

Betriebssysteme Projekt WS22/23



Thema 2: Clustering mit Docker Dokumentation

Von Ahmed Abdaal und Jenny Rötzer

Inhaltsverzeichnis

1. VM einrichten

2. Installation von Docker

3. NFS

4. Portainer

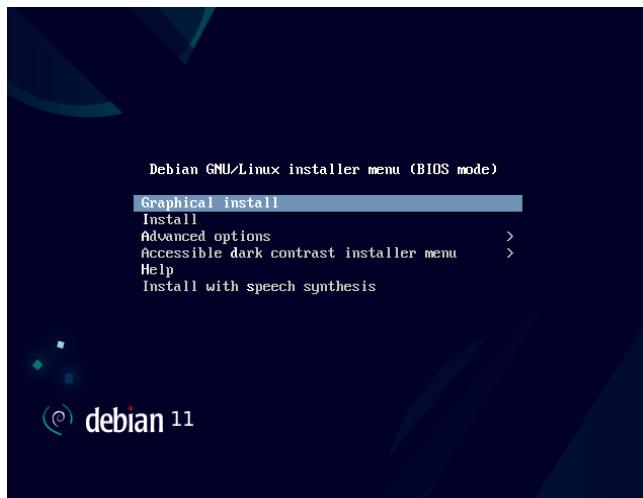
5. Docker Swarm einrichten

6. Webservice mit Node, Nginx und MongoDB

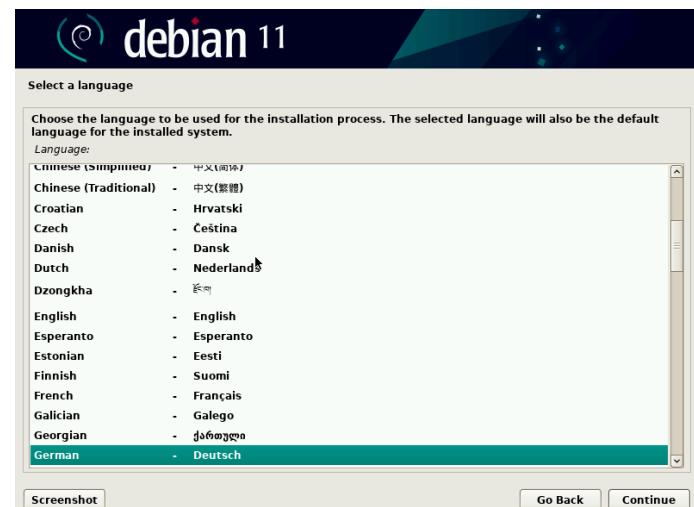
7. Test

8. Quellen

1. VM einrichten



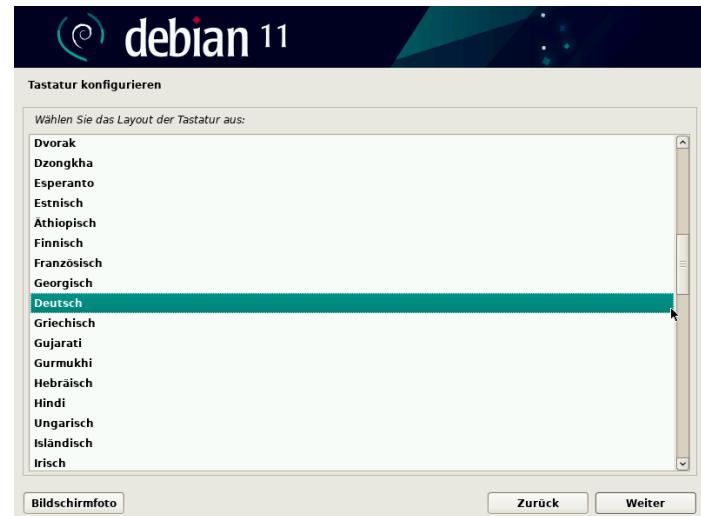
1. Schritt: „Graphical Install“ Option auswählen und mit Enter bestätigen



2. Schritt: Die bevorzugte Sprache auswählen und auf Continue drücken



3. Schritt: Das Land auswählen und bestätigen



4. Schritt: Die Tastaturbelegung auswählen

Netzwerk einrichten

Bitte geben Sie den Namen dieses Rechners ein.

Der Rechnername ist ein einzelnes Wort; über ihn wird Ihr Rechner im Netzwerk identifiziert. Wenn Sie Ihren Rechnernamen nicht kennen, fragen Sie den Netzwerkadministrator. Wenn Sie ein lokales Heimnetz aufbauen, ist es egal, was Sie angeben.

Rechnername:

Bildschirmfoto **Weiter** **Zurück**

Netzwerk einrichten

Der Domain-Name ist der rechte Teil Ihrer Internetadresse nach Ihrem Rechnernamen. Er endet oft mit .de, .com, .net oder .org. Wenn Sie ein lokales Heimnetz aufbauen, ist es egal, was Sie angeben. Diese Information sollte dann aber auf allen Rechnern gleich sein.

Domain-Name:

Bildschirmfoto **Weiter** **Zurück**

5. Schritt: Einrichtung des Netzwerks und ein Rechnername vergeben

6. Schritt: Ein Domain-Name vergeben

Benutzer und Passwörter einrichten

Sie müssen ein Passwort für >root<, das Systemadministrator-Konto, angeben. Ein bösartiger Benutzer oder jemand, der sich nicht auskennt und Root-Rechte besitzt, kann verheerende Schäden anrichten. Deswegen sollten Sie darauf achten, ein Passwort zu wählen, das nicht einfach zu erraten ist. Es sollte nicht in einem Wörterbuch vorkommen oder leicht mit Ihnen in Verbindung gebracht werden können.

Ein gutes Passwort enthält eine Mischung aus Buchstaben, Zahlen und Sonderzeichen und wird in regelmäßigen Abständen geändert.

Das Passwort für den Superuser root sollte nicht leer sein. Wenn Sie es leer lassen, wird der root-Zugang deaktiviert und der als erstes eingerichtete Benutzer in diesem System erhält die nötigen Rechte, mittels >sudo<-Befehl zu root zu wechseln.

Hinweis: Sie werden das Passwort während der Eingabe nicht sehen.

Root-Passwort:

Passwort im Klartext anzeigen

Bitte geben Sie dasselbe root-Passwort nochmals ein, um sicherzustellen, dass Sie sich nicht vertippt haben.
Bitte geben Sie das Passwort zur Bestätigung nochmals ein:

Passwort im Klartext anzeigen

Bildschirmfoto **Weiter** **Zurück**

Benutzer und Passwörter einrichten

Für Sie wird ein Konto angelegt, das Sie statt dem root-Konto für die alltägliche Arbeit verwenden können.

