A close-up of water droplets

Description automatically generated with low confidence**Betriebsysteme 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***

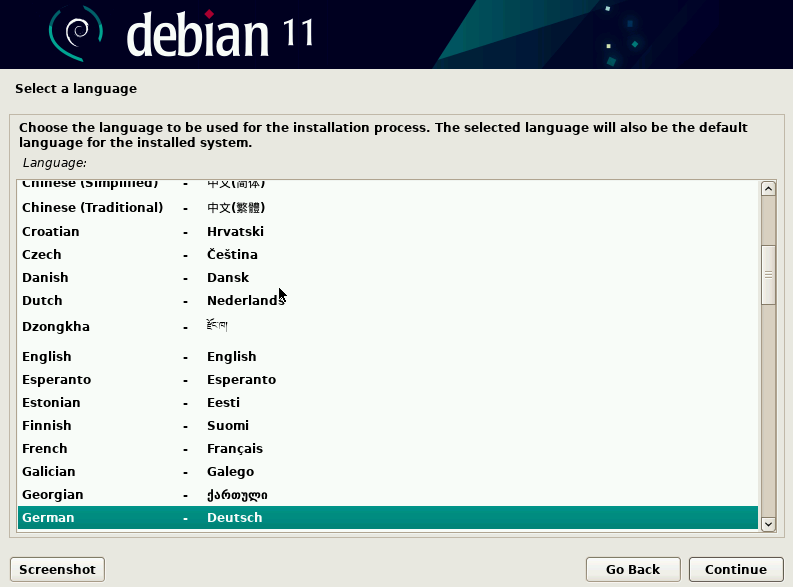
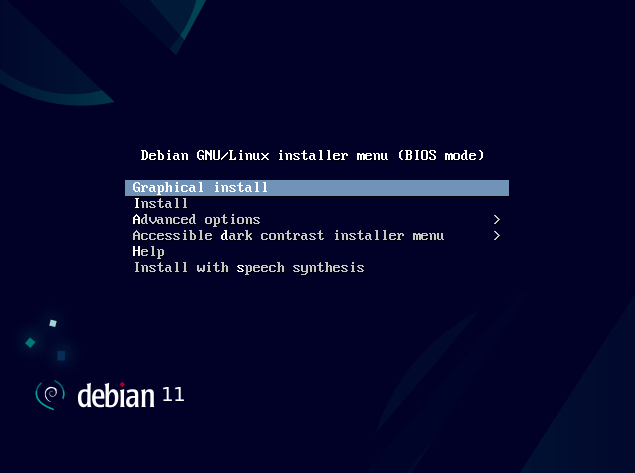
***4. Portainer***

