

**Projektarbeit**

**Verwaltung unterschiedlicher Datenbanktechnologien mittels Docker**

zu

Data Engineering

Master:

“Data Science & Intelligent Analytics”

Autor:

Jochen Hollich | 1810837475

Datum

28.02.2019

Inhalt

[Theorie 3](#_Toc4480356)

[Hintergrund 3](#_Toc4480357)

[Grundlegendes zu Docker 3](#_Toc4480358)

[Grundlegendes zur Historie 4](#_Toc4480359)

[Zielformulierung dieser Arbeit, Abgrenzung und Vorgehen 5](#_Toc4480360)

[Lab-Setup 6](#_Toc4480361)

[Praxis / Dokumentation 7](#_Toc4480362)

[Linux Installation 7](#_Toc4480363)

[Docker Installation 12](#_Toc4480364)

[Docker arbeiten 14](#_Toc4480365)

[Konzeptionell 14](#_Toc4480366)

[Exemplarische Befehlsaufschlüsselung am Beispiel MS-SQL Server 15](#_Toc4480367)

[Operativ 16](#_Toc4480368)

[Interaktion mit MS-SQL-Server Containern 18](#_Toc4480369)

[Zugriff aus Linux mittels der Shell in den SQL Server hinein 18](#_Toc4480370)

[Zugriff aus dem Windows Wirtssystem in den SQL Server mittels SSMS 19](#_Toc4480371)

[Docker verteilen 20](#_Toc4480372)

[Remove 20](#_Toc4480373)

[Push 20](#_Toc4480374)

[Pull 21](#_Toc4480375)

[Abschluss SQL-Server 21](#_Toc4480376)

[MySQL 21](#_Toc4480377)

[Cassandra 23](#_Toc4480378)

[Neo4j 24](#_Toc4480379)

[Ausblick 28](#_Toc4480380)

[Quellen 29](#_Toc4480381)

# Theorie

## Hintergrund

Die vorliegende Seminararbeit wurde im Rahmen des Modules DATA ENGINEERING, betreut von Prof. (FH) Dr. Michael Kohlegger im Wintersemester 2018-19 an der FH-Kufstein angefertigt. In dem Modul Data-Engineering lernen die Studierende unterschiedliche Technologien zu Data-Engineering Fragestellungen kennen und die grundlegende Entwicklung dieser Umgebungen. Im konkreten werden die Technologien MS-SQL & MySQL als exemplarisches Beispiel für Relationale Datenbanken, Appache Cassandra als Beispiel für einen Wide-Column-Store, Neo4J aus den Graphen-Datenbanken und MongoDB exemplarisch für eine NoSQL-Datenabank kennen. Die Interaktion der Studierenden besteht überwiegend aus den Bereichen SQL-Statements der DDL, DML und DQL.

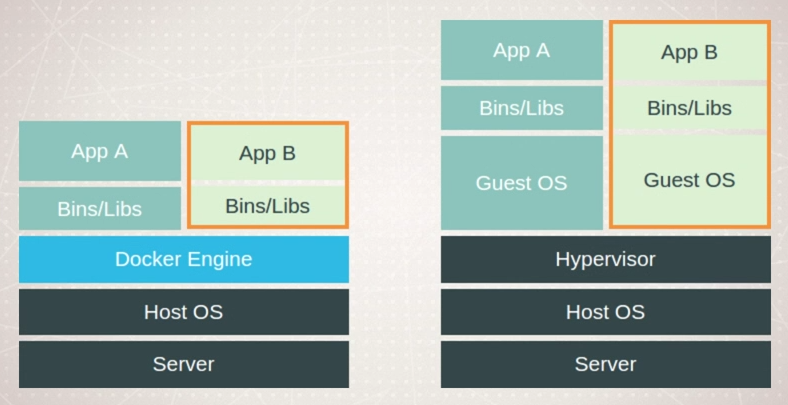
Neben dieser vorgegebenen Roadmap werden die Studierenden in einer Projektarbeit mit einer Fragestellung konfrontiert, in welcher sie sich über den abgesteckten Lernhorizont mit dem Thema Data-Engineering im Kontext von anderen Technologien bzw. anderen Anwendungsbereichen als in den Lehrveranstaltungen behandelt erweitern soll. Hierzu soll ein individuelles Projekt angefertigt und dokumentiert werden.