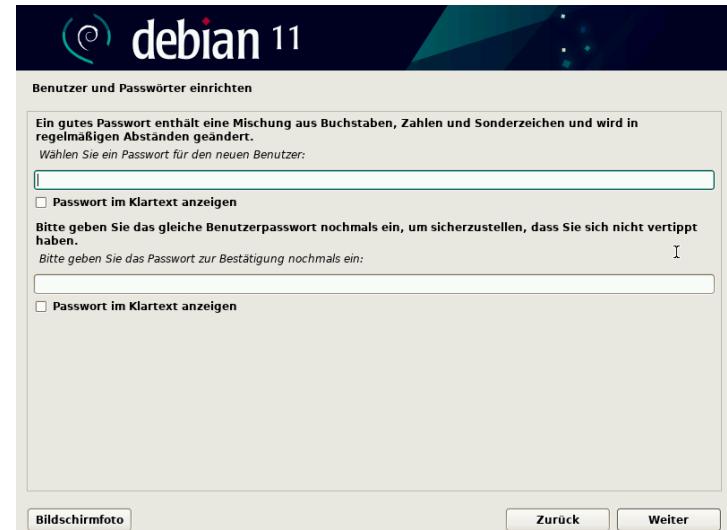
Bitte geben Sie den vollständigen Namen des Benutzers an. Diese Information wird z.B. im Absender von E-Mails, die er verschiickt, oder in Programmen, die den Namen des Benutzers anzeigen, verwendet. Ihr kompletter Name wäre sinnvoll.

Vollständiger Name des neuen Benutzers:

Bildschirmfoto **Weiter** **Zurück**

7. Schritt: Ein sicheres Kennwort für den Benutzer eingeben

8. Schritt: Name des ersten Benutzers eingeben



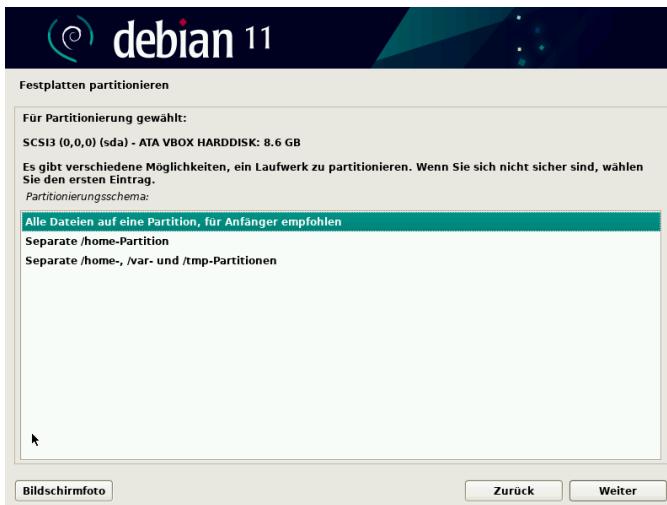
9. Schritt: Benutzername vergeben, meistens wird der Vorname vom Benutzer gewählt

10. Schritt: Ein sicheres Passwort für den Benutzer eingeben



11. Schritt: Die vollständige Festplatte für die Partitionierung auswählen

12. Schritt: Die aufgeführte Auswahl mit Weiter bestätigen



13. Schritt: 1. Option mit „Alle Dateien auf eine Partition“ auswählen und Weiter klicken

14. Schritt: 2. Option „Partitionierung beenden und Änderungen übernehmen“ auswählen und auf Weiter drücken



15. Schritt: „Ja“ auswählen, um die Änderungen auf die Festplatte zu schreiben, und auf Weiter klicken



16. Schritt: Die Option „Nein“ auswählen, da kein weiteres Medium eingelesen werden soll und bestätigen



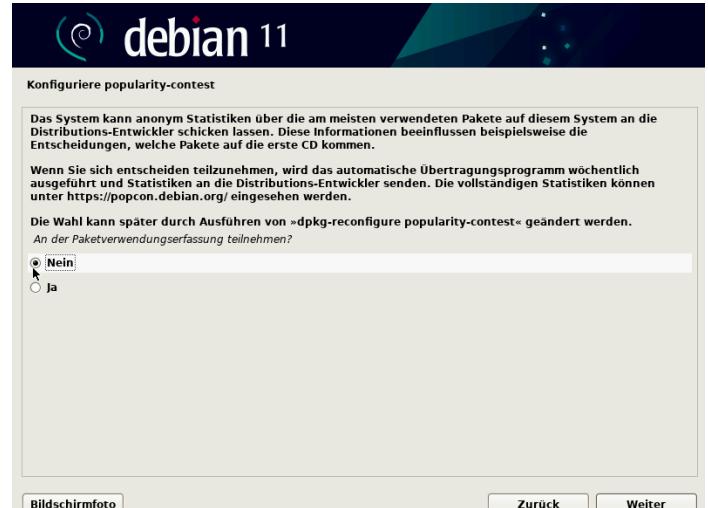
17. Schritt: Land auswählen, in dem der Spiegelserver stehen soll



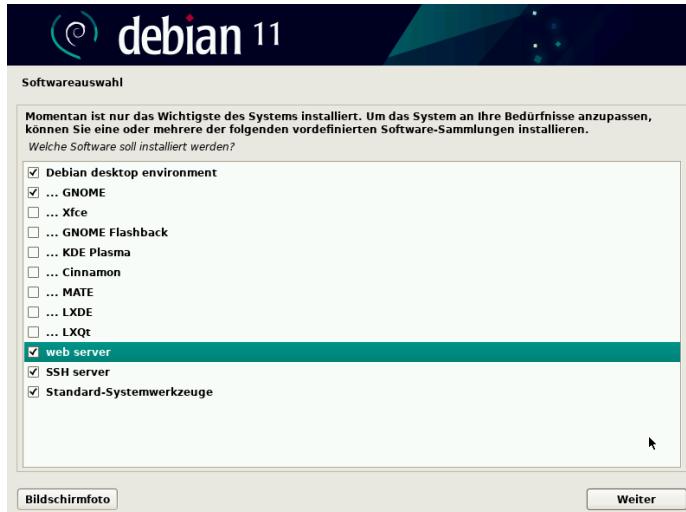
18. Schritt: Den Spiegelserver wählen, der die beste Internetverbindung herstellen kann



19. Schritt: Proxy-Daten leer lassen und auf Weiter klicken



20. Schritt: An der Paketverwendungserfassung nicht teilnehmen und mit „Nein“ fortfahren



21. Schritt: Auswahl der Softwarepakete, die zusätzlich installiert werden sollen und mit Weiter



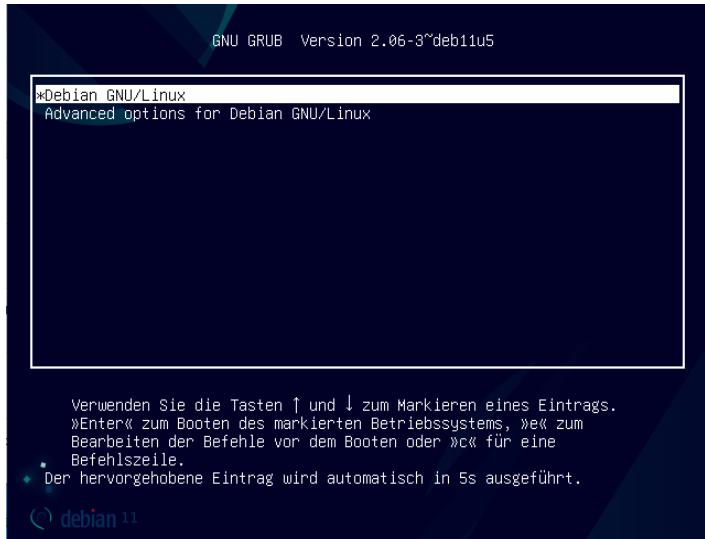
22. Schritt: Mit „Ja“ den GRUB-Bootloader installieren und fortsetzen