***5. Docker Swarm einrichten***

***6. Webservice mit Node, Nginx und MongoDB***

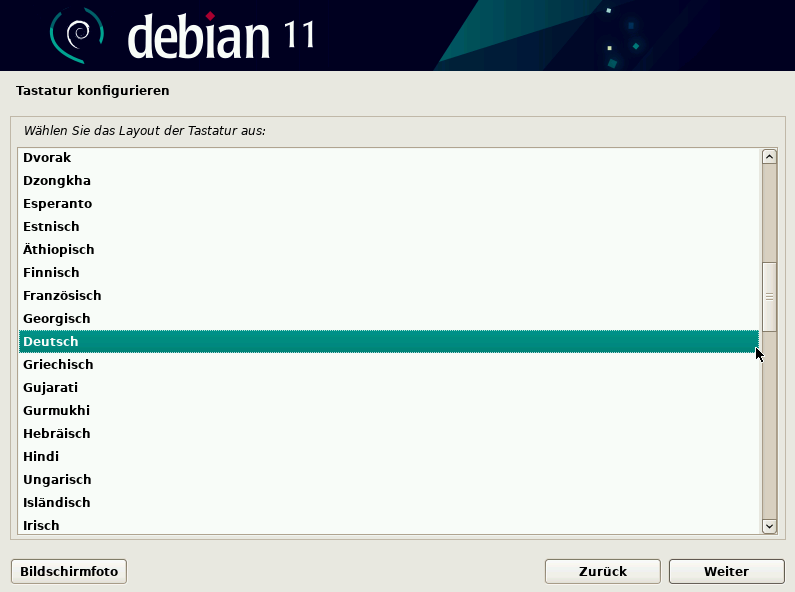
***7. Test***

***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**

**Ein Bild, das Text enthält.

Automatisch generierte Beschreibung**

**4. Schritt: Die Tastaturbelegung auswählen**

**3. Schritt: Das Land auswählen und bestätigen**

**Ein Bild, das Text enthält.

Automatisch generierte BeschreibungEin Bild, das Text enthält.

Automatisch generierte Beschreibung**

**6. Schritt: Ein Domain-Name vergeben**

**5. Schritt: Einrichtung des Netzwerks und ein Rechnername vergeben**

**Ein Bild, das Text enthält.

Automatisch generierte BeschreibungEin Bild, das Text enthält.

Automatisch generierte Beschreibung**

**8. Schritt: Name des ersten Benutzers eingeben**

**7. Schritt: Ein sicheres Kennwort für den Benutzer eingeben**

**Ein Bild, das Text enthält.

Automatisch generierte BeschreibungEin Bild, das Text enthält.

Automatisch generierte Beschreibung**

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

**10. Schritt: Ein sicheres Passwort für den Benutzer eingeben**

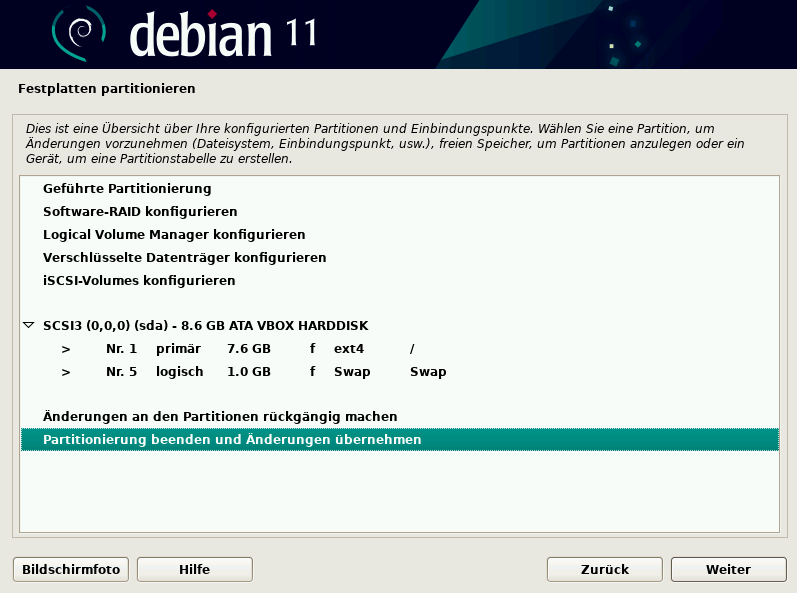
**Ein Bild, das Text enthält.

Automatisch generierte BeschreibungEin Bild, das Text enthält.

Automatisch generierte Beschreibung**

**11. Schritt: Die vollständige Festplatte für die Partitionierung auswählen**

**12. Schritt: Die aufgeführte Auswahl mit Weiter bestätigen**

**Ein Bild, das Text enthält.

Automatisch generierte Beschreibung**

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

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

**Ein Bild, das Text enthält.

Automatisch generierte Beschreibung**

**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**

**Ein Bild, das Text enthält.

Automatisch generierte Beschreibung**

**Ein Bild, das Text enthält.

Automatisch generierte Beschreibung**

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

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

**Ein Bild, das Text enthält.

Automatisch generierte BeschreibungEin Bild, das Text enthält.

Automatisch generierte Beschreibung**

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

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

**Ein Bild, das Text enthält.

Automatisch generierte BeschreibungEin Bild, das Text enthält.

Automatisch generierte Beschreibung**

**Ein Bild, das Text enthält.

Automatisch generierte BeschreibungEin Bild, das Text enthält.