Ich habe mich für das Projektthema „Umsetzung unterschiedlicher Datenbanktechnologien mittels Docker“ entschlossen, da mich neben der erlernten Entwickler- Brille auch die Management und Administrations Sicht von Datenbanken interessiert. Docker ist eine Container-Virtualisierungsumgebung für unterschiedliche Dienste.

Für die Umsetzung der Aufgabe wurde sich eng an den Lehrplan des Udemy Kurses [Docker Mastery](https://www.udemy.com/docker-mastery/) gehalten. Dieser Kurs hat in didaktisch gut aufgearbeiteter Art und Weise alle wichtigen Inhalte für diese Seminararbeit bereitgestellt.

## Grundlegendes zu Docker

Das Anwendungsprogramm Docker lässt sich am besten dadurch erklären indem man einen Vergleich gegenüber Virtualisierungsprogrammen zieht:



In der klassischen Darstellung mittels Virtualisierung (Rechts in der Grafik)reden wir in der Regel von einem „OS im OS“. Neben den didaktischen Anwendungsfällen hat diese Virtualisierung aber auch Use-Cases im Bereich der Wirtschaft, da hierbei u.a. gegebene physische Ressourcen effizienter verwendet werden können.

Ziel in einem wirtschaftlichen Szenario könnte sein eine Neue Funktionalität bzw. einen Dienst in das Unternehmen aufzunehmen.

**Lösungsbeschreibung Virtualisierung:**

In einer Virtualisierungsumgebung würde man für die neue Funktionalität vermeintlich einen neuen (virtuellen) Server (Linux/Windows….) ausrollen und on-top dieses neue OS den jeweiligen neuen Dienst/ Funktionalität bereitstellen. Durch die Virtualisierung wird unter anderem die Hardware abstrahiert, was zu geringeren Schwierigkeiten in der Wartung führen kann, aber als Resultat für den neuen Dienst muss ein weiteres OS (Das virtuelle Host Operating-System für den Dienst) gewartet werden.

**Lösungsbeschreibung Container-Management:**

Wenn man oben beschriebenes Szenario mit einer Container Management Software wie Docker löst dann geschieht dies nach folgendem Vorgehen:

Docker wird zunächst in dem physischen Hostsystem implementiert. Sobald Docker ausgerollt ist wird ein Container erzeugt, welcher die nötigen Umgebungsvariablen für den Dienst beinhaltet. Somit kann man den gewünschten Dienst direkt in diesen neuen Container hinein-deployed und der Dienst kann aus dem Container heraus mit den restlichen Teilnehmern (Hostsystem & etwaiges physisches und logisches Netzwerk kommunizieren). Mit diesem Ansatz wurde das etwaige OS-System als Basis von einem Dienst, wie im Beispiel der Virtualisierung abstrahiert. Demzufolge sind die reinen Core-Container hinsichtlich ihrer Größe sehr schlank.

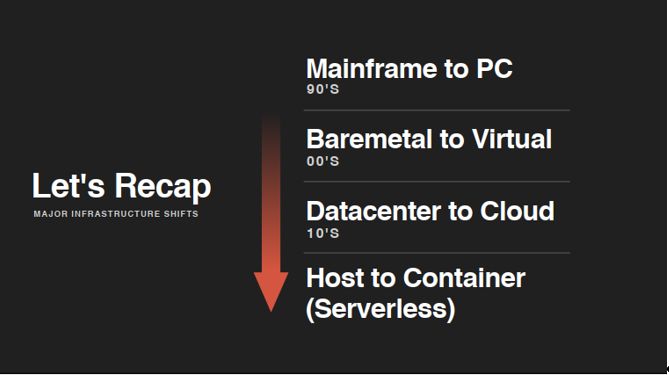
Docker bietet hierbei neben der Erstellung und Verwaltung von Containern auch die Möglichkeit diese Container in bestehende Netzwerke mittels unterschiedlicher Technologien (v.a. NAT & Bridged) einzubinden. Dazu werden den Container IP Adressen zugewiesen, auf welche dann von den jeweiligen Clients zugegriffen werden kann.

