Dokumentation

Abschlussprojekt Wintersemester 2019/20
Web, Protokolle und Service (DT5 -WPS)

Thema:

Vagrant virtuelle Maschine und Docker Container:
Installation und Konfiguration

Fachhochschule Dortmund
University of Applied Sciences and Arts

Name:	Jean-Michel Ekra
Projektbetreuer:	Prof. Ulf Niemeyer
Durchführungszeitraum:	15.01.2020 bis 07.02.2020

Einleitung: Installation der VirtualBox und Vagrant auf einem Privatrechner

Vagrant: Erstellen einer Basis-Virtuellen Maschine

- Debian Stretch
- Deutsch Lokalisierung
- GUI/ X11-Desktop
- Bonus: Bereinigung mit möglichst geringem Plattenplatzbedarf der VM

Vagrant: Erstellen einer abgeleiteten virtuellen Maschine

- Docker Container
- Aktuelles Docker-Compose
- Wireshark
- Bonus: Nutzer Debian/Debian statt Vagrant/Vagrant

Docker: Erstellen eines Basis-Images:

- Debian 9:9

Bonus: Vorkehrung für eine einfache Bedienung bei Build des Images und

Start, Stop und Remove eines Containers sowie beim Aufruf einer Shel im laufenden Container

Docker: Erstellen dreier abgeleiteter Images

- Router mit Router-Funktionalität
- SSH-Client
- Bonus: Jdocker-compose.yml & Makefile zur einfachen Bedienung

und einheitlichen Parametrierung bei Build, Start, Stop & Remove eines Containers

Docker: Aufbau eines Demo-Netzwerks

- lineares Gesamtnetz aus 3 Teilnetzen
- 192.168.10.0/24 mit host1 (SSH-client), router1
- 192.168.20.0/24 mit Router1 und Router2
- 192.168.30.0/24 mit router2, host2 (SSH-Server) und host3

Irgendwie stimmen deine Titel nicht mit dem Inhaltsverzeichnis guck noch Alle Netzelemente (jeweils im eigenen Container) sollen sich Kontaktieren können (ping) und der Netzverkehr soll per Wireshark beobachtet werden können. Alle Netzelemente (jeweils im eigenen Container) sollen sich kontaktieren können (ping) und der Netzverkehr soll per Wireshark beobachtet werden können. Das gehört nicht ins Inhaltsverzeichnis. Warum steht das hier?

I. Einleitung: Installation von VirtualBox und Vagrant

VirtualBox ist ab der Version 4.0, ein unter der „GNU General License, Version2“ stehender Virtualisierungsprogramm, das für die Betriebssysteme.

- Windows,
- Mac OS X,
- Solaris und für
- verschiedene Linux-Distributionen (z. B. Debian, Ubuntu, SUSE und Fedora)

zum Download angeboten wird Das separat erhältliche "VirtualBox Extension Pack" erweitert das Hauptprogramm um ein nützliches Feature (z. B. um die Möglichkeit, die Konsole einer virtuellen Maschine (VM) über das VirtualBox Remote Desktop Protocol (VRDP) aufzurufen). Für das Extension Pack gilt die „Personal User and Evaluation License (PUEL)“, die neben dem rein persönlichen Gebrauch auch die Nutzung an Schulen und für Lehrkräfte abdeckt. (Stand: September 2017, siehe https://www.virtualbox.org/wiki/Downloads).

I.I. Installation auf einem Debian-Host (Debian 9)

Bei Debian ist VirtualBox nicht in den Paketquellen enthalten, es gibt jedoch die Möglichkeit, eine zusätzliche Quelle in der Datei /etc./apt/sources.list einzutragen: deb http://download.virtualbox.org/virtualbox/debian Stretch Contrib ist das die Quelle dann muss es in Klammern mit Datum

Zusätzlich sollte noch der erforderliche Schlüssel für dieses Repository installiert werden, um Warnmeldungen bei der Installation mit apt-get bzw. aptitude zu vermeiden: wget -q https://www.virtualbox.org/download/oracle_vbox_2016.asc -O- | apt-key add – . Nach der Installation des Schlüssels sollten die Update-Befehle

- sudo apt-get update
- sudo apt-get upgrade

keine Fehlermeldungen mehr bringen. Über den Befehl

- sudo apt-get install virtualbox-5.1

wird die Virtualbox installiert. Über diese Methode ist sichergestellt, dass bei den regelmäßig auszuführenden Updates immer die neueste Version von VirtualBox installiert wird.