Automatisch generierte Beschreibung**

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

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

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

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

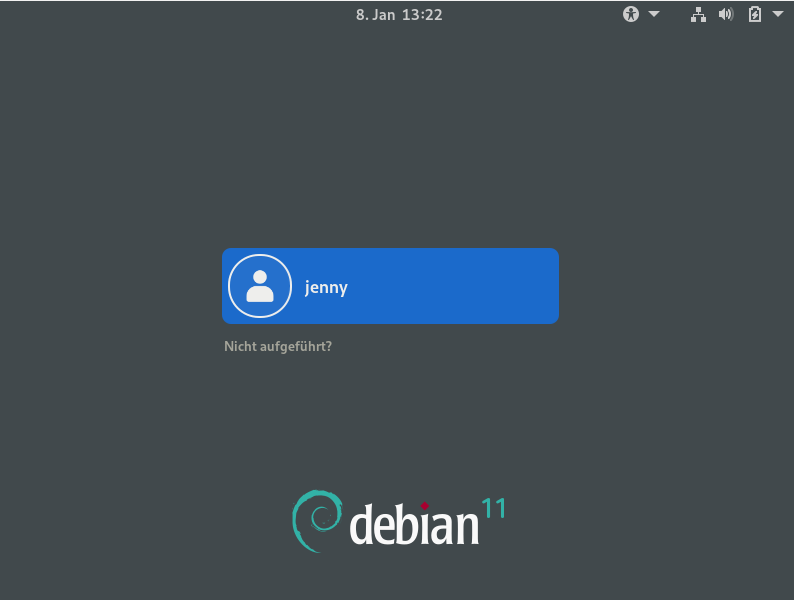
**Ein Bild, das Text enthält.

Automatisch generierte BeschreibungEin Bild, das Text enthält.

Automatisch generierte Beschreibung**

**25. Schritt: Dienste und Dateien werden geprüft und gestartet**

**24. Schritt: Debian 11 zum ersten Mal booten**

****

**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*

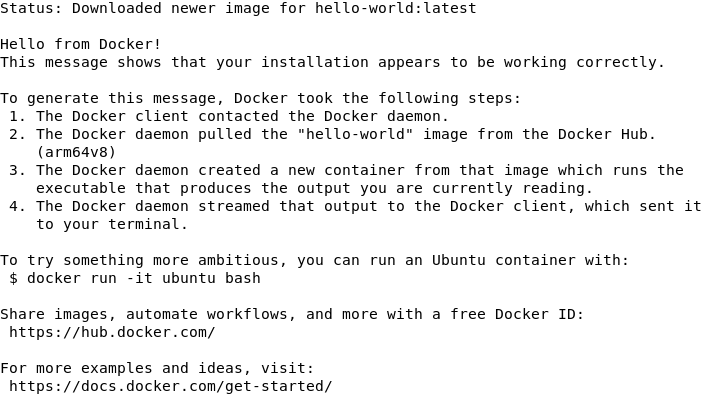
**Schritt 2:** die neuste Version der Docker Engine herunterladen

$ *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:

**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*

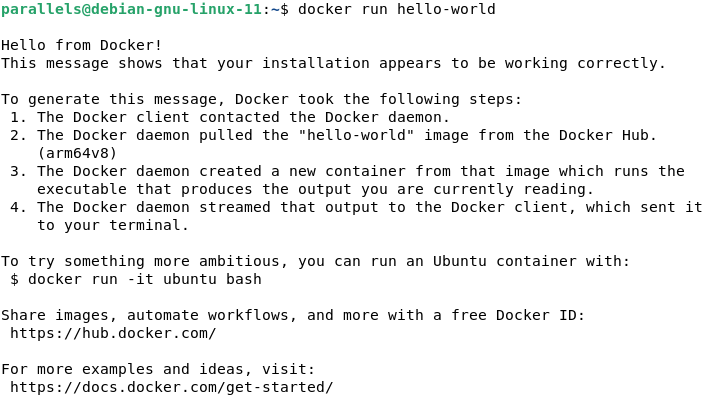
**Schritt 2:** User in der Docker Gruppe hinzufügen

$ *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.