Vorteile aus der Containerverwaltung

1. Schnelles Deployment von Server-Diensten
2. Rasches Entwickeln von Umgebungen für Developmentteams
3. Einsparung von Festplattenkapazität durch fehlendes (virtuelles) Server-OS-System
4. Abstraktion des Hosts-Systems => Einsparung von Wartungszeiten
5. Plattformunabhängigkeit der Container

## Grundlegendes zur Historie

In der IT leben große Innovationen in der Regel etwa eine Dekade, wenn wir nun die „Booms“ der letzten 30 Jahren betrachten so können wir feststellen, dass in den 90er Jahren die Bewegung von großen Mainframe Rechner zu Stand Alone PCs erkennbar war. Die 00er Jahre basierten auf der Abstraktion der Hardware und die Virtualisierung war ein zentraler Begriff. Durch den Ausbau der Internettechnologien wurde es in den 10er Jahren, in welchen wir uns gegenwärtig befinden die Abstraktion von ganzen Datenzentren, durch die Migration in den ominösen Begriff Cloudcomputing abgelöst. Es gibt Meinungen, dass die nächste große Welle in der IT die Abstraktion von OS sein wird indem man gewünschte Dienste oder Applikationen direkt in Container verschiebt und für diese Anwendungszwecke ausschließlich mit Containern plattformunabhängig arbeiten wird.



## Zielformulierung dieser Arbeit, Abgrenzung und Vorgehen

Das Ziel, welches in diesem Seminar-Projekt verfolgt wurde lautet:

**„Verwaltung unterschiedlicher Datenbanktechnologien mittels Docker“**

Da die Docker Communtity Edition(kostenlos) im Gegensatz zur Enterprise Edition (Kostenpflichtig) nicht unter den Red-Head Linux Distributionen lauffähig ist wurde sich für das Debian basierte OS Ubuntu entschlossen. Außerdem waren in diesem Bereich seitens des Studierenden grundlegende Kenntnisse vorhanden.

Dieses Ubuntu-OS wurde wiederum als Desktop Version ausgerollt, da sich der Studierende zum Zeitpunkt der Erstellung dieser SA nicht fundiert mit der Anwendung Docker ausgekannt hatte. Im Laufe der Umsetzung des Projektes wurde sich schrittweise in Docker eingearbeitet und aus diesem Grund war es hilfreich innerhalb des Host Systems von Docker (Virtueller Maschine/ Ubuntu) eine grafische Oberfläche zu besitzen.

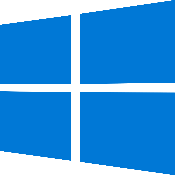
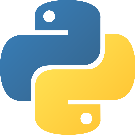
## Lab-Setup











Das blaue Feld in der obigen Darstellung symbolisiert das Grundlegende Host/Wirts-Operating System. In diesem Fall liegt ein „Windows 10 PRO“ in der Version „10.0.17763 Build 17763“ vor. Innerhalb sind die nötigen Managementtools für die Containerinstanzen installiert. Weitere Betriebssysteme werden mittels „VMware® Workstation 12 Pro 12.5.2 build-4638234“ virtualisiert.

Das Orange Feld symbolisiert die logische virtuelle Maschine Ubuntu. Die genauen HW-Spezifikationen werden in der Dokumentation erläutert. In dieser virtuellen Maschine wiederum Ubuntu -Desktop Version 18.04.01 als Basis für die Anwendung „Docker 18.09.0“ installiert. In diese Umgebung wird die Docker Applikation eingebettet.

Aufbauend auf der Docker Anwendung wiederum wird ein Container erzeugt in welchem eine Instanz des SQL Servers läuft und soweit konfiguriert, sodass man aus dem aus dem Windows Host direkt auf diese Container-Instanz mit einem geeigneten Management Tool für die jeweilige Technologie zugreifen und Änderungen vornehmen kann.