23. Schritt: Die angegebene Festplatte auswählen und auf Weiter klicken



24. Schritt: Die Installation ist abgeschlossen und mit Weiter fortfahren



```
[ OK ] Finished Update UTMP about System Boot/Shutdown.
[ OK ] Started Rule-based Manager for Device Events and Files.
      Starting Show Plymouth Boot Screen...
[ OK ] Started Network Time Synchronization.
[ OK ] Reached target System Time Set.
[ OK ] Reached target System Time Synchronized.
[ OK ] Started Show Plymouth Boot Screen.
[ OK ] Started Forward Password Requests to Plymouth Directory Watch.
[ OK ] Reached target Local Encrypted Volumes.
[ OK ] Found device VBOX_HARDDISK 5.
      Activating swap /dev/disk/by-uuid/c55914c3-ad0c-4dec-a890-89dd4824b3a0...
[ OK ] Activated swap /dev/disk/by-uuid/c55914c3-ad0c-4dec-a890-89dd4824b3a0.
[ OK ] Reached target Swap.
[ OK ] Finished Load/Save Random Seed.
[ OK ] Finished Load AppArmor profiles.
[ OK ] Reached target System Initialization.
[ OK ] Started CUPS Scheduler.
[ OK ] Started Trigger anacron every hour.
[ OK ] Started Trigger anacron every hour.
[ OK ] Started Daily apt download activities.
[ OK ] Started Daily apt upgrade and clean activities.
[ OK ] Started Periodic ext4 Online Metadata Check for All Filesystems.
[ OK ] Started Discard unused blocks once a week.
[ OK ] Started Refresh fwupd metadata regularly.
[ OK ] Started Daily rotation of log files.
[ OK ] Started Daily man-db regeneration.
[ OK ] Started Daily Cleanup of Temporary Directories.
[ OK ] Reached target Timers.
[ OK ] Listening on Avahi mDNS/DNS-SD Stack Activation Socket.
[ OK ] Listening on CUPS Scheduler.
[ OK ] Listening on D-Bus System Message Bus Socket.
[ OK ] Reached target Sockets.
[ OK ] Reached target Basic System.
      Starting Accounts Service...
[ OK ] Started Run anacron jobs.
      Starting Avahi mDNS/DNS-SD Stack...
```

24. Schritt: Debian 11 zum ersten Mal booten

25. Schritt: Dienste und Dateien werden geprüft und gestartet



26. Schritt: Anmeldebildschirm, um mit einem Benutzer zu starten

2. Installation von Docker:

Zunächst laden wir Docker auf den Virtuellen Maschinen runter. Der Prozess ist auf allen vier Maschinen ist derselbe.

Damit wir Docker installieren können müssen wir ein Verzeichnis für Docker auf unseren Maschinen einrichten.

Schritt 1: Pakete aktualisieren und dann neue installieren, damit *apt* das Verzeichnis über HTTPS benutzen kann

```
$ sudo apt-get update

$ sudo apt-get install \
  ca-certificates \
  curl \
  gnupg \
  lsb-release
```

Schritt 2: Docker GPG key hinzufügen

```
$ sudo mkdir -p /etc/apt/keyrings

$ curl -fsSL https://download.docker.com/linux/debian/gpg | sudo
gpg --dearmor -o /etc/apt/keyrings/docker.gpg
```

Schritt 3: Verzeichnis anlegen

```
$ echo \
"deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.gpg]
https://download.docker.com/linux/debian \
$(lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list >
/dev/null
```

Nachdem wir das Verzeichnis eingerichtet haben, installieren wir jetzt die Docker Engine.

Schritt 1: apt Pakete aktualisieren

```
$ sudo apt-get update
```

Schritt 2: die neuste Version der Docker Engine herunterladen

```
$ sudo apt-get install docker-ce docker-ce-cli containerd.io docker-compose-
plugin
```

Schritt 3: testen mit *hello-world* image

```
$ sudo docker run hello-world
```

Bei erfolgreichem Download sollte die Ausgabe so ausschauen:

```
Status: Downloaded newer image for hello-world:latest
Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
1. The Docker client contacted the Docker daemon.
2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
   (arm64v8)
3. The Docker daemon created a new container from that image which runs the
   executable that produces the output you are currently reading.
4. The Docker daemon streamed that output to the Docker client, which sent it
   to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
$ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
https://hub.docker.com/

For more examples and ideas, visit:
https://docs.docker.com/get-started/
```

Docker als non root-user benutzen:

Vor jedem Docker Befehl müssen wir *sudo* anhangen, weil der Docker Deamon immer als *root* User läuft. Um das zu vermeiden, tun wir unseren User in der Docker Gruppe rein.

Schritt 1: Gruppe erstellen (falls sie noch nicht existiert)

```
$ sudo groupadd docker
```

Schritt 2: User in der Docker Gruppe hinzufügen

```
$ sudo usermod -aG docker $USER
```

Danach sollte man seine VM neu starten, damit die Veränderungen gültig werden.

Schritt 3: *hello-world* image starten ohne sudo

```
$ docker run hello-world
```

Jetzt können wir ohne sudo docker benutzen, was uns später vieles erleichtert.

```
parallels@debian-gnu-linux-11:~$ docker run hello-world
Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (arm64v8)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
$ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
https://hub.docker.com/

For more examples and ideas, visit:
https://docs.docker.com/get-started/
```

3. NFS

Wir benutzen *Network File System* als unser gemeinsamen Speichermittel für unseren Swarm Cluster. Dieses Protocol benutzt den Server/Client Prinzip. Eine Maschine ist der Server (Manager) und die anderen sind die Clients (Worker). Dabei sind, wie man schon ahnt, alle Daten auf dem Server gespeichert und die Clients können darauf zugreifen, so als wären die Dateien lokal gespeichert.

Dabei müsst ihr darauf achten, dass die Einrichtung von NFS auf der Server-Maschine anders ist als bei den Client-Maschinen.

Schritt 1: Pakete installieren

auf der Server-Maschine

```
$ sudo apt update
$ sudo apt-get install nfs-kernel-server
```

auf den Client-Maschinen

```
$ sudo apt update
$ sudo apt-get install nfs-common
```

Schritt 2: Verzeichnis erstellen und Rechte ändern

auf der Server-Maschine

```
$ mkdir nfs-share
$ sudo chown nobody:nogroup nfs-share
```

auf den Client-Maschinen

```
$ mkdir nfs-server
```

Schritt 3: NFS freigeben

auf der Server-Maschine

```
$ sudo nano /etc/exports
```

```
GNU nano 5.4                               /etc/exports *
# /etc/exports: the access control list for filesystems which may be exported
#           to NFS clients. See exports(5).
#
# Example for NFSv2 and NFSv3:
# /srv/homes      hostname1(rw,sync,no_subtree_check) hostname2(ro,sync,no_subtree_check)
#
# Example for NFSv4:
# /srv/nfs4       gss/krb5i(rw,sync,fsid=0,crossmnt,no_subtree_check)
# /srv/nfs4/homes gss/krb5i(rw,sync,no_subtree_check)
#
/home/parallels/nfs-share
```

Nachdem öffnen von `/etc/exports` sollte euer Bildschirm so ausschauen. Die drinstehenden Kommentare zeigen uns wie die Eingabe erfolgen soll.

