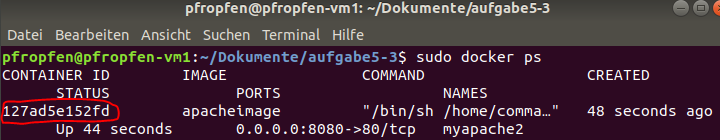
4.

sudo docker run -d -p 8080:80 --name=myapache2 apacheimage

(startet den container aus dem apacheimage)

localhost:8080 aufrufen, gezeigte ID stimmt mit Container-ID (sudo docker ps -a) überein.





AUFGABE 5.4

1. Eigene Registry:

sudo docker run -d -p 5000:5000 --restart always -v regvol --name pfropfreg registry:2

(Standard Pfad für Volume im Container: /var/lib/registry) (VOLUME Direktive in Registry-Dockerfile)

2.

sudo docker tag apacheimage localhost:5000/apacheimage:version1

sudo docker push localhost:5000/apacheimage

sudo docker tag apacheimage localhost:5000/apacheimage:version2

sudo docker push localhost:5000/apacheimage

3.

sudo docker image rm localhost:5000/meinapache:version1 (löscht das Image lokal!)

sudo docker image ls (zeigt die lokalen Images)

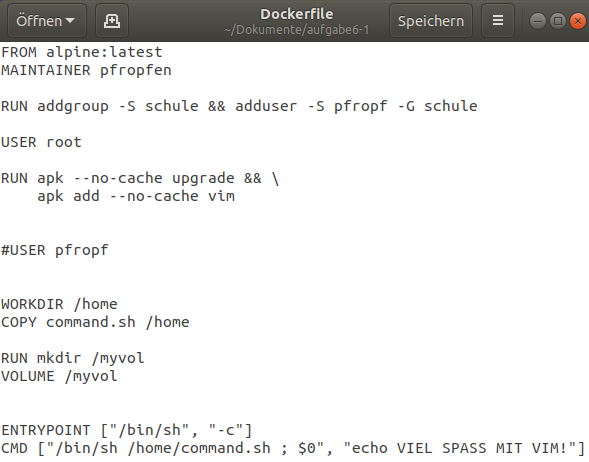
4.

sudo docker run -d -p 8080:80 --name=myapachetest localhost:5000/apacheimage:version1

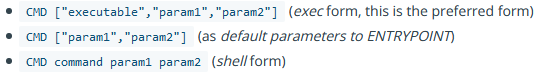
"Unable to find locally" zeigt dass das Image nicht lokal vorhanden ist.

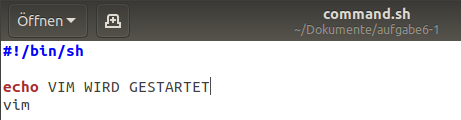
AUFGABENBLATT 6

AUFGABE 6.1



ENTRYPOINT -> CMD , cmd wird bei run container ggf. überschrieben





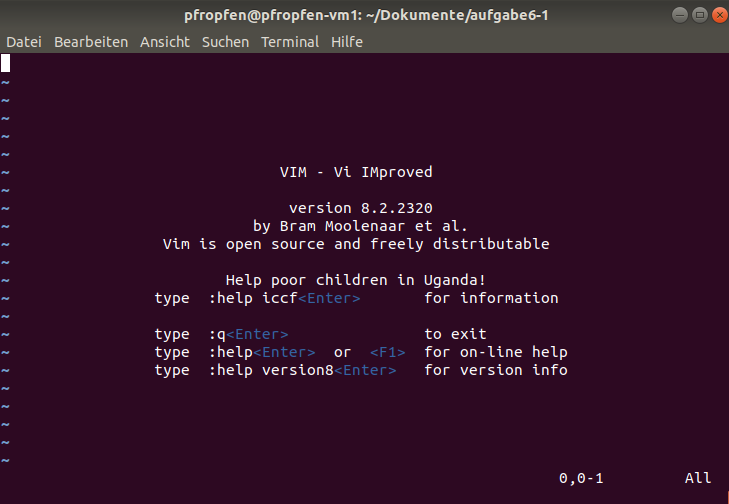
sudo docker build -t vimage . (baut das Image)

sudo docker run --name=vimmeister vimage (startet den container)

sudo docker exec -it vimmeister /bin/sh (inhalt des containers)

MIT VOLUME:

sudo docker run --name=vimmeister -v /home/pfropfen/Dokumente/aufgabe6-1/volumes:/myvol vimage



AUFGABE 6.2

MariaDB-Image

1.

Supported Tags: 5 (3x focal, 2x bionic) jeweils mit verschiedenen versionen (24 Insgesamt)



2.

Images laden, und mit history die Schichten anzeigen lassen.

-> sudo docker image history apacheimage

(oder: per Link Inhalt anzeigen und vergleichen)

3.

5 inhaltlich verschiedene Varianten

Ingesamt namentlich 24 (23 weil latest = mit letzter versionsnummer)

4.

/etc/mysql/my.cnf (im Container) <- Start Konfiguration

/etc/mysql/conf.d <- alle .cnf dateien an diesem Ort ergänzen/überschreiben die settings aus my.cnf

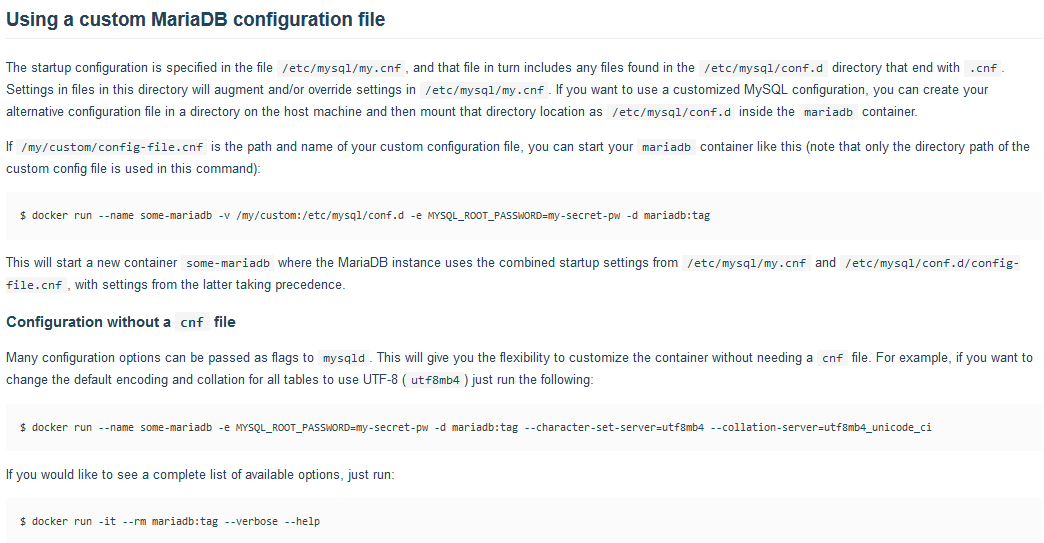
Ohne .cnf-Datei können manche Flags auch bei run befehl mitgegeben werden:

docker run --name some-mariadb -e MYSQL\_ROOT\_PASSWORD=my-secret-pw -d mariadb:tag --character-set-server=utf8mb4 --collation-server=utf8mb4\_unicode\_ci