# Praxis / Dokumentation

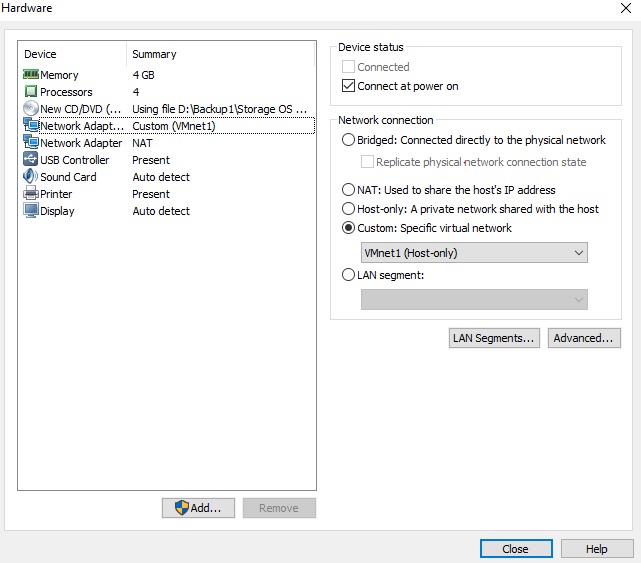
Der nachfolgende Teil dieser Seminararbeit stellt die Dokumentation der operativen Umsetzung des Projektes beginnend von der initialen Laberstellung in VM-Ware, über die Integration der Technologien bis hin zum zentralen Austausch der Docker Container.

## Linux Installation

Wie bereits in der Theorie beschrieben wird zunächst auf dem bestehenden Host System eine Linux Ubuntu Desktop nach dem aktuellsten Stable-Release benötigt. Hierzu wird die Iso entsprechend von der Ubuntu Webseite heruntergeladen:

<https://www.ubuntu.com/download/desktop>

Im Anschluss wird diese Iso Datei auf einer virtuellen Bare-Metall mit folgenden Hardwarespezifikationen deployed:



Was an dieser Hardwarekonstellation signifikant ins Auge sticht sind die beiden NIC-Adapter, die in dieses Setup integriert worden sind. Der Grund hierfür liegt darin:

Nic 1: Hier ist die Virtuelle Maschine auf dem VM-Ware Switch 1 / VM-Net1 (Host-Only) angeschlossen. Diese entsprechende NIC ist mit einer statischen IP ausgestattet. Somit kann aus dem Host System mit einer entsprechenden SSH-Applikation auf der CLI-Ebene eine Verbindung zu dem virtuellen Server hergestellt werden. Das hat den Vorteil, dass bei der Entwicklung und Umsetzung der Lösung lediglich „auf einem OS“ (in diesem Falle Windows 10) gearbeitet werden kann. // es wurde versucht die Netzwerkkonfiguration des basierenden Host-Windows Systems dahingehend zu ändern, sodass dieser als an der virtuellen Schnittstelle zur Linux VM in Form eines virtuellen Routers zwischen den Netzwerken übersetzt, dies ist jedoch leider nicht geglückt und wurde somit mit den beiden NICs umarbeitet

Nic 2: Hier ist die Voreinstellung NAT (Network-Address-Translation) gewählt. Somit wird durch das Host System hindurch eine Konnektivität mit dem Internet gewährleistet. Die IP-Vergabe auf dieser NIC übernimmt der DHCP-Dienst der VM-Ware intern.

Das Deployment von Ubuntu wurde standardmäßig durchgeführt. Die Credentials für dieses OS in diesem Test-Lab lauten:

USER: labadmin

Password: Test123

sudo-Password: Test123

Nach dem fertigen Deployment des OS wurde zunächst die Maschine hinsichtlich der hinterlegten Repositories, der Applikation und des OS-Kernels auf dem aktuellsten Stand gebracht mittels des Befehles:

sudo apt-get update && sudo apt-get -y upgrade && sudo apt-get dist-upgrade && sudo reboot

Nach dem Updaten wird mit dem Befehl:

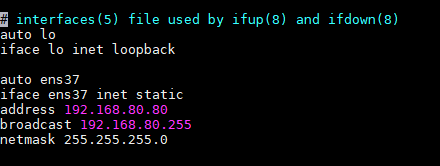
sudo apt-get install openssh-server

der SSH-Server auf der Maschine installiert. Ab diesem Zeitpunkt kann von dem Host System mit einem geeigneten SSH-Client Application auf den Ubuntu Server zugegriffen werden. Da sich jedoch im Moment noch eine Dynamische IP auf NIC 2 hinterlegt ist, schalten wir im Nächsten Schritt die IP-Vergabe von NIC2 auf statisch, sodass gewährleistet werden kann. dass auch künftig die Maschine unter der gleichen IP zu erreichen ist:

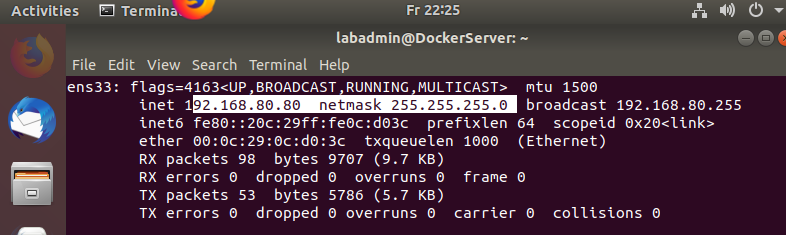
1. Öffnen des entsprechenden Textfiles:

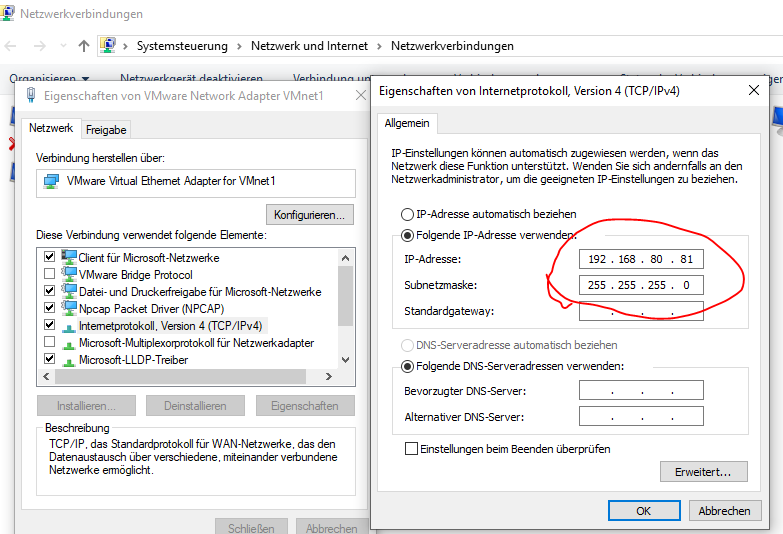
sudo nano etc/network/interfaces

Und hinterlegt folgenden Text:

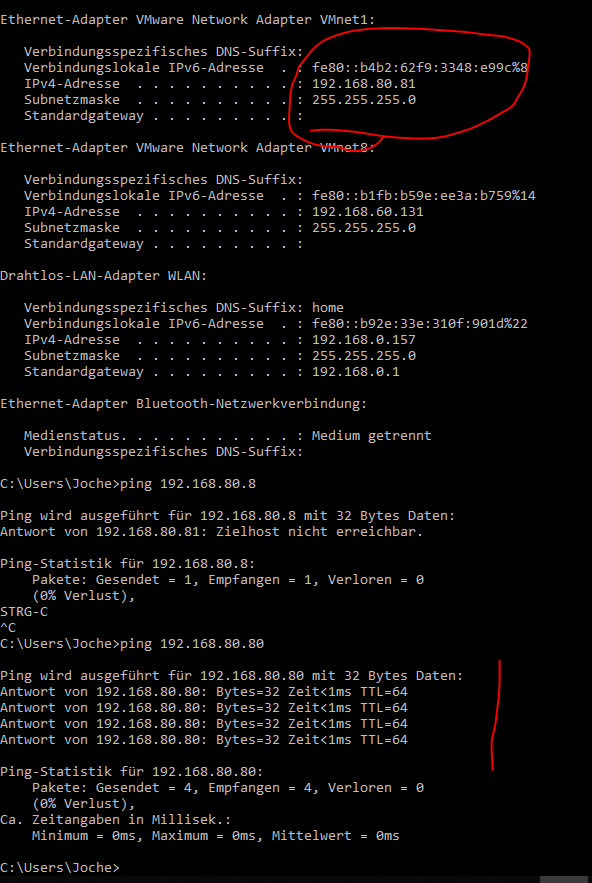


Ob die Konfiguration geglückt ist kann in Linux mittels ifconfig überprüft werden.