**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*

Graphical user interface, text

Description automatically generated

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

Wir ergänzen ganz unten:

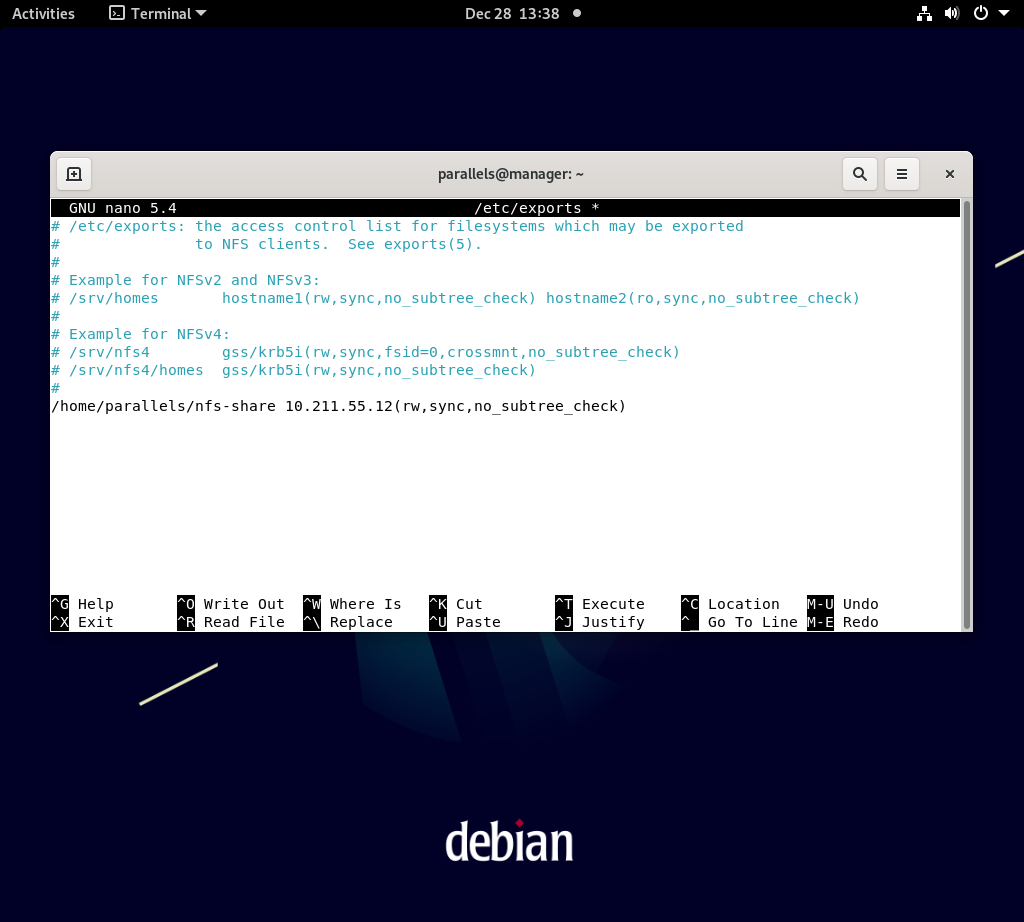
*/home/parallels/nfs-share 10.211.55.13(rw,sync,no\_root\_squash,no\_subtree\_check)*

Shareoptionen

Client\_ip

Pfad zum Verzeichnis auf Host

**Wichtig: kein Leerzeichen nach dem Komma machen!!!**

****

Der Pfad, der am Anfang steht */home/parallesl/nfs-share ist der Pfad zu dem geteilten Verzeichnis. Danach kommt die IP-Addresse 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-Addresse des anderen Clients hinten an der Zeile an.

**Graphical user interface, text

Description automatically generated**

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*

Graphical user interface, text, application, email

Description automatically generatedAusgaben bei unserem Fall schaut so aus:

**(Schritt 4:** NFS Serverneustarten ) noch testen

auf der Server-Maschine

$ *sudo systemctl restart nfs-kernel-server*

**Schritt 5:** Firewall ändern

(mussen wir noch testen)

**Schritt 6:** Verzeichnisse *„mounten“*

auf den Client-Maschinen

Pfad zum Verzeichnis auf client

Pfad zum Verzeichnis auf Host

host\_ip

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

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:

Graphical user interface, text, application

Description automatically generatedauf der Server-Maschin

$ *ls -lh nfs-share/*

Normalerweise müss 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*

Graphical user interface

Description automatically generatedSo sollte bei euch der Editor ausschauen:

In der letzten Zeile schreiben wir:

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

host\_ip

Dateisysteme checks

Systemsicherung

Optionen für die Lese- und Schreibegeschwindigkeit

Dateisystem

Pfad zum Verzeichnis auf Host

Pfad zum Verzeichnis auf Client

Graphical user interface

Description automatically generated

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.

Graphical user interface, text, application

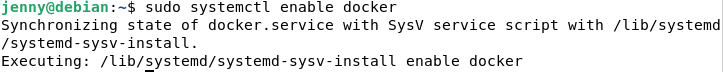
Description automatically generatedDiesen Prozess müssen wir auf jeder Client Maschine ausführen, damit alle ein gemeinsamer Speicher haben.

**4. Portainer installieren und einrichten:**

Um unseren Swarm besser zu visualizieren 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

****

**Schritt 1:** Portainer YML-Manifest herrunterladen

$ 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

**Graphical user interface, text, application

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

$ docker service ls

**Graphical user interface, text, application

Description automatically generatedSchritt 3:** Auf Webbrowser öffnen

Jetzt, wo die Installation abgeschlossen ist, können wir uns in unseren Portainer Server Instanz einloggen, indem wir einen Webbrowser 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 müss 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

Graphical user interface, text

Description automatically generatedMit 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.

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.

Graphical user interface, text

Description automatically generated**Schritt 2:** Ausführung des Befehls *docker node ls*, um Informationen über die Nodes anzuzeigen:

$ docker node ls

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.

**Graphical user interface, text, application

Description automatically generated**Graphical user interface, text

Description automatically generatedZunächst 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:

Graphical user interface, text

Description automatically generated

Graphical user interface, text, application

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

**Schritt 3:** Nodes auflisten

$ docker node ls

Table

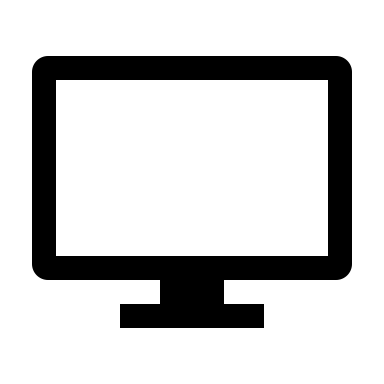
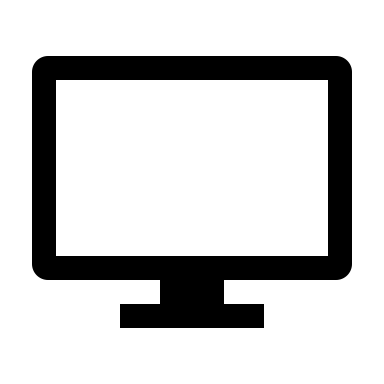
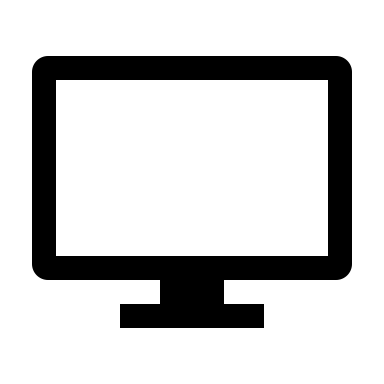
Description automatically generatedWir gehen auf dem Manager Node und führen dort den Befehl „docker node ls“ aus, um die Worker Nodes anzuzeigen:

**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 Datanbank eine Benutzeröberfläche 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, müss man sich zuerst einen User anlegen, mit Username und Password. Danch 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 Packete *„express-session“* und *„connect-mongo“* auf unseren Webservice herruntergeladen. 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:

[1] Architektur unserers Webservices



8081

3000

80 (von Host)

80

27017

Ports

Host

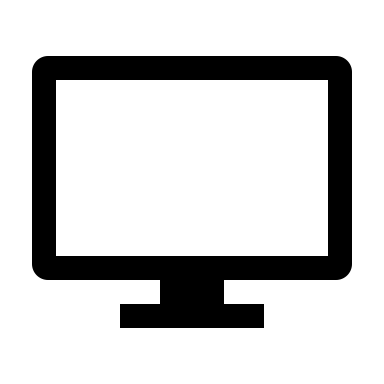
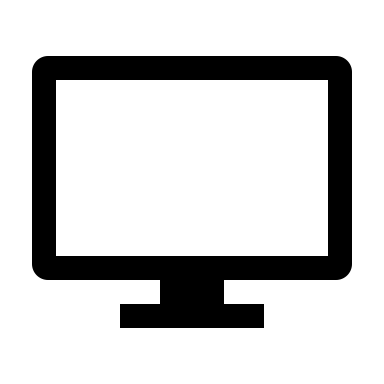
Wir schicken eine Anfrage an unseren Node Server über dem Port 80 von unserem Host-Gerät. Dieser Port ist dann auf dem Port 80, vom Nginx Container, gemappt. Dieser leitet dann die Anfrage weiter an den Server, also den Node Container, welcher von Port 3000 aus kommuniziert. Und unser Server kann die 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 neuer Container müss 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 uneffiezient 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 selbem Port zugreifen und mit dem Server kommunizieren.

[2] Architektur ohne Nginx



3000

3000



3000

3001

27017

27017

Host

**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 Andwendung enthalten sind, und geben den Befehl *./start-app.sh* eingeben:

$ ./start-app.sh

Text

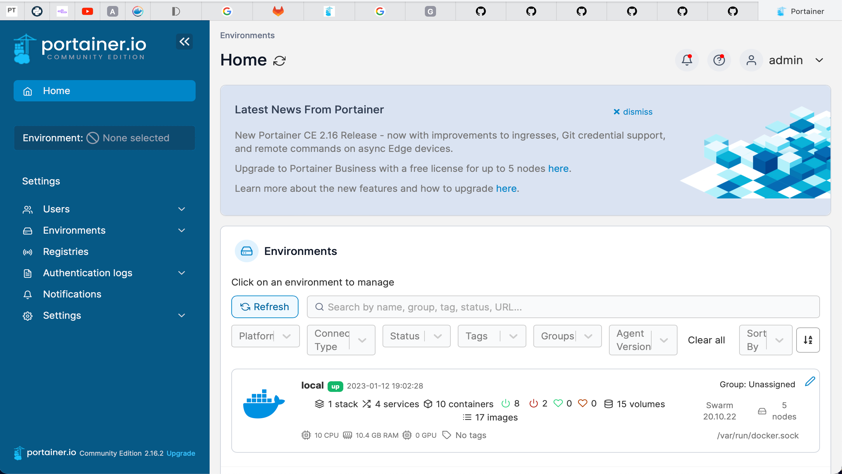
Description automatically generatedDanach solltet ihr euren GitLab-Username und Password eingeben. Es sollte die folgende Ausgabe auf dem Bildschirm erscheinen:

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 auschauen.