Alle Möglichkeiten von Flags für ein Image ausgeben:

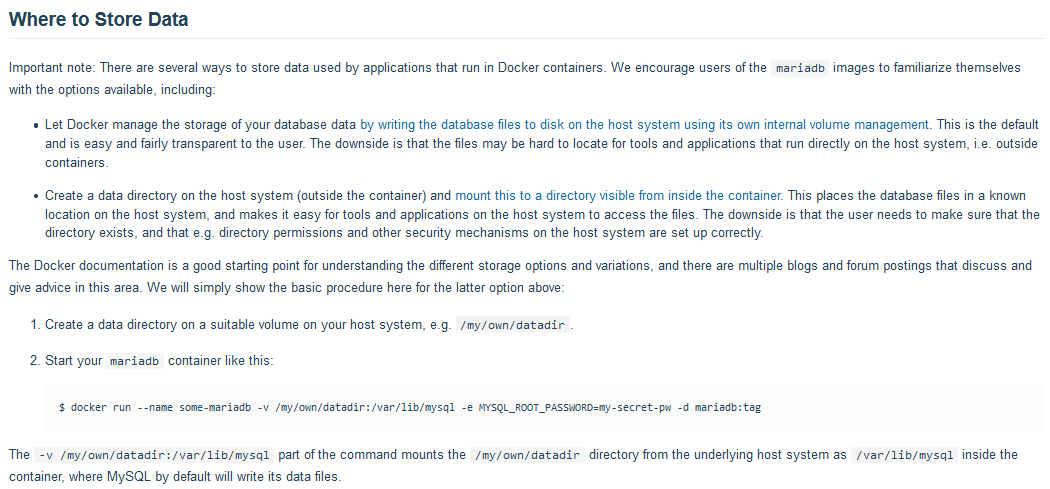
docker run -it --rm mariadb:tag --verbose –help

(--rm löscht den Container bei beenden)



5.

Verwendung von Volumes



6.

Es gibt 7 Umgebungsvariablen:

MYSQL\_ROOT\_PASSWORD legt das root passwort fest

MYSQL\_USER legt einen bestimmten Nutzer fest

MYSQL\_PASSWORD legt Passwort für den Nutzer fest

MYSQL\_ALLOW\_EMPTY\_PASSWORD erlaubt das starten mit leerem root-passwort

7.

Image enthält (sehr wahrscheinlich) Software welche eigenen Lizenzbestimmungen unterliegt.

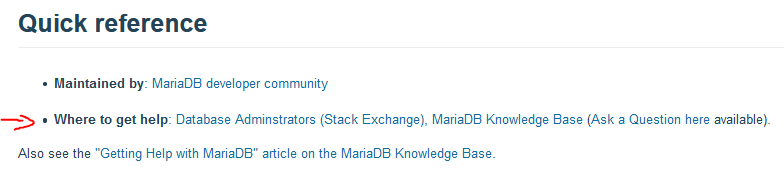
Möglicherweise zusätzliche Informationen im repo-info repository.

8.

the Docker Community Forums

the Docker Community Slack

Stack Overflow



9.

Aus dem Container heraus (exec) einen DUMP der Datenbank erzeugen.

-> docker exec some-mariadb sh -c 'exec mysqldump --all-databases -uroot -p"$MYSQL\_ROOT\_PASSWORD"' > /some/path/on/your/host/all-databases.sql

DUMP für restore nutzen

-> docker exec -i some-mariadb sh -c 'exec mysql -uroot -p"$MYSQL\_ROOT\_PASSWORD"' < /some/path/on/your/host/all-databases.sql

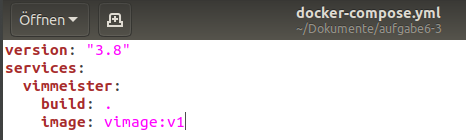


10.

Über den Link bei supported tags kann die Dockerfile angezeigt werden.

AUFGABE 6.3

Dockerfile: siehe Aufgabe 6.1

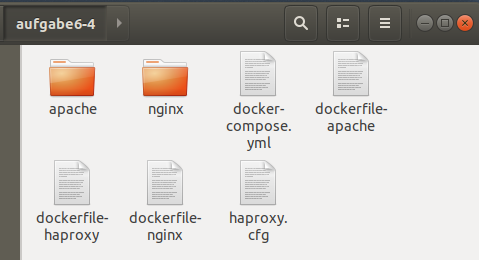


sudo docker-compose up

sudo docker-compose build / sudo docker-compose up --build (<- Image wird neu gebaut falls schon vorhanden)

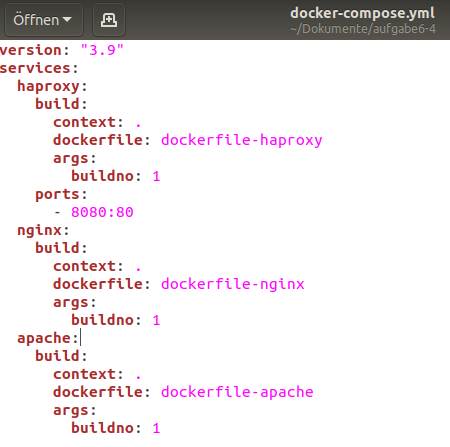
AUFGABE 6.4

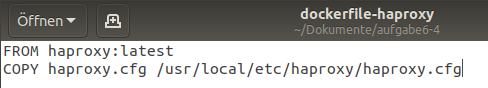
HAProxy mit Apache und Nginx

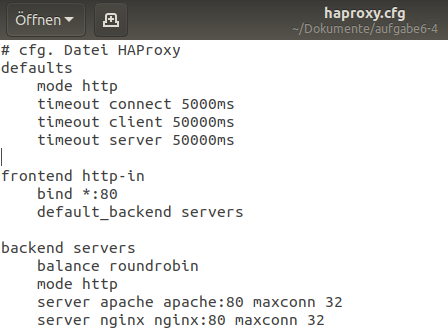


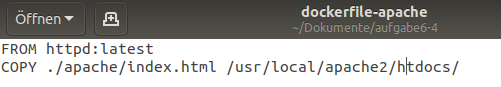
**benötigt werden:** yml-datei, dockerfile-haproxy, haproxy.cfg, dockerfile-apache, dockerfile-nginx