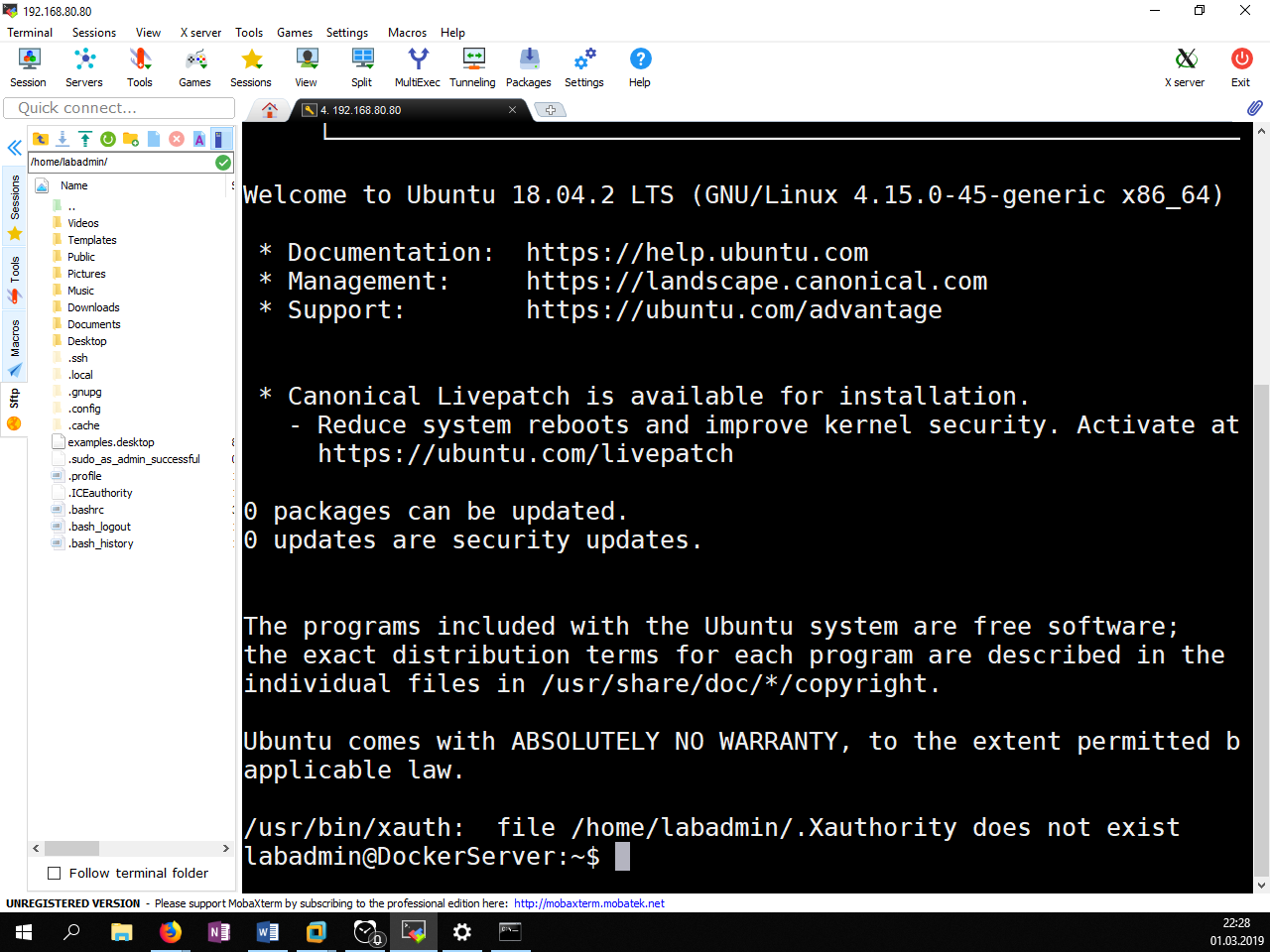
Die IP & Subnetzmaske sind für dieses Szenario angepasst und müssen entsprechend auf das jeweilige Test-lab angepasst werden.

Ob nun eine Verbindung zwischen den beiden logischen Betriebssystemen besteht wird mittels ping geprüft



Mit diesem Schritt ist das Deployment, updaten und bereitstellen der nötigen Entwicklungsumgebung auf dem virtuellen Server abgeschlossen. Der Windows Client(mein Hostsystem in diesem Szenario), erhält auf dem virutelle Interface zur Linux-VM ebenfalls eine statische IP-Adresse im selben Netz.

Im weiteren Verlauf dieser Dokumentation wird die Kommunikation mit dem Ubuntu Server und somit auch das Deployment von Docker mittels der SSH-Lösung vom