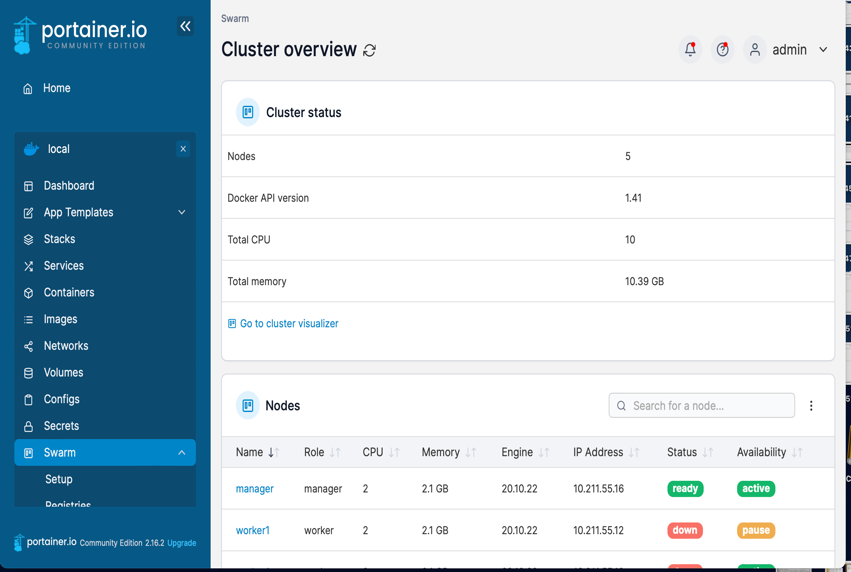
Graphical user interface, application, Word

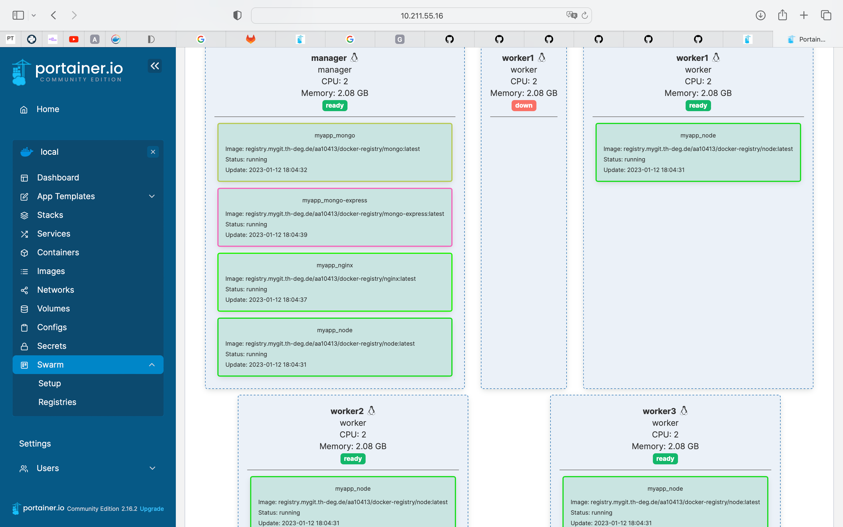
Description automatically generatedEs sollte folgendes erscheinen:

Des Weiteren öffnet ihr die Portainer-Seite auf euren Webbrowser, mit ***localhost:9443***, loggt euch ein und wählt das Enviroments 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 Conainter auf dem Cluster verteilt sind. *(Ignorieren Sie bitte den drowned Worker1, es ist irgendwas vorher schiefgelaufen und deswegen zeigt er den Worker1 zweimal an)*

Graphical user interface, text, application

Description automatically generatedEin weiterer Test ist, wenn man den folgenden Befehl ***„docker logs mynode\_app -f“*** auf der Host-Maschine eingeben und die Ausgabe betrachten:

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

Graphical user interface, text, application, email

Description automatically generatedWir könner auch Users als Test anlegen, die dann auf der MongoDB gespeichert werden soadass wir uns von anderen Maschinen aus einloggen können. Hier für 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

die URL von unserer Andwendung 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 nächtes versuchen wir uns auf der Host-Maschine einzuloggen. Dafür ändern wir das *signup* in der IP-Addresse in ein login. Und schicken die POST-Anfrage nochmal.

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, text, application, email

Description automatically generatedWenn 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 Andwendung auf mehrern 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- ondebian-11](https://www.digitalocean.com/community/tutorials/how-to-set-up-an-nfs-mount-%20ondebian-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>