I.2. Schritt 2 – Debian 9-Server mit Vagrant installieren

•Installation von Virtualbox, Vagrant und Git-for-Windows

• Anlegen eines Verzeichnisses als Home für die Vagrant-Experimente, z.B. D:\My_Data\vagrant

• Setzen der Umgebungsvariablen VAGRANT_HOME, z.B. auf D:\My_Data\vagrant\.vagrant.d

• Ggf. Anlegen einer ~/. bashrc Konfiguration bspw. mit alias ll='ls -al'

Voreinstellung ist ll='ls -' und
ls='ls -F –Color=Auto –show-Control-chars' (zum Abschalten der Einfügung --Color=no; alias liefert die Liste der Aliase; echo ~ → /c/Users/)

• Durchsehen des Box-Katalogs bei app.vagrantup.com/boxes/search → Debian/contrib-stretch64

• Verfügbarmachen der Box per

◦ Vagrant add box bzw. Vagrant add box --Name

◦ für Offline-Betrieb können die anderweitig heruntergeladenen Dateien der Box (debianVAGRANTSLASH-contrib-stretch64) in Vagrand.d\boxes abgelegt werden.

• Überprüfung der vorhandenen Boxen per vagrant box List

• Anlegen eines Verzeichnisses für die erste Vagrant-VM, bspw. in bash: ◦ cd/d/My_Data/vagrant

◦ mkdir test_debianContrib-stretch64 cd test_debianContrib-stretch64

• Anlegen des Vagrantfile per bspw. 'vim Vagrantfile'

```
Vagrant. Configure ("2") do |config|
 config.vm.box = "Debian/contrib-stretch64"
 config.vm. provider "VirtualBox" do |vb|
 vb. customize ["modifyvm “, id, "--cableconnected1", "on"]
 end
end
```

• vagrant up

◦ ggf. Firewall-Anfrage für VboxHeadless.exe beantworten

• Überprüfen des Status per

◦ VirtualBox-GUI

◦ vagrant Status bzw. vagrant global-status

• Anhalten (Neustart jeweils mit vagrant up)

Vagrant: Erstellen einer Basis-VM

II.I. **Provisioning von German Locales auf einer VM**

Schreibe hier auf für was das die Anleitung ist:
Starten der eben angelegten VM aus der VirtualBox-GUI und Einloggen per vagrant / vagrant

→ US-amerikanisches Keyboard und Datumsformat

• locale -a zeigt die vorhandenen Lokalisierungen an (nicht DE)

• Erzeugen der Lokalisierung per

echo "de_DE ISO-8859-1" >> /etc/locale.gen echo "de_DE.UTF-8 UTF-8">>/etc/locale.gen

echo "de_DE@euro ISO-8859-15" >> /etc/locale.gen sudo /usr/sbin/locale-gen

sudo localectl set-locale LANG=de_DE.UTF-8 (oder Auskommentieren der entsprechenden Zeilen in /etc/locale.gen)

• Überprüfen per locale -a locale

• Setzen des Keyboards per sudo localectl set-keyboard de

• Neustart des entsprechenden System-Deamons sudo service systemd-localed restart (funktioniert nicht, stattdessen Neustart)

• Setzen der Zeitzone per sudo timedatectl set-timezone Europe/Berlin

• Überprüfen mit sudo service systemd-timedated status

• Automatiserung dieser Handarbeit per vagrant-provisioning

II.II. **GUI / X11-Desktop**

• Vagrantfile:

Vagrant.configure("2") do |config| config.vm.box = "contrib-stretch-de-box" config.vm.provider "virtualbox" do |vb| vb.customize ["modifyvm", :id, "--cableconnected1", "on"] vb.linked_clone=true vb.memory = 1536 vb.cpus = 1

end

end

• Erproben mit vagrant up / halt Zyklus

• Einschalten des GUIs: Ergänzen von vb.gui = true

• vagrant up –provision macht die Ergänzung wirksam und öffnet unmittelbar ein eigenes Fenster für die VM

• Für den X11-Desktop müssen weitere Pakete installiert werden (bei debian via apt)