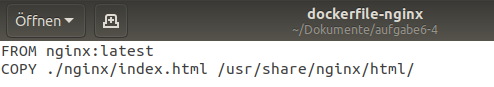
/apache/index.html und /nginx/index.html

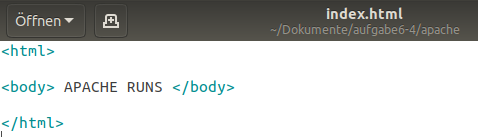


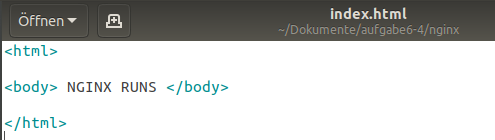








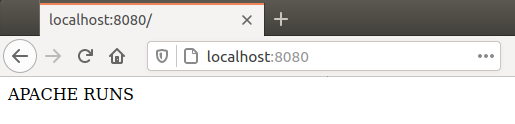


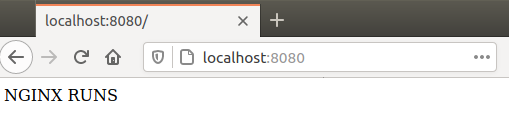


sudo docker-compose up

sudo docker-compose down -v

Aufruf von localhost:8080 zeigt die editierten index.html. Bei aktualisieren wechselt Apache und Nginx (round robin)





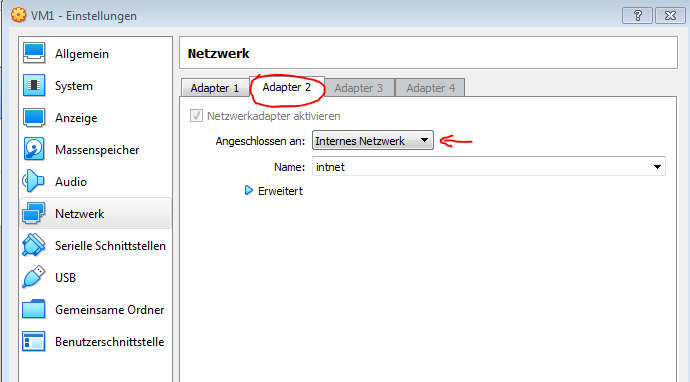
AUFGABENBLATT 7

DOCKER SWARM

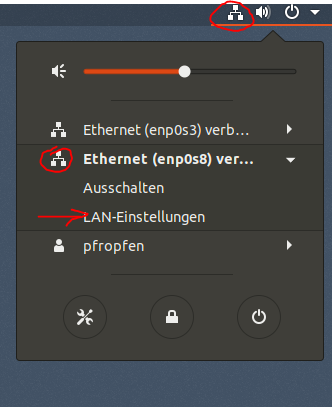
**AUFGABE 7.1**

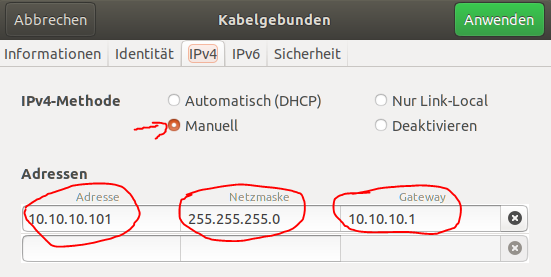
Mehrere VMs aufsetzten (bzw. klonen).

Für jede VM einen zweiten Netzwerkadapter erstellen für internes Netzwerk:



In den VMs in den LAN-Einstellungen des zweiten Adapters (internes Netzwerk) die IPv4 Adresse Manuell eintragen:





IP: 10.10.10.101 für die erste VM …..102 für die zweite und so weiter.

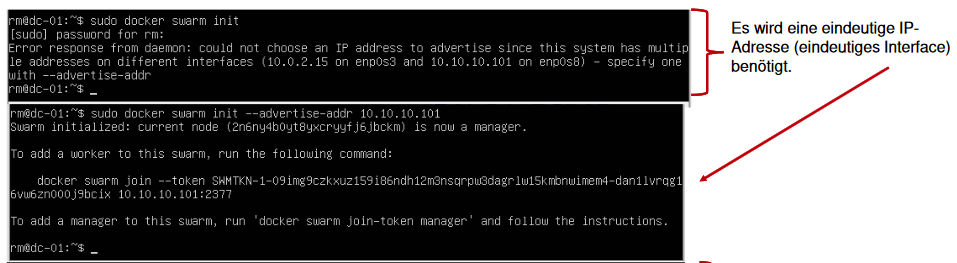
Gateway ist der virtuelle Router der VM.

Nun kann der Docker Swarm aufgesetzt werden.

In der ersten (leader) VM:

sudo docker swarm init –advertise-addr 10.10.10.101

Das System gibt daraufhin einen Token zurück mit dessen Hilfe dem Swarm beigetreten werden kann:



Mit den anderen VMs kann nun beigetreten werden:

sudo docker swarm join –token SWMTKN-1-685ruek4jtxti5mxhblfj7tw26jl1nv3gnr1smbada65aqhtpe-7g1j1mnjzqm40jyn5sn3psgns 10.10.10.101:2377

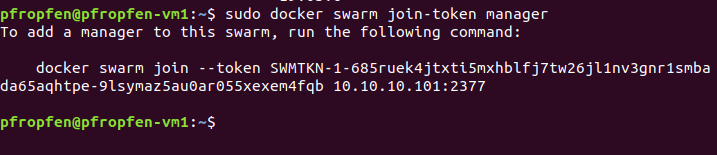
Der Swarm kann mit

sudo docker swarm leave

wieder verlassen werden. (Beim Leader muss zusätzlich -f angegeben werden)

Um eine Maschine als Manager hinzuzufügen, **in der leader VM** eingeben:

sudo docker swarm join-token manager

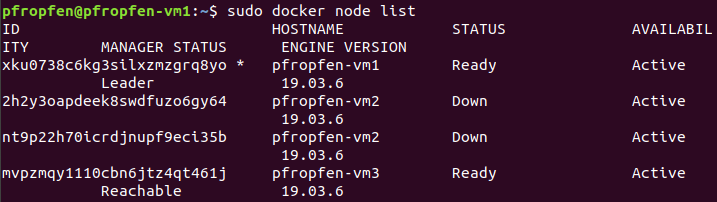


Das System gibt einen speziellen token zurück mit dem dann in der 3.VM als Manager gejoined werden kann.

Mit

sudo docker node list

können die Nodes des Swarm aufgelistet werden.



Manager Status „Leader“ zeigt den Leader an.

Manager Status „Reachable“ zeigt einen Manager an.

Kein Manager Status bedeutet Worker.