Wir ergänzen ganz unten:

/home/parallels/nfs-share 10.211.55.13(rw,sync,no_root_squash,no_subtree_check)

Pfad zum Verzeichnis
auf Host Client_ip Shareoptionen

Wichtig: kein Leerzeichen nach dem Komma machen!!!

```
GNU nano 5.4                               /etc/exports *
# /etc/exports: the access control list for filesystems which may be exported
#           to NFS clients. See exports(5).
#
# Example for NFSv2 and NFSv3:
# /srv/homes      hostname1(rw,sync,no_subtree_check) hostname2(ro,sync,no_subtree_check)
#
# Example for NFSv4:
# /srv/nfs4       gss/krb5i(rw,sync,fsid=0,crossmnt,no_subtree_check)
# /srv/nfs4/homes gss/krb5i(rw,sync,no_subtree_check)
#
/home/parallels/nfs-share 10.211.55.12(rw,sync,no_subtree_check)
```

Der Pfad, der am Anfang steht `/home/parallels/nfs-share` ist der Pfad zu dem geteilten Verzeichnis. Danach kommt die IP-Adresse oder Hostname des Clients, im unserem Fall `10.211.55.13`. Die im klammer stehende Syntaxen sind die Shareoptionen.

rw:

client kann Lesen und Schreiben

sync:

Synchroner Datentransfer

no_root_squash:

begrenzt die Rechte für einen SU per Remotezugriff

no_subtree_check:

überprüft Unterverzeichnisbäumen nicht ab

Wenn man mehrere Clients hat, so fügt man dasselbe wie oben aber mit der IP-Adresse des anderen Clients hinten an der Zeile an.

```
GNU nano 5.4                               /etc/exports
# /etc/exports: the access control list for filesystems which may be exported
#           to NFS clients. See exports(5).
#
# Example for NFSv2 and NFSv3:
# /srv/homes      hostname1(rw,sync,no_subtree_check) hostname2(ro,sync,no_subtree_check)
#
# Example for NFSv4:
# /srv/nfs4        gss/krb5i(rw,async,fsid=0,crossmnt,no_subtree_check)
# /srv/nfs4/homes  gss/krb5i(rw,async,no_subtree_check)
#
/home/parallels/nfs-share 10.211.55.12(rw,async,no_root_squash,no_subtree_check)
                           10.211.55.13(rw,async,no_root_squash,no_subtree_check)
                           10.211.55.14(rw,async,no_root_squash,no_subtree_check)
```

Nachdem wir die Datei gespeichert bzw. geschlossen haben, müssen wir den neuen Befehlen in der *exports*-Datei aktivieren.

auf der Server-Maschine

```
$ sudo exportfs -ra
```

Wir können uns die aktiven Maschinen auflisten

auf der Server-Maschine

```
$ sudo exportfs -v
```

Ausgaben bei unserem Fall schaut so aus:

```
parallels@manager:~$ sudo exportfs -v
/home/parallels/nfs-share
  10.211.55.12(rw,wdelay,no_root_squash,no_subtree_check,sec=sys,rw,secure,no_root_squash,no_all_squash)
/home/parallels/nfs-share
  10.211.55.13(rw,wdelay,no_root_squash,no_subtree_check,sec=sys,rw,secure,no_root_squash,no_all_squash)
/home/parallels/nfs-share
  10.211.55.14(rw,wdelay,no_root_squash,no_subtree_check,sec=sys,rw,secure,no_root_squash,no_all_squash)
```

(Schrift 4: NFS Server neustarten) noch testen

auf der Server-Maschine

```
$ sudo systemctl restart nfs-kernel-server
```

Schritt 5: Firewall ändern (müssen wir noch testen)

Schritt 6: Verzeichnisse „mounten“

auf den Client-Maschinen

```
$ sudo mount 10.211.55.15:/home/parallels/nfs-share /nfs-server
```

host_ip

Pfad zum Verzeichnis auf Host

Pfad zum Verzeichnis auf client

Um es zu testen erstellen wir eine Datei mit *touch*:

auf den Client-Maschinen

```
$ sudo touch nfs-server/testdatei
```

Und wir lassen uns auf der Server-Maschine die Datei anzeigen:

auf der Server-Maschine

```
$ ls -lh nfs-share/
```

```
parallels@manager:~$ ls -lh nfs-share/
total 0
-rw-r--r-- 1 root root 0 Dec 28 17:23 testdatei
```

Normalerweise müsst man den mount-Befehl jedes Mal beim Neustarten der Maschine wieder eingeben, um das zu vermeiden schreiben wir was in der „fstab-Datei“ der Client-Maschinen rein.

auf den Client-Maschinen

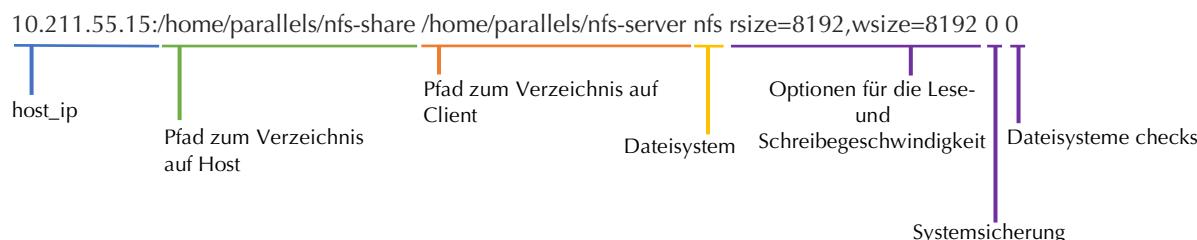
```
$ sudo nano /etc/fstab
```

So sollte bei euch der Editor ausschauen:

```
GNU nano 5.4                               /etc/fstab *
# /etc/fstab: static file system information.
#
# Use 'blkid' to print the universally unique identifier for a
# device; this may be used with UUID= as a more robust way to name devices
# that works even if disks are added and removed. See fstab(5).
#
# systemd generates mount units based on this file, see systemd.mount(5).
# Please run 'systemctl daemon-reload' after making changes here.
#
# <file system> <mount point> <type> <options>      <dump>  <pass>
# / was on /dev/sda2 during installation
UUID=a919f271-0600-427f-a9aa-ee5d90f36ff8 /          ext4    errors=remount-ro 0        1
# /boot/efi was on /dev/sda1 during installation
UUID=6F8D-2439 /boot/efi   vfat    umask=0077    0        1
# swap was on /dev/sda3 during installation
UUID=3a4c055a-4efe-4e72-88c8-81f84ef05aa0 none    swap    sw        0        0
/dev/sr0      /media/cdrom0  udf,iso9660 user,noauto  0        0

^G Help      ^O Write Out     ^W Where Is      ^K Cut       ^T Execute      ^C Location      M-U Undo
^X Exit      ^R Read File     ^Y Replace      ^U Paste      ^J Justify      ^G Go To Line    M-E Redo
                                         M-A Set Mark
                                         M-6 Copy
```

In der letzten Zeile schreiben wir:



```
GNU nano 5.4                               /etc/fstab *
# /etc/fstab: static file system information.
#
# Use 'blkid' to print the universally unique identifier for a
# device; this may be used with UUID= as a more robust way to name devices
# that works even if disks are added and removed. See fstab(5).
#
# systemd generates mount units based on this file, see systemd.mount(5).
# Please run 'systemctl daemon-reload' after making changes here.
#
# <file system> <mount point> <type> <options>      <dump>  <pass>
# / was on /dev/sda2 during installation
UUID=a919f271-0600-427f-a9aa-ee5d90f36ff8 /          ext4    errors=remount-ro 0        1
# /boot/efi was on /dev/sda1 during installation
UUID=6F8D-2439 /boot/efi   vfat    umask=0077    0        1
# swap was on /dev/sda3 during installation
UUID=3a4c055a-4efe-4e72-88c8-81f84ef05aa0 none    swap    sw        0        0
/dev/sr0      /media/cdrom0  udf,iso9660 user,noauto  0        0

10.211.55.15:/home/parallels/nfs-share /home/parallels/nfs-server nfs rsize=8192,wsize=8192 0 0
```

Jetzt müssen wir das Verzeichnis noch „unmounten“:

auf den Client-Maschinen

```
$ sudo umount nfs-server
```

Dann starten wir unsere Maschine neu und sehen, dass das NFS beim Start schon funktioniert.