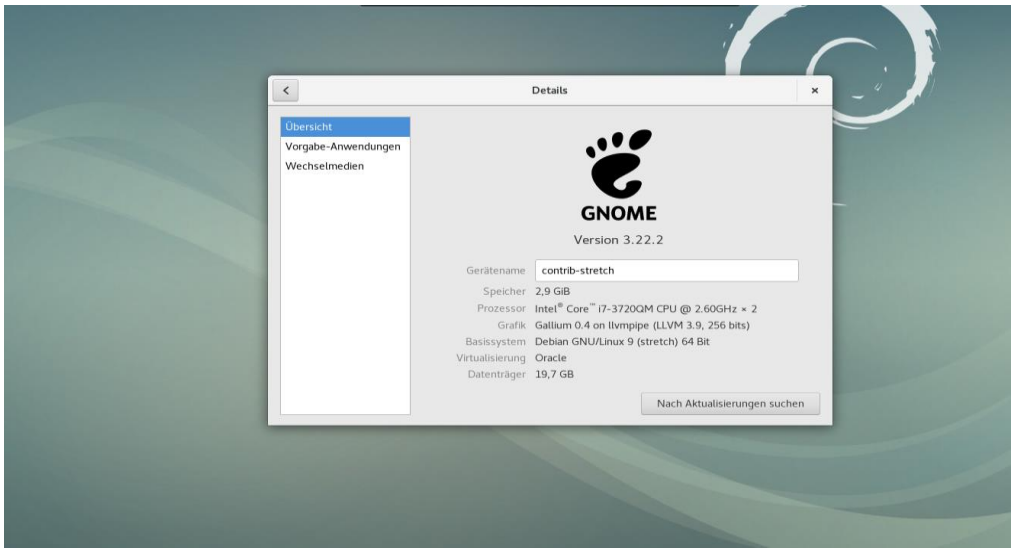
Außerdem besteht die Möglichkeit durch Gnome-core eine GUI/X11-Desktop Maschine zu installieren.

Upgrading Debian 10 Packages:

```
$ sudo apt update
$ sudo apt upgrade
$ sudo reboot
```

Installing GNOME Classic Desktop Environment:

```
$ sudo tasksel install desktop gnome-desktop
```



II. Vagrant: Erstellen einer abgeleitet Virtuelle Maschine

- Erstellen einer Docker-VM per Vagrant:

Docker ist ein Werkzeug, das die Erstellung, Bereitstellung und Ausführung von Anwendungen durch die Verwendung von Containern erleichtert. Mit Containern kann ein Entwickler eine Anwendung mit allen benötigten Teilen wie Bibliotheken und anderen Abhängigkeiten zusammenstellen und als ein Paket ausliefern. Auf diese Weise kann der Entwickler dank des Containers sicher sein, dass die Anwendung auf jedem anderen Linux-Rechner läuft, unabhängig von den angepassten Einstellungen, die dieser Rechner möglicherweise hat und die sich von dem Rechner unterscheiden, der zum Schreiben und Testen des Codes verwendet wird.

Die folgende Vagrantfile legt eine Debian-VM mit der Docker Community Edition an. Es bestehen folgende Besonderheiten

- Basis ist die selbsterstellte Vagrant-Box contrib-stretch-de-X11-box.
- Probleme beim Verändern der GUI-Fenstergröße werden durch die Einstellung des Video RAMs auf 32MByte behoben: vb. customize [modifyvm, id, „--vram“, 32]
- Das Vagrant-Plugin: Vagrant-persistent-Storage wird verwendet, um eine eigene virtuelle Platte für die Docker-Daten anzulegen.
- Verwende Vagrant Plugin install Vagrant-persistent Storage. Damit überstehen diese Daten auch ein etwaiges Vagrant destroy. 18 Default des plugins ist die Anlage von LVM-Volumes. Hier ist aber die Verwendung einer einfachen, direkt zu mount-enden Platte gewünscht: config.persistent_storage.use_lvm = false.
- Docker wird auf die Verwendung des neuen Root-Directories /home/vagrant/docker/var_lib_docker durch das An- bzw. Ablegen von /etc./docker/daemon.json mit dem Inhalt

  {

  “data-root “: “/home/debian/docker/var_lib_docker “ eingestellt.

Nach dem Starten der VM kann die Einstellung der Platten überprüft werden mit df -Th. Überprüfen der Docker-Einstellungen per sudo docker info.