Mit

sudo docker node rm [node-ID]

Kann ein Node aus dem Swarm entfernt werden.

**AUFGABE 7.2**



Availability = Active zeigt, dass der Node Container hosten kann. (active/drain (stop)/pause)

Per Default können alle Nodes Container hosten.

Mit

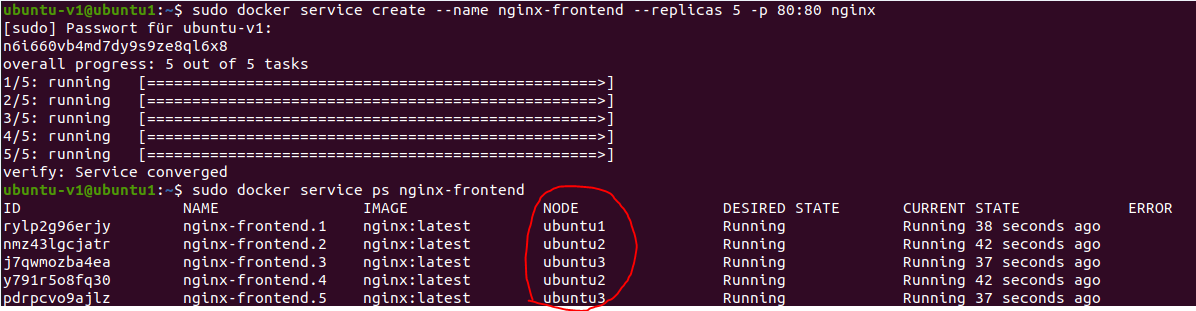
sudo docker service create –name nginx-frontend –replicas 5 -p 80:80 nginx

werden 5 Replicas des Dienstes Nginx gestartet und auf die Nodes verteilt.

Mit:

sudo docker service ps nginx-frontend

wird angezeigt auf welchen Nodes jeweils die Dienste laufen:

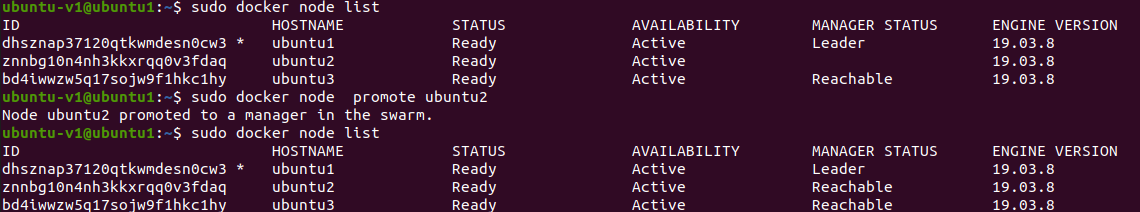


In diesem Cluster darf kein Manager ausfallen, da immer mehr als die Hälfte der Manager laufen müssen. (Da die Hälfte 1 ist, und so mehr als die Hälfte =2 sind, müssen demnach alle Manager laufen ).

Mit

sudo docker node promote ubuntu2

Kann ein Worker-Node zum Manager gemacht werden.



Mit

sudo docker service rm [service-name]

Kann ein Service gestoppt (bzw. entfernt) werden.

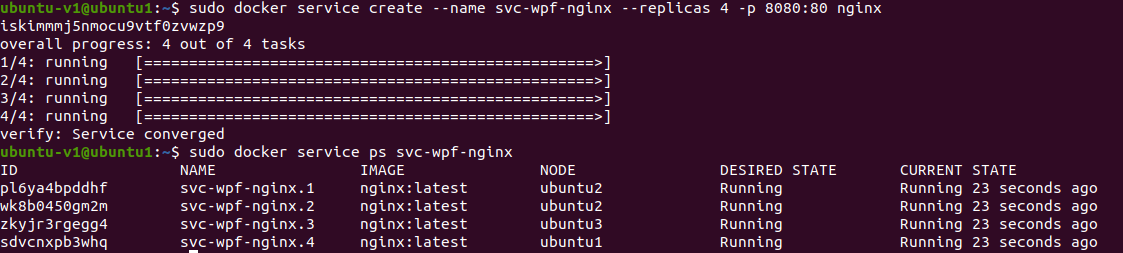
**AUFGABE 7.3**

**1.**

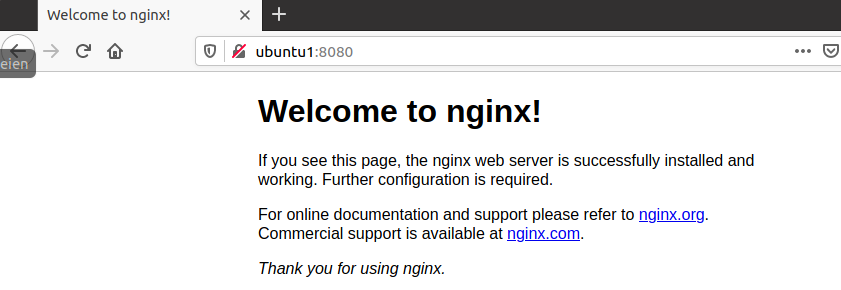
sudo docker service create –-name svc-wpf-nginx -–replicas 4 -p 8080:80 nginx

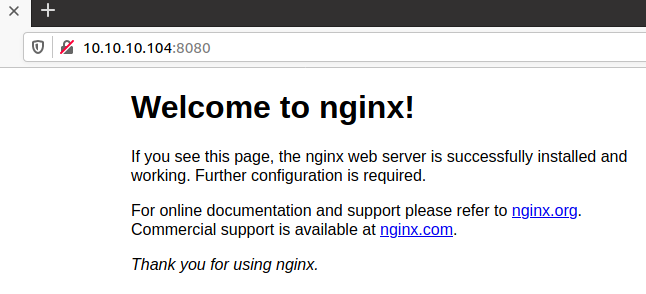
sudo docker service ps svc-wpf-nginx

**2.**



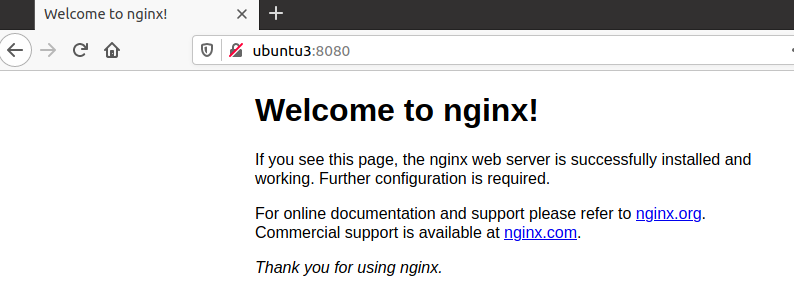
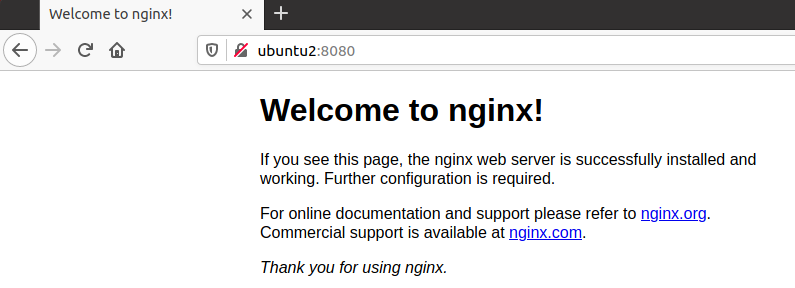
**3.**





Über die entsprechende IP kann ebenso die nginx-seite einer anderen Node aufgerufen werden!

Auf den anderen Maschinen ist nginx entsprechend auch aufrufbar:

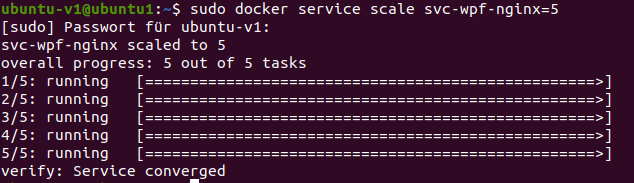


**4.**

Mit

sudo docker service scale svc-wpf-nginx=5

kann ein zusätzliches Replica für einen Service erzeigt werden.



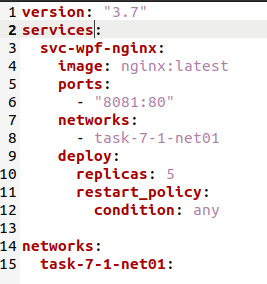
Der Service muss dazu also nicht neu erstellt werden.

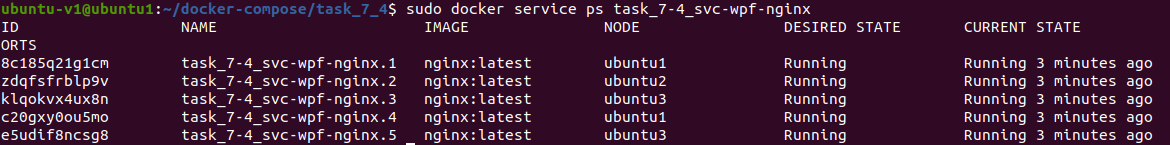
**AUFGABE 7.4**

sudo docker stack deploy -c docker-compose.yml task\_7-4



yml-Datei:



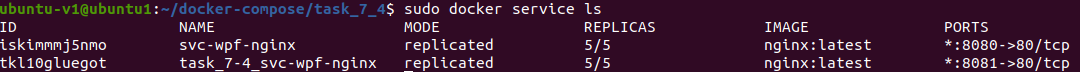


Im Kontext des Swarms ist der Name des Services nun:

Task\_7-4\_svc-wpf-nginx(.1)

[StackName]\_[ServiceName aus yml-Datei](.[ReplicaNo])

Da der Servicename am Anfang den Stack-Namen als Präfix beinhaltet, kommt er nicht mit anderen Diensten in die Quere.



**AUFGABENBLATT 8**

**8.1**

Mit

sudo docker stack rm STACKNAME

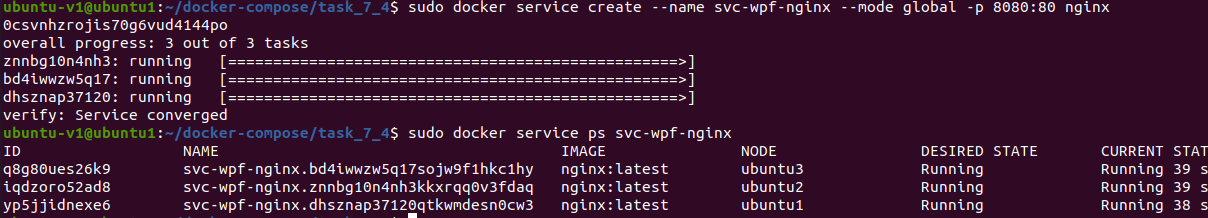
Kann der Stack gestoppt/entfernt werden

sudo docker service rm SERVICENAME

Kann der Service gestoppt/entfernt werden

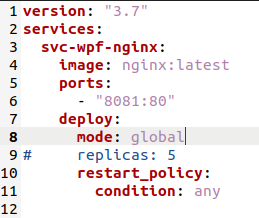
8.1.1

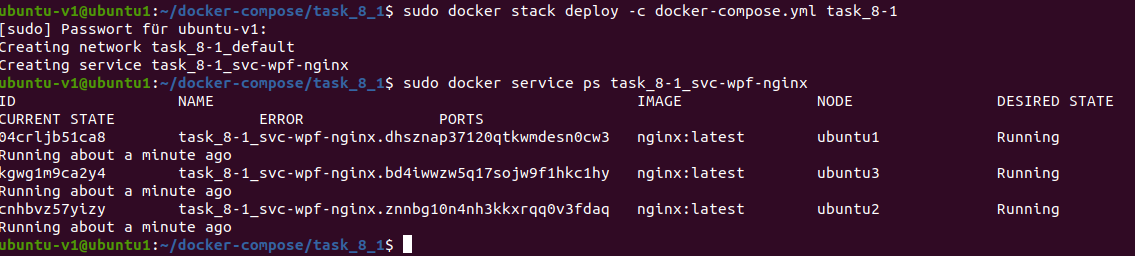
sudo docker service create –name svc-wpf-nginx –mode global -p 8080:80 nginx



**Mit stack deploy und yml-Datei:**

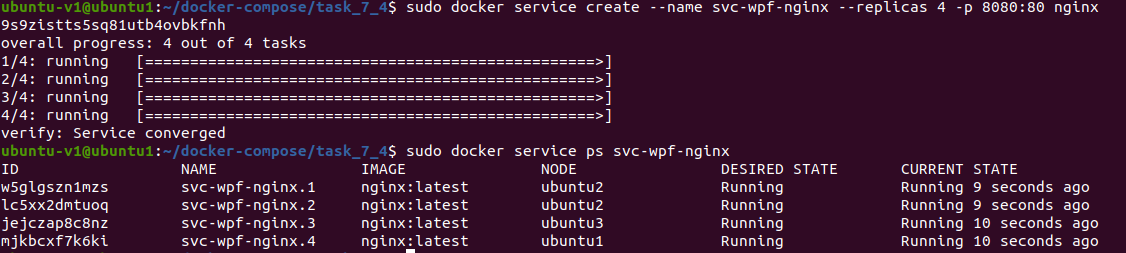
sudo docker stack deploy -c docker-compose.yml mystack





8.1.2

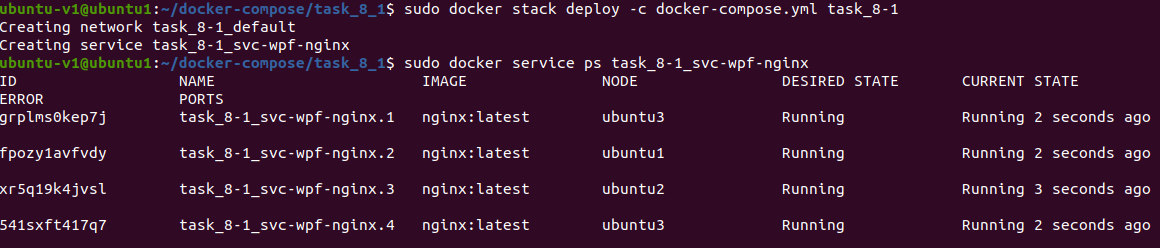
sudo docker service create –name svc-wpf-nginx –replicas 4 -p 8080:80 nginx



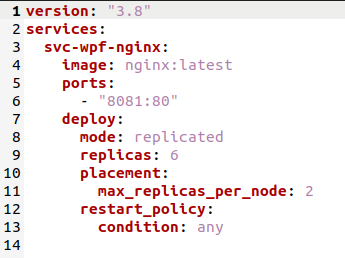
**Mit stack deploy und yml-Datei:**

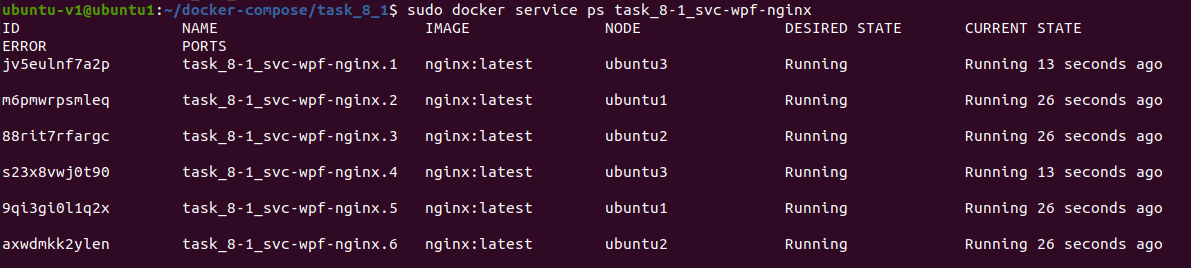
sudo docker stack deploy -c docker-compose.yml mystack





8.1.3





8.1.4

Zuerst Labels für die Nodes anlegen:

sudo docker node update –label-add type=stage1 ubuntu1

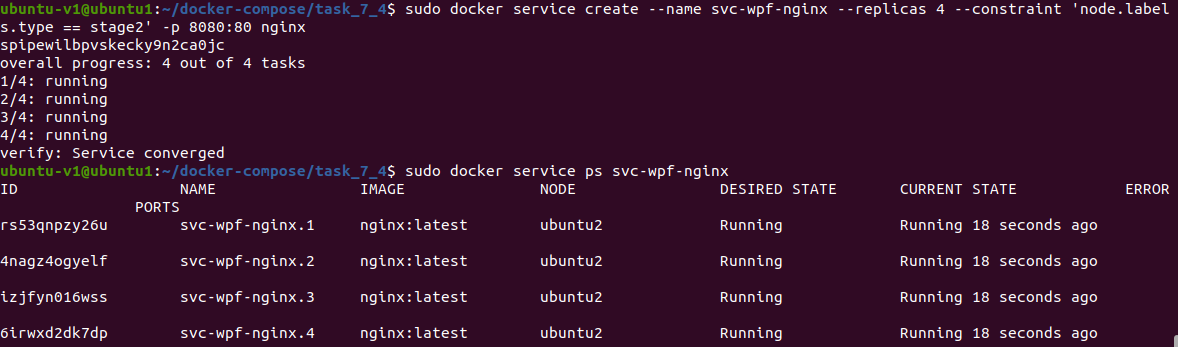
sudo docker node update –label-add type=stage2 ubuntu2

sudo docker node update –label-add type=stage3 ubuntu3

Dann den Service mit Constraint starten:

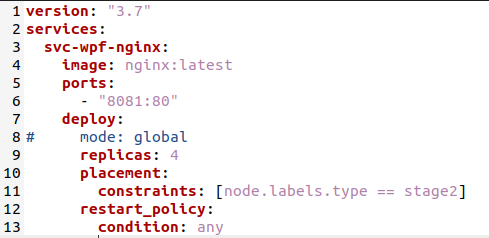
sudo docker service create –name svc-wpf-nginx –replicas 4 –constraint ‘node.labels.type == stage2’ -p 8080:80 nginx

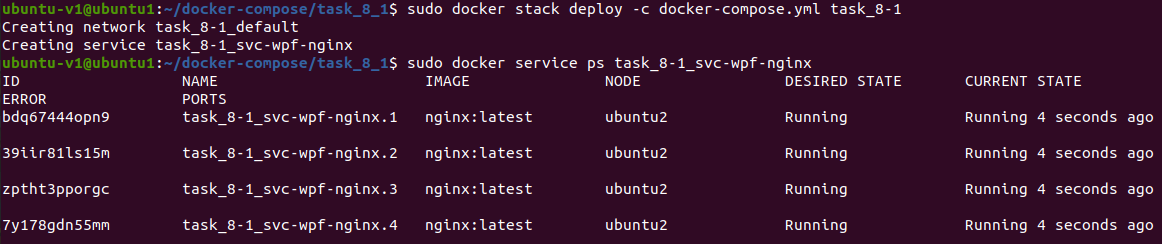
Nun laufen alle Replicas nur auf einer Node (ubuntu2)



**Mit stack deploy und yml-Datei:**

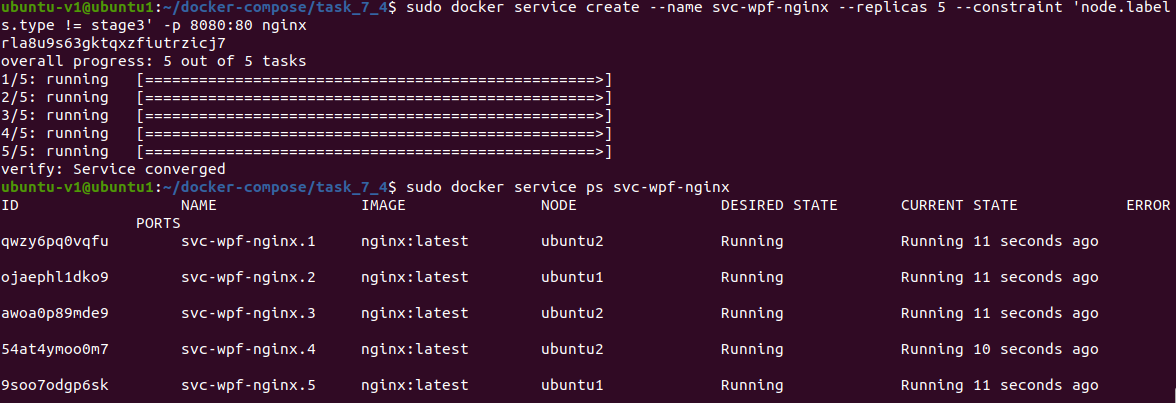
sudo docker stack deploy -c docker-compose.yml mystack

****

****

8.1.5

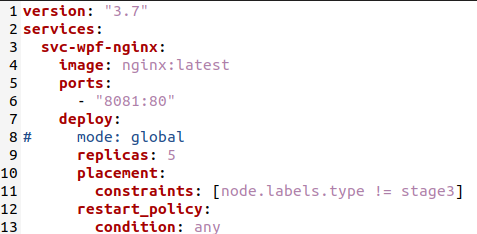
sudo docker service create –name svc-wpf-nginx –replicas 5 –constraint ‘node.labels.type != stage3’ -p 8080:80 nginx



Tasks laufen nicht auf ubuntu3.

**Mit stack deploy und yml-Datei:**

sudo docker stack deploy -c docker-compose.yml mystack

****

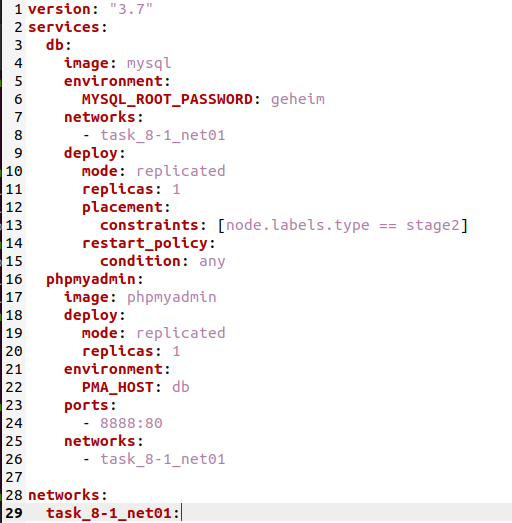
****

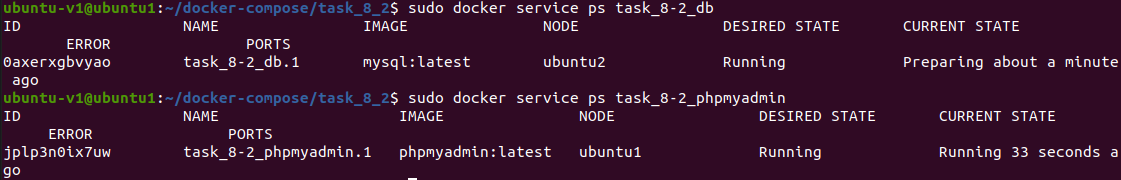
**8.2**

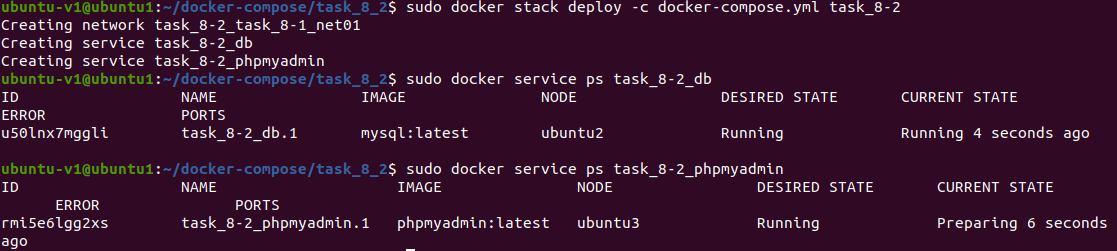
sudo docker stack deploy -c docker-compose.yml task\_8-2

sudo docker service ps task\_8-2\_db

sudo docker service ps task\_8-2\_phpmyadmin







MySql läuft immer auf Ubuntu2 während phpmyadmin random auf einer der Maschinen läuft.

Wenn MySQL im Laufe der Zeit auf verschiedenen Nodes gehostet wird,

wird nach und nach auf alles Nodes eine Teil-Datenbank angelegt. Datenerhaltung gefährdet.

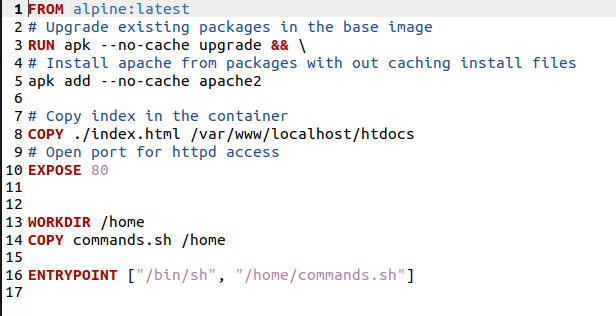
**8.3**

Das selbst zu erzeugende Image muss im Voraus erstellt und in einer eigenen Registry gespeichert werden damit anschließend jeder Node im Cluster darauf zugreifen kann.

Bei docker stack deploy kann innerhalb der yml-Datei kein eigenes Image erzeugt werden (anders bei docker-compose).

**8.4**

Dateien für Image: dockerfile, commands.sh und index.html



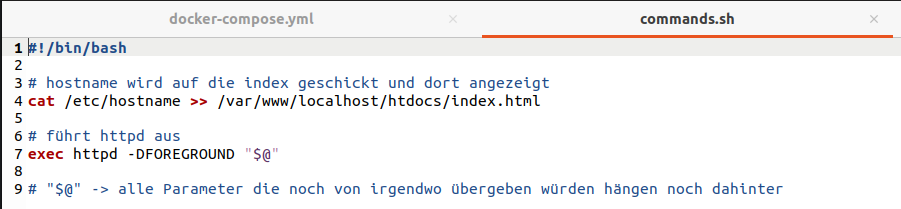




Image bauen:

sudo docker build -t httpd\_swarm .

Registry anlegen:

sudo docker service create --name reg --replicas 1 --constraint 'node.labels.type == stage2' -p 5000:5000 registry

Image Taggen:

sudo docker tag httpd\_swarm 10.10.10.103:5000/httpd\_swarm:Version1

Image in Registry pushen:

sudo docker push 10.10.10.103:5000/httpd\_swarm:Version1

In /etc/docker/daemon.json folgende Zeile ergänzen:

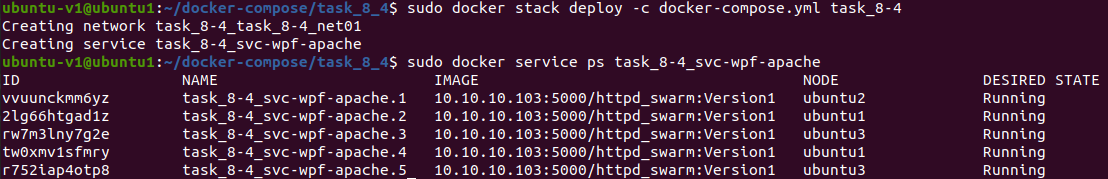
{ "insecure-registries":["10.10.10.103:5000"] }

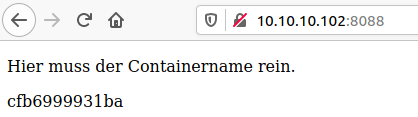
Danach den Daemon neustarten:

sudo service docker restart

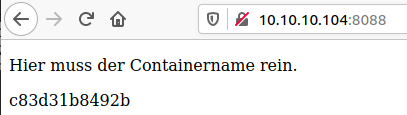
Service ausrollen:

sudo docker stack deploy -c docker-compose.yml task\_8-4









Container sind nun über alle Nodes erreichbar.

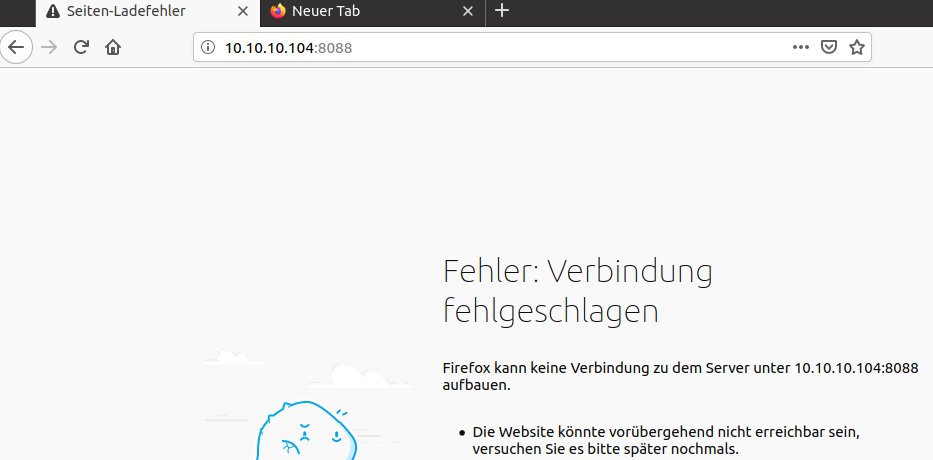
**8.5**

Auf VM3 (worker node):

sudo service docker stop

Dann von VM1 aus die Adresse: 10.10.10.104:8088 im Browser aufrufen.

„Seite kann nicht angezeigt werden“ da der Node gestoppt wurde.



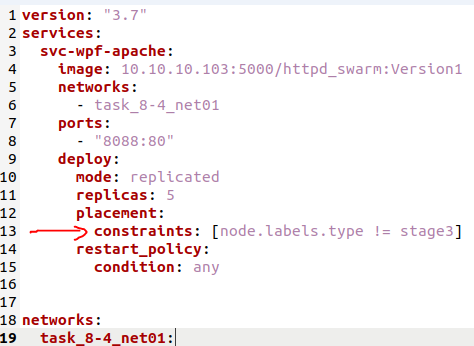
Auf VM3 (worker node): Dockerd erneut starten

sudo service docker start

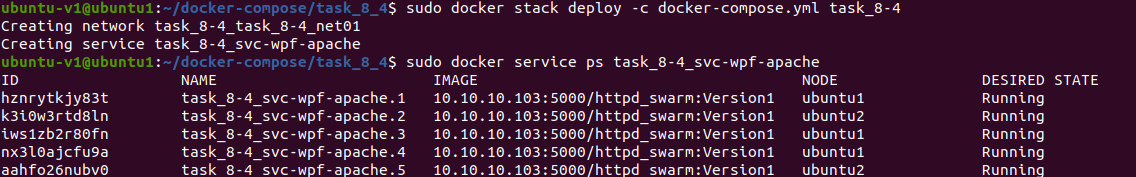
Nun den Service stoppen/löschen:

sudo docker stack rm task\_8-4

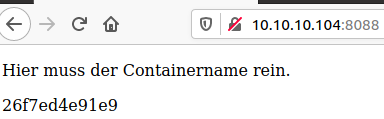
Constraint in der yml-Datei hinzufügen:



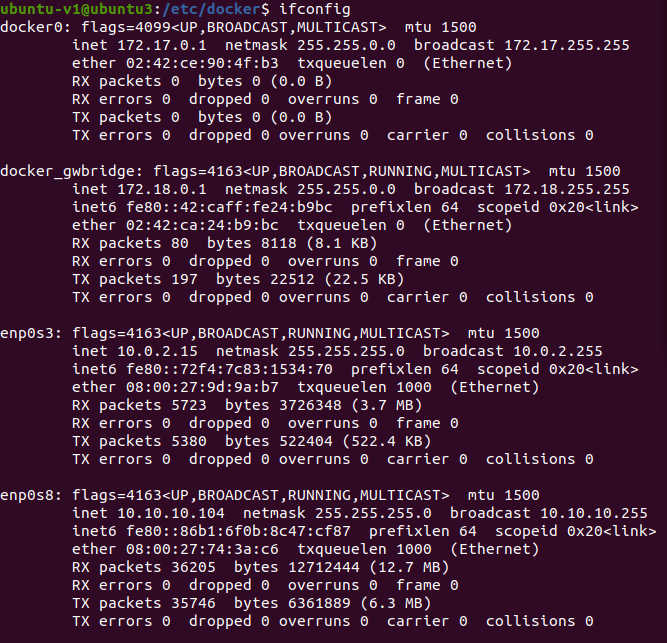
Service neu ausrollen:



Container werden nur auf ubuntu1 und ubuntu2, also den Manager-Nodes gehostet.



Ubuntu3 hat dier IP 10.10.10.104:



Dennoch kann über die IP der Node ubuntu3 der Container aufgerufen werden.

Wenn versucht wird ein Service auf einem Node zu erreichen auf dem er nicht läuft, wird die Anfrage an einen anderen Node geleitet (auf dem der Service läuft). Dadurch sieht es von außen so aus, als würde der Node den Service anbieten. Dies wird von Swarm automatisch erledigt.