Diesen Prozess müssen wir auf jeder Client Maschine ausführen, damit alle ein gemeinsamer Speicher haben.

```
parallels@worker1:~$ ls -lh nfs-server/
total 0
-rw-r--r-- 1 root root 0 Dec 28 17:23 testdatei
```

4. Portainer installieren und einrichten:

Um unseren Swarm besser zu visualisieren und zu überprüfen, laden wir Portainer herunter.

Falls bei euch Docker noch nicht aktiviert sein sollte, dann könnt ihr es mit diesem Befehl aktivieren:

```
$ sudo systemctl enable docker
jenny@debian:~$ sudo systemctl enable docker
Synchronizing state of docker.service with SysV service script with /lib/systemd
/systemd-sysv-install.
Executing: /lib/systemd/systemd-sysv-install enable docker
```

Schritt 1: Portainer YML-Manifest hherunterladen

```
$ curl -L https://downloads.portainer.io/ce2-16/portainer-agent-stack.yml
-o portainer-agent-stack.yml
```

Schritt 2: Mit dem YML-Manifest einen Docker Stack starten

```
$ docker stack deploy -c portainer-agent-stack.yml portainer
```

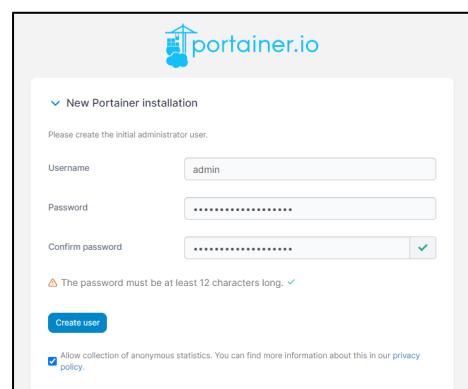
Wir können überprüfen, ob die Portainer Server und Agent Container gestartet sind, indem wir docker ps ausführen

```
$ docker service ls
parallels@manager:~$ docker service ls
ID          NAME      MODE      REPLICAS  IMAGE
k74ytuqcfmn6  portainer_agent   global     4/1      portainer/agent:2.16.2
4asb981mqt8q  portainer_portainer  replicated 1/1      portainer/portainer-ce:2.16.2
                                         PORTS
                                         *:8000->8000/tcp, *:9000->9000/tcp, *:9443->9443/tcp
```

Schritt 3: Auf Webbrowser öffnen

Jetzt, wo die Installation abgeschlossen ist, können wir uns in unseren Portainer Server Instanz einloggen, indem wir einen Webbrower öffnen und die IP-Adresse mit dem zugehörigen Port eingeben: **http://localhost:9443**

Nachdem ihr euch ein Portainerkonto angelegt habt, seid ihr mit dem Download von Portainer fertig.



5. Docker Swarm einrichten

Nachdem wir Portainer installiert und eingerichtet haben, können wir einen Swarm erstellen. Man muss sicherstellen, dass der Docker-Engine-Daemon auf dem Host-Rechner gestartet ist.

Schritt 1: Manager Node einrichten

Bei Docker Swarm muss man immer den Manager Node als ersten einrichten, damit die anderen Nodes diesen Swarm beitreten können. Deshalb wählen wir eine Maschine aus, und die wird dann als Swarm-Manager fungieren. Der Befehl dazu lautet:

```
$ docker swarm init --advertise-addr 10.211.55.16
```

Mit dem Befehl `--advertise-addr` wird der Manager Node so konfiguriert, dass er seine Adresse als 10.211.55.16 veröffentlicht. Die anderen Nodes im Swarm müssen in der Lage sein, unter dieser IP-Adresse auf den Manager zuzugreifen.

```
parallels@manager:~$ docker swarm init --advertise-addr 10.211.55.16
Swarm initialized: current node (xm8nsz18xhxpnd8q0khyhld36) is now a manager.

To add a worker to this swarm, run the following command:

  docker swarm join --token SWMTKN-1-5igcsh7dlq8m0ci59980p6d4fbwyc6r6mfq1x2rhjiy8t4tawi-f2cs9fplljaz823af53acsz4o 10.211.55.16:2377

To add a manager to this swarm, run 'docker swarm join-token manager' and follow the instructions.
```

Die Ausgabe enthält die Befehle zum Beitritt neuer Nodes zum Swarm. Die Nodes werden je nach dem Wert des `--token`-Flags als Manager oder Worker beitreten.

Schritt 2: Ausführung des Befehls `docker node ls`, um Informationen über die Nodes anzuzeigen:

```
$ docker node ls
parallels@manager:~$ docker node ls
ID           HOSTNAME   STATUS    AVAILABILITY  MANAGER STATUS   ENGINE VERSION
xm8nsz18xhxpnd8q0khyhld36 *   manager   Ready     Active        Leader       20.10.22
```

Der * neben der Node-ID zeigt an, dass wir derzeit mit diesem Knoten verbunden sind.

Schritt 3: Worker Nodes einrichten

Sobald ein Swarm mit einem Manager Node erstellt wurde, können wir die Worker Nodes hinzufügen. In unserem Fall sind es drei Worker.

Zunächst wird der Befehl ausgeführt, den die Ausgabe von „`docker swarm init`“ vom Manager Node gegeben wurde, um einen Worker Node zu erstellen, der mit dem bestehenden Swarm verbunden wird:

```
parallels@worker1:~$ docker swarm join --token SWMTKN-1-5igcsh7dlq8m0ci59980p6d4fbwyc6r6mfq1x2rhjiy8t4tawi-f2cs9fplljaz823af53acsz4o 10.211.55.16:2377
This node joined a swarm as a worker.
```

```
parallels@worker2:~$ docker swarm join --token SWMTKN-1-5igcsh7dlq8m0ci59980p6d4fbwyc6r6mfq1x2rhjiy8t4tawi-f2cs9fplljaz823af53acsz4o 10.211.55.16:2377
This node joined a swarm as a worker.
```

```
parallels@worker3:~$ docker swarm join --token SWMTKN-1-5igcsh7dlq8m0ci59980p6d4fbwyc6r6mfq1x2rhjiy8t4tawi-f2cs9fplljaz823af53acsz4o 10.211.55.16:2377
This node joined a swarm as a worker.
```

Wenn man den Befehl nicht zur Verfügung hat, dann kann folgender Befehl auf einem Manager Node ausgeführt werden, um den Join-Befehl für einen Worker abzurufen:

```
parallels@manager:~$ docker swarm join-token worker
To add a worker to this swarm, run the following command:

    docker swarm join --token SWMTKN-1-5igcsh7dlq8m0ci59980p6d4fbwyc6r6mfq1x2rhjiy8t4tawi-f2cs9fplljaz823af53acsz4o 10.211.55.16:2377
```

Schritt 3: Nodes auflisten

Wir gehen auf dem Manager Node und führen dort den Befehl „docker node ls“ aus, um die Worker Nodes anzuzeigen:

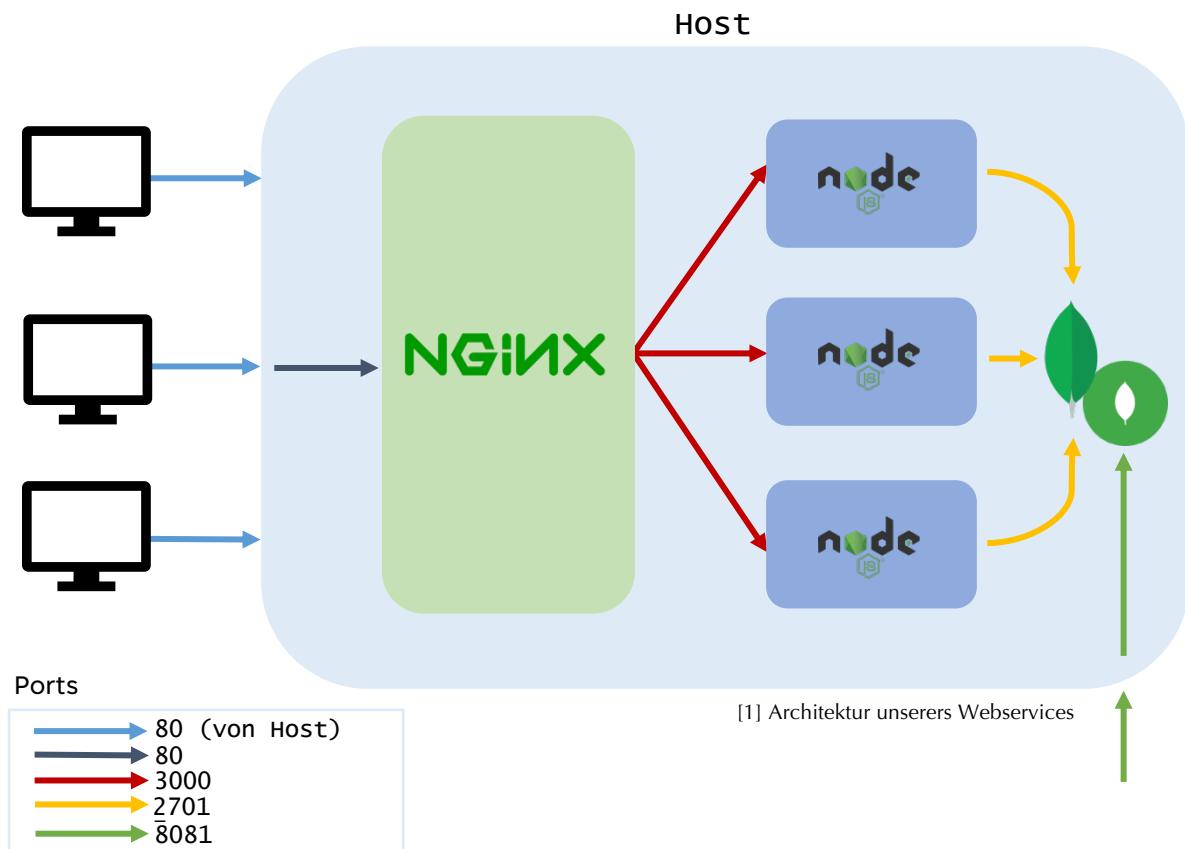
```
$ docker node ls
```

ID	HOSTNAME	STATUS	AVAILABILITY	MANAGER STATUS	ENGINE VERSION
xm8nsz18xhpnd8q0khyhld36 *	manager	Ready	Active	Leader	20.10.22
8qjrx1vy1w4pmf31pb7j2s88e	worker1	Down	Active		20.10.22
wkj65dszm1bfv3gccnwdgjwo9	worker1	Ready	Active		20.10.22
irftmyjruwzuc50t1w38ah9xt	worker2	Ready	Active		20.10.22
acnpqred814qn8rbhkns1333o	worker3	Ready	Active		20.10.22

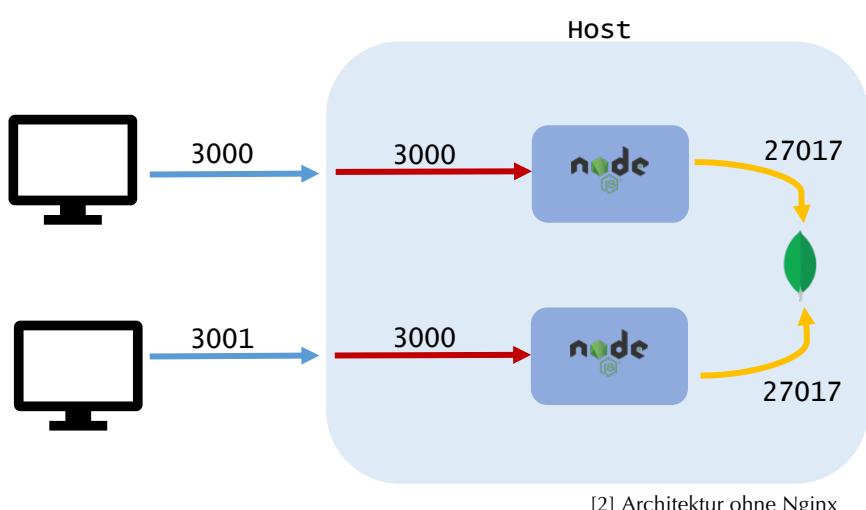
6. Webservice mit Node, Nginx und MongoDB

Unser Webservice besteht aus einem Server (Node), eine Datenbank (MongoDB) und einen Reverse Proxy (Nginx). Zusätzlich haben wir Mongo-Express benutzt, damit unsere Datenbank eine Benutzeroberfläche hat, welche uns später das Testen erleichtert. Die Anwendung soll einen einfachen Blog-Post darstellen, welches aus einem Titel (title) und einen Körper (body) besteht. Damit man ein Blog hochladen kann, muss man sich zuerst einen User anlegen, mit Username und Password. Danach kann man sich einloggen und hat 5min Zeit, um den Blog zu posten bzw. um die Aktionen durchzuführen. Um den Server mitzuteilen, dass wir gerade eingeloggt sind, haben wir Sessions benutzt. Wir haben uns die Pakete „express-session“ und „connect-mongo“ auf unseren Webservice heruntergeladen. Wenn der User sich einloggt, startet der Server eine Session und speichert die auf der Mongo-Datenbank ab. Zusammen mit der Session wird ein Cookie erstellt, wo die Session ID enthalten ist. Dieses Cookie wird, von dem Client, bei jeder Anfrage an den Server geschickt, und somit können Sie miteinander reden, natürlich so lange bis der Cookie existiert. Bei unserer Anwendung löschen sich die Cookies nach 5min, daher hat man 5min Zeit, um die verschiedenen Aktionen auf unseren Blog auszuführen.

Die Architektur von unserem Webservice kann man aus der Abbildung entnehmen:



Wir schicken eine Anfrage an unseren Node Server über dem Port 80 von unserem Host-Gerät. Dieser Port ist dann auf dem Port 80, vom Nginx Container, gemappt. Dieser leitet dann die Anfrage weiter an den Server, also den Node Container, welcher von Port 3000 aus kommuniziert. Und unser Server kann die Daten auf dem Mongo Container, Port 27017, speichern oder lesen. Die Mongo Datenbank kann im normalfall nur von dem Server aus erreicht werden aber bei unserem Service haben wir noch eine GUI für die Datenbank implementiert, welche auch von aussen erreichbar ist. Der Nginx Container ist unser Load Balancer. Ohne diesen könnte unsere Anwendung nicht mit einer erhöhten Belastung umgehen. Zum Beispiel wir haben zwei Clients und fügen deshalb einen weiteren Node Container hinzu damit es mit der Belastung klarkommt. Dieser neue Container muss dann auf einen anderen Port, von dem Host-Gerät, gemappt



werden, weil der Port 3000, von dem Host-Gerät, schon für die erste Instanz verwendet wird. Und wenn wir uns jetzt vorstellen wir haben 50 von den Node Container, also müssten wir dementsprechend auch 50 Ports, auf dem Host-Gerät, öffnen. Wie man schon merkt, ist es uneffizient also implementiert man einen Load Balancer dazwischen in unserem Fall Nginx. Dieser Nginx Container hört auf Port 80 und ist auf einem Port (bei uns 80), von dem Host-Gerät, gemappt. So können mehrere Clients, auf dem selben Port zugreifen und mit dem Server kommunizieren.

7. Test

Zunächst mal starten wir alle 4 VM und gehen bei dem Swarm-Manager ins Terminal rein. Dann wechseln wir in dem Ordner, wo alle Dateien von der Anwendung enthalten sind, und geben den Befehl `./start-app.sh` eingeben:

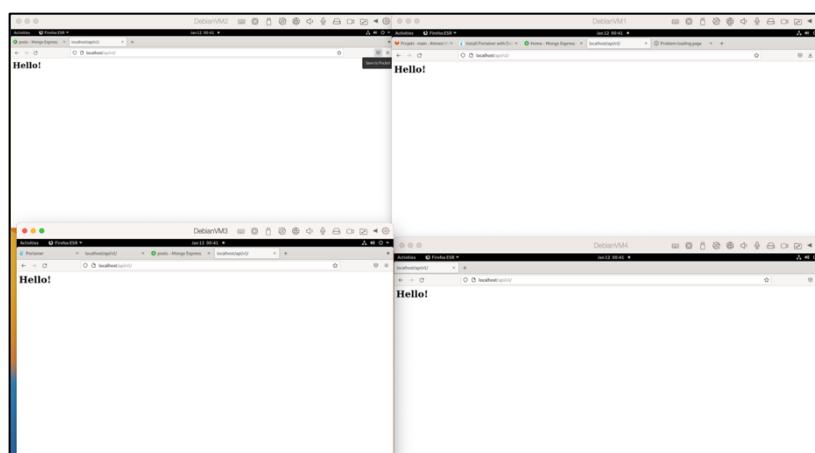
```
$ ./start-app.sh
```

Danach solltet ihr euren GitLab-Username und Password eingeben. Es sollte die folgende Ausgabe auf dem Bildschirm erscheinen:

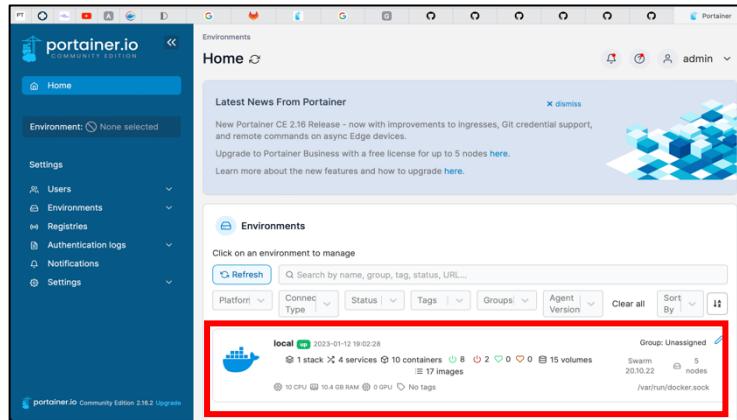
```
parallels@manager:~/docker-registry-main-Projekt/Projekt$ ./start-app.sh
Enter mygit Username: aa10413
Enter mygit Password: mkdir: cannot create directory 'home/node-app/mongo-data':
No such file or directory
WARNING! Using --password via the CLI is insecure. Use --password-stdin.
WARNING! Your password will be stored unencrypted in /home/parallels/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store

Login Succeeded
Creating network myapp_default
Creating service myapp_nginx
Creating service myapp_node
Creating service myapp_mongo
Creating service myapp_mongo-express
```

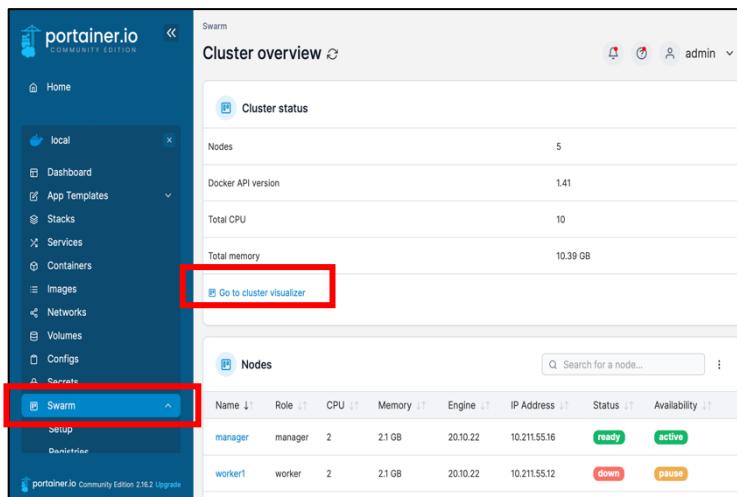
Als nächstes öffnet ihr euren Webbrowser und tippt ***localhost:80/api/v1*** ein. Das könnt ihr bei allen 4 VM machen. Die Webseite sollte bei allen 4 Maschinen gleich aussehen. Es sollte folgendes erscheinen:



Des Weiteren öffnet ihr die Portainer-Seite auf euren Webbrowser, mit **localhost:9443**, loggt euch ein und wählt das Environments local aus.



Im Menu, links vom Bildschirm, solltet ihr dann ganz unten Swarm auswählen. Danach geht ihr auf „Go to cluster visualizer“.



Wenn es keine Fehler gab, dann solltet ihr die Swarmnodes, auf der Seite, sehen. Zusätzlich können wir sehen das die 4 myapp_node Container auf dem Cluster verteilt sind. (*Ignorieren Sie bitte den drowned Worker1, es ist irgendwas vorher schiefgelaufen und deswegen zeigt er den Worker1 zweimal an*)



Ein weiterer Test ist, wenn man den folgenden Befehl „**docker logs mynode_app -f**“ auf der Host-Maschine eingeben und die Ausgabe betrachten:

```
parallels@manager:~/docker-registry-main-Projekt/Projekt$ docker service logs myapp_node -f
myapp_node.1.ybazicb3xji0@worker1    | listening on port 3000
myapp_node.1.ybazicb3xji0@worker1    | Succesfully connected to DB
myapp_node.3.0eo2hlz7chr5@worker2   | listening on port 3000
myapp_node.3.0eo2hlz7chr5@worker2   | Succesfully connected to DB
myapp_node.4.0qbyggz69xx9@worker3  | listening on port 3000
myapp_node.4.0qbyggz69xx9@worker3  | Succesfully connected to DB
myapp_node.2.ke7kxejnd3rb@manager   | listening on port 3000
myapp_node.2.ke7kxejnd3rb@manager   | Succesfully connected to DB
```

Wie wir sehen, läuft der Server auf jedem Node in unserem Swarm und gibt uns die Ausgabe, welche wir auf unseren Server programmiert haben.

Wir können auch Users als Test anlegen, die dann auf der MongoDB gespeichert werden so dass wir uns von anderen Maschinen aus einloggen können. Hierfür verwenden wir die Applikation „Postman“, welches ein Werkzeug zum Testen von APIs ist. Wir schicken mit Hilfe von Postman eine POST-Request an unsere Anwendung. Dafür geben

The screenshot shows the Postman interface with the following details:

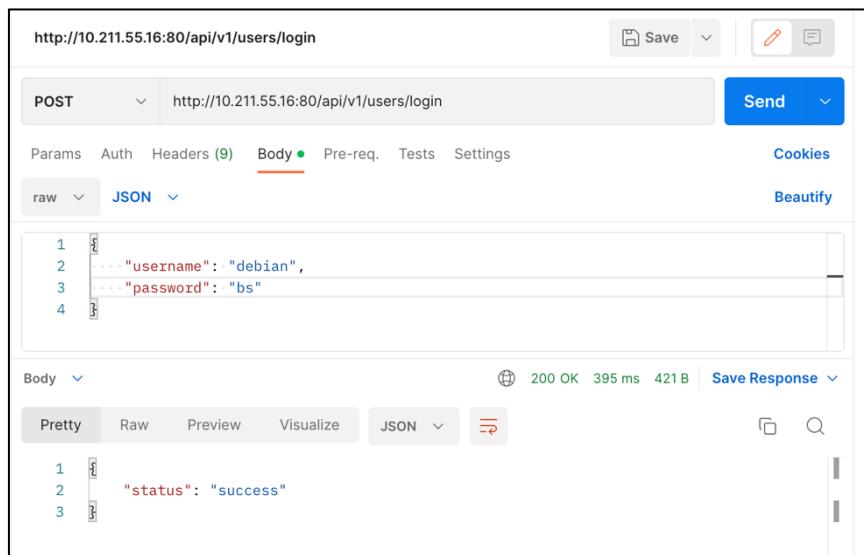
- URL:** http://10.211.55.16:80/api/v1/users/signup
- Method:** POST
- Body:** JSON (containing user signup data)
- Response Status:** 201 Created
- Response Body:** A JSON object indicating success with a generated user ID and timestamp.

```

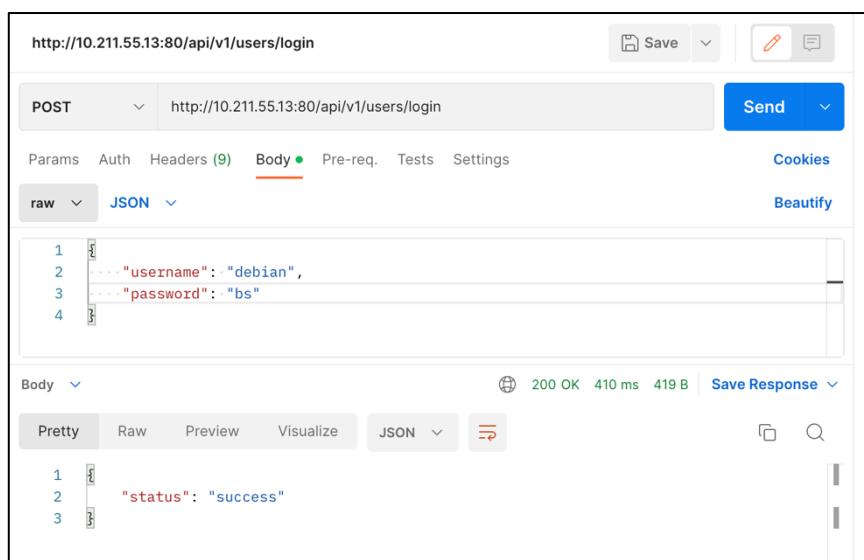
1
2   "status": "success",
3   "data": {
4     "user": {
5       "username": "debian",
6       "password": "$2a$12$J6CXCra/.jq102s5Tpfn7eg8rGh6.0s3dI64gLteaZ4WV.
9Y44GHq",
7       "_id": "63bf4a6f200b8d0f95959b66",
8       "__v": 0
9     }
10   }
11 }
```

die URL von unserer Anwendung ein: **10.211.55.16:80/api/v1/users/signup**.

Den User legen wir, für unseren Beispiel, am Host-Maschine an. Als Rückmeldung erhalten wir „Success“, somit wissen wir dass die Anfragen erfolgreich waren. Als nächstes versuchen wir uns auf der Host-Maschine einzuloggen. Dafür ändern wir das *signup* in der IP-Adresse in ein *login*. Und schicken die POST-Anfrage nochmal.



Wenn wir ein *Success* zurückbekommen, hat das Login funktioniert. Als nächstes versuchen wir es von einer anderen VM. Dafür ändern wir die URL bzw. die IP-Adresse oben in die eines Workers-Nodes. Bei uns: **10.211.55.13:80/api/v1/users/login**



Wie man vom Bild ablesen kann, hat das auch fehlerfrei funktioniert. Somit ist bewiesen das unsere Anwendung auf mehreren Maschinen laufen kann.

8. Quellen:

- -<https://docs.docker.com/engine/swarm/swarm-tutorial/add-nodes/>
 - -<https://docs.docker.com/engine/swarm/swarm-tutorial/create-swarm/>
 - <https://www.digitalocean.com/community/tutorials/how-to-set-up-an-nfs-mount-on-debian-11>
 - <https://www.howtoforge.com/tutorial/install-nfs-server-and-client-on-debian/>
 - <https://www.digitalocean.com/community/tutorials/how-to-set-up-an-nfs-mount-on-debian-11>
 - <https://wiki.ubuntuusers.de/NFS/>
 - <https://www.mongodb.com/docs/manual/installation/>
 - <https://docs.npmjs.com/downloading-and-installing-node-js-and-npm>
 - <https://nginxproxymanager.com/setup/>
 - <https://www.npmjs.com/package/express>
 - <https://www.npmjs.com/package/express-session>
 - <https://www.npmjs.com/package/connect-mongodb-session>
 - <https://www.theserverside.com/blog/Coffee-Talk-Java-News-Stories-and-Opinions/Docker-Nginx-reverse-proxy-setup-example>
 - https://hub.docker.com/_/mongo-express
-
- Deckbild: <https://rocketloop.de/wp-content/uploads/2021/05/clustering-machine-learning-comprehensive-guide.jpg>
 - Node.js Logo:
https://upload.wikimedia.org/wikipedia/commons/thumb/d/d9/Node.js_logo.svg/2560px-Node.js_logo.svg.png
 - Mongodb Logo:
https://upload.wikimedia.org/wikipedia/commons/thumb/9/93/MongoDB_Logo.svg/2560px-MongoDB_Logo.svg.png
 - Nginx Logo:
https://upload.wikimedia.org/wikipedia/commons/thumb/c/c5/Nginx_logo.svg/1280px-Nginx_logo.svg.png
 - Mongo-Express: <https://raw.githubusercontent.com/docker-library/docs/b9077663f53d2a5f3ce3ce52c9249d4b0c684fd9/mongo-express/logo.png>