III.I. Vagrant: Aktuelles Docker-Compose

Docker Compose ermöglicht es Ihnen, mit ein paar einfachen Befehlen mehrere Container mit einzelnen Anwendungen oder Komponenten zu starten. Docker Compose kann auch verwendet werden, um Container automatisch zu verknüpfen und Container-Volumes einzurichten.

Wenn Sie eine lange Reihe von Docker-Run-befehlen ausführen, um mehrere Container aufzurufen, erspart Ihnen Docker Compose eine Menge Arbeit. Welche Arbeit genau?

```
sudo apt-get install python-pip
sudo pip install docker-compose
sudo mkdir hello-world
cd hello-world
sudo nano docker-compose.yml
```

```
version: '2'

services:

  compose-test:
    image: centos



sudo docker-compose up -d

sudo docker-compose ps
```

Um alle Container einer Anwendungsgruppe zu stoppen, verwenden Sie den Befehl:

```
sudo docker-compose stop
```

Nachdem die Container gestoppt wurden, können Sie alle Container, die mit Docker Compose gestartet wurden, mit folgendem Befehl entfernen:

```
sudo docker-compose rm
```

Um sowohl alle Container in einer Anwendungsgruppe zu stoppen als auch alle mit einem einzigen Befehl zu entfernen, verwenden Sie:

```
sudo docker-compose down
```

Um auch alle Data-Volumes zu entfernen, fügen Sie das Kennzeichen -v hinzu:

```
sudo docker-compose down -v
```

III.II. Vagrant: Wireshark Installation

Wireshark ist ein sogenannter "Sniffer", mit dem man Netzwerkpakete auf beliebigen Schnittstellen mitschneiden und danach bequem über eine grafische Oberfläche analysieren kann. Netzwerksniffer genießen leider teilweise einen schlechten Ruf, weil sie u.a. auch von Angreifern benutzt werden können, um Passwörter oder andere sensible Daten im Netzwerkverkehr zu erspähen. Sie sind aber auch aus dem Werkzeugschatz seriöser Netzwerk-Administratoren nicht wegzudenken, weil man mit ihnen Probleme in der Vernetzung aufspüren kann. Auch um die Funktion einer Software zu testen, z.B. wenn man wissen will, ob die eingeschaltete Verschlüsselung funktioniert oder wenn man "Nach-Hause-Telefonierern" auf die Schliche kommen will, ist ein Netzwerksniffer nahezu unverzichtbar.
Für Wireshark muss folgendes Paket installiert [1] werden:

```
sudo apt-get install wireshark
```

I. Docker: Erstellen eines Basis Images: Debian 9 Stretch

Das im offiziellen Ubuntu-Repository verfügbare Docker-Installationspaket ist möglicherweise nicht die neueste Version. Um sicherzugehen, dass ich die neueste Version erhalte, installiere ich Docker aus dem offiziellen Docker-Repository. Dazu fügen wir eine neue Paketquelle und den GPG-Schlüssel von Docker hinzu, um sicherzustellen, dass die Downloads gültig sind, woraufhin wir das Paket installieren.

Aktualisieren Sie zunächst Ihre vorhandene Paketliste:

$ sudo apt-get Update

Installieren Sie schließlich den Docker:

$ sudo apt-install docker-ce

Docker-Container werden aus Docker-Bildern erstellt. Standardmäßig bezieht Docker diese Bilder aus dem Docker Hub, einem Docker-Verzeichnis, das von Docker, der Firma hinter dem Docker-Projekt, verwaltet wird. Jeder kann seine Docker-Bilder auf dem Docker Hub hosten, so dass die meisten Applikationen und Linux-Distributionen, die Sie benötigen, über Bilder verfügen, die dort bereitgestellt werden.Um zu überprüfen, ob Sie auf Bilder vom Docker Hub zugreifen und sie herunterladen können, geben Sie Folgendes ein:

$ docker run hello-world

Sie können nach Bildern suchen, die auf dem Docker Hub verfügbar sind, indem Sie den docker -Befehl mit dem Unterbefehl search verwenden. Geben Sie beispielsweise für die Suche nach dem Ubuntu-Bild folgendes ein:

$ docker search ubuntu

Um die Bilder anzuzeigen, die auf Ihren Computer heruntergeladen wurden, geben Sie Folgendes ein:

$ docker images

Die Meldung sollte wie folgt aussehen:

```
REPOSITORY          TAG                 IMAGE ID            CREATED
SIZE
ubuntu              latest              113a43faa138        4 weeks ago
81.2MB
hello-world         latest              e38bc07ac18e        2 months
ago        1.85kB
```

Container können viel nützlicher sein als das, und sie können auch interaktiv sein. Schließlich sind sie ähnlich wie virtuelle Maschinen, nur ressourcenschonender.
Betrachten wir als Beispiel einen Container mit dem neuesten Bild von Ubuntu. Die Kombination der Schalter **-i** und **-t**ermöglicht Ihnen den interaktiven Shell-Zugriff auf den Container:
$ docker run -it ubuntu

Um die **active ones** anzuzeigen, verwenden Sie:

$ docker ps
Um alle Container — aktive und inaktive — anzuzeigen, starten Sie docker ps mit dem Schalter -a:
$ docker ps -a
Sie werden folgende Meldung sehen:

```
d9b100f2f636            ubuntu        "/bin/bash"      About an hour ago
Exited (0) 8 minutes ago

01c950718166            hello-world   "/hello"         About an hour ago
Exited (0) About an hour ago
```

Den zuletzt erstellten Container mit dem Schalter-l anzeigen:
$ docker ps -l

```
CONTAINER ID            IMAGE           COMMAND                  CREATED
STATUS                              PORTS                     NAMES

d9b100f2f636         ubuntu      "/bin/bash"          About an hour ago
Exited (0) 10 minutes ago                           sharp_volhard
```

III. Docker: Erstellen dreier abgeleiteter Images

IV.I. Docker: SSH-Client

Um SSH-Zugriff für einen RUN-Befehl im Dockerfile anzufordern, müssen Sie eine Verbindung mit dem Typ "ssh" definieren. Das bedeutet, dass ein Socket mit einem Nur-Lese-Zugriff auf den SSH-Agenten eingehängt wird, während dieser Prozess läuft. Dadurch wird auch die Umgebungsvariable SSH_AUTH_SOCK eingerichtet, damit Programme, die sich auf SSH verlassen, diesen Socket automatisch verwenden.

```
# Syntax=Docker/Docker File: experimentalerem alpine

# install ssh client and git

RUN apk add --no-cache openssh-client git

# download public key for github.com

RUN mkdir -p -m 0600 ~/.ssh && ssh-keyscan github.com >> ~/.ssh/known_hosts

# clone our private repositoryRUN --mount=type=ssh git clone git@github.com:myorg/myproject.git myproject
```

Auf der Seite des Docker-Clients müssen Sie festlegen, dass die SSH-Weiterleitung für diesen Build erlaubt ist, indem Sie das Flag--ssh verwenden.

docker build --ssh default.

Das Flag akzeptiert ein Schlüssel-Wert-Paar, das den Ort für den lokalen SSH-Agent-Socket oder die privaten Schlüssel definiert. Der Socket-Pfad kann leer gelassen werden, wenn Sie den Wert von default=$SSH_AUTH_SOCK verwenden wollen. Beachten Sie, dass Sie bei Verwendung der Standardkonfiguration Ihre Schlüssel zu Ihrem lokalen SSH-Agenten hinzufügen müssen und sie Ihren ~/.ssh/id_rsa-Schlüssel nicht automatisch verbinden werden. Sie können ssh-add -L lokal überprüfen, um zu sehen, ob die öffentlichen Schlüssel für den Agenten sichtbar sind.

Das Mount in der Docker-Datei kann auch einen "id"-Schlüssel verwenden, um verschiedene Server, die Teil desselben Build sind, zu trennen. Beispielsweise kann auf die verschiedenen Repositorys in Ihrer Docker-Datei mit unterschiedlichen Deploy-Schlüsseln zugegriffen werden. In diesem Fall würden Sie in der Dockerdatei einen "id"-Schlüssel verwenden:

…RUN --mount=type=ssh,id=projecta git clone projecta

…RUN --mount=type=ssh,id=projectb git clone projectb …

IV.II. Docker SSH-Server

Das folgende Dockerfile richtet einen SSHd-Dienst in einem Container ein, den Sie verwenden können, um eine Verbindung zu anderen Containervolumes herzustellen und diese zu inspizieren oder um schnellen Zugriff auf einen Testcontainer zu erhalten. Nehmen Sie die folgenden Ersetzungen vor:

```
FROM ubuntu:16.04

RUN apt-get update && apt-get install -y openssh-server
RUN mkdir /var/run/sshd
RUN echo 'root:THEPASSWORDYOUCREATED' | chpasswd
RUN sed -i 's/PermitRootLogin prohibit-password/PermitRootLogin yes/' /etc/ssh/sshd_config

# SSH login fix. Otherwise user is kicked off after login
RUN sed 's@session\s*required\s*pam_loginuid.so@session optional pam_loginuid.so@g' -i /etc/pam.d/sshd

ENV NOTVISIBLE "in users profile"
RUN echo "export VISIBLE=now" >> /etc/profile

EXPOSE 22
```

```
CMD ["/usr/sbin/sshd", "-D"]
```

Erstellen Sie das Bild mit:

```
$ docker build -t eg_sshd
```

Dann führen Sie es aus. Sie können dann den Docker-Port verwenden, um herauszufinden, auf welchen Host-Port der Port 22 des Containers abgebildet ist:

```
$ docker run -d -P --name test_sshd eg_sshd

$ docker port test_sshd 22



0.0.0.0:49154
```

Und jetzt können Sie ssh als root auf der IP-Adresse des Containers (Sie können es mit Docker Inspect finden) oder auf Port 49154 der Host-IP-Adresse des Docker-Daemons (ip-Adresse oder ifconfig kann Ihnen das mitteilen) oder localhost, wenn auf dem Host des Docker-Daemons:

```
$ ssh root@192.168.1.2 -p 49154

# or

$ ssh root@localhost -p 49154

# The password is ``screencast``.

root@f38c87f2a42d:/#
```

IV.Docker: Aufbau eines Demo-Netzwerks

Docker bietet Unterstützung für Netzwerk-Container durch die Verwendung von Netzwerktreibern. Standardmäßig stellt Docker zwei Netzwerktreiber für Sie bereit, den Bridge- und den Overlay-Treiber. Sie können auch ein Netzwerktreiber-Plugin schreiben, so dass Sie Ihre eigenen Treiber erstellen können, aber das ist eine fortgeschrittene Aufgabe. Jede Installation der Docker-Engine umfasst automatisch drei Standard-Netzwerke. Sie können diese auflisten:

```
$ docker network ls


NETWORK ID          NAME                DRIVER
18a2866682b8        none                null
c288470c46f6        host                host
7b369448dccb        bridge              bridge
```

```
$ docker run -itd --name=networktest ubuntu


74695c9cea6d9810718fddadc01a727a5dd3ce6a69d09752239736c030599741
```

Die Inspektion des Netzwerks ist eine einfache Möglichkeit, die IP-Adresse des Containers herauszufinden.

```
$ docker network inspect bridge
[
    {
        "Name": "bridge",
        "Id": "17e324f459648a9baaea32b248d3884da102dde19396c25b30ec800068ce6b10",
        "Created": "2017-06-22T20:27:43.826654485Z",
```

```
        "Scope": "local",
        "Driver": "bridge",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": null,
            "Config": [
                {
                    "Subnet": "192.168.10.0/24",
                    "Gateway": "192.168.20.0/24"
                }
            ]
        },
        "Internal": false,
        "Attachable": false,
        "Containers": {
            "602dbf1edc81813304b6cf0a647e65333dc6fe6ee6ed572dc0f686a3307c6a2c": {
                "Name": "alpine2",
                "EndpointID":
"03b6aafb7ca4d7e531e292901b43719c0e34cc7eef565b38a6bf84acf50f38cd",
                "MacAddress": "02:42:ac:11:00:03",
                "IPv4Address": "192.168.30.0/24",
                "IPv6Address": ""
            },
            "da33b7aa74b0bf3bda3ebd502d404320ca112a268aafe05b4851d1e3312ed168": {
                "Name": "alpine1",
```

```
            "EndpointID":
"46c044a645d6afc42ddd7857d19e9dcfb89ad790afb5c239a35ac0af5e8a5bc5",
            "MacAddress": "02:42:ac:11:00:02",
            "IPv4Address": "192.168.10.0/24",
            "IPv6Address": ""
        }
    },
    "Options": {
        "com.docker.network.bridge.default_bridge": "true",
        "com.docker.network.bridge.enable_icc": "true",
        "com.docker.network.bridge.enable_ip_masquerade": "true",
        "com.docker.network.bridge.host_binding_ipv4": "0.0.0.0",
        "com.docker.network.bridge.name": "docker0",
        "com.docker.network.driver.mtu": "1500"
    },
    "Labels": {}
  }
]
```

Listen Sie die Netzwerke von Docker auf:

```
$ docker network ls

NETWORK ID          NAME                DRIVER              SCOPE
e9261a8c9a19        alpine-net          bridge              local
17e324f45964        bridge              bridge              local
6ed54d316334        host                host                local
7092879f2cc8        none                null                local
```

Erstellen Sie Ihre vier Container. Beachten Sie die Netzwerkflaggen. Sie können sich während des Docker-Run-Befehls nur mit einem Netzwerk verbinden, daher müssen Sie danach Docker Network Connect verwenden, um alpine4 auch mit dem Brückennetzwerk zu verbinden.

```
$ docker run -dit --name alpine1 --network alpine-net alpine ash

$ docker run -dit --name alpine2 --network alpine-net alpine ash

$ docker run -dit --name alpine3 alpine ash

$ docker run -dit --name alpine4 --network alpine-net alpine ash

$ docker network connect bridge alpine4
```

Überprüfen Sie, ob alle Container laufen:

```
$ docker container ls


CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS               NAMES
156849ccd902        alpine              "ash"               41 seconds ago      Up 41 seconds                           alpine4
fa1340b8d83e        alpine              "ash"               51 seconds ago      Up 51 seconds                           alpine3
a535d969081e        alpine              "ash"               About a minute ago  Up About a minute                       alpine2
0a02c449a6e9        alpine              "ash"               About a minute ago  Up About a minute                       alpine1
```

V. Zusammenfassung

Die Open-Source-Software Vagrant hat sich, neben Docker oder klassischen virtuellen Maschinen, schnell zu einem Entwickler-Tool der Superlative entwickelt. Warum? Weil mit Vagrant die Zeit, die für das Aufsetzen der Entwicklungsumgebung benötigt wird, drastisch verkürzt werden kann. Früher hat sich jeder Entwickler lokal alle Tools installiert, die er braucht. Wer dieses Konzept ins Heute überträgt, merkt schnell:

Dieser Workflow ist nicht mehr zeitgemäß, neben unterschiedlichen Ziel- und Entwicklungssystem oder der Entwicklung innerhalb eines Teams. Container sind in hybriden Netzwerken, in denen mit verschiedenen Betriebssystemen gearbeitet wird, meistens besser geeignet als VMs. Der Vorteil von Containern besteht darin, dass die Technologie betriebssystemübergreifend funktioniert und auch in der Cloud eine große Verbreitung hat. Container sind also ideal dafür geeignet, wenn in Netzwerken verschiedene Betriebssysteme eingesetzt werden und auch in die Cloud Container verschoben werden sollen. Natürlich lassen sich auch VMs in hybriden Netzwerken einsetzen, allerdings nicht so stark skalierbar und flexibel, wie Container.

VI.Literaturverzeichnis

- https://medium.com/@tonistiigi/build-secrets-and-ssh-forwarding-in-docker-18-09-ae81 , letzter Zugriff 07.02.2020
- https://docs.docker.com/engine/examples/running_ssh_service/, letzter Zugriff 07.02.2020
- https://docs.docker.com/network/network-tutorial-standalone/#use-the-default-bridge-network ,letzter Zugriff 07.02.2020
- https://www.youtube.com/watch?v=hd1q5lFiUKU, letzter Zugriff 07.02.2020
- https://www.ionos.de/community/server-cloud-infrastructure/docker/starten-und orchestrieren-sie-docker-container-mit-docker-compose/, letzter Zugriff 07.02.2020
- https://www.heise.de/developer/artikel/Vagrant-und-Docker-Eine-zeitgemaesse-Entwicklungsumgebung-2411620.html, letzter Zugriff 07.02.2020
- https://t3n.de/news/vagrant-einfuehrung-592650/, letzter Zugriff 07.02.2020
- https://www.ip-insider.de/die-10-wichtigsten-vor-und-nachteile-von-docker-containern-a-844230/, letzter Zugriff 07.02.2020
- https://www.redhat.com/de/topics/containers/what-is-docker, letzter Zugriff 07.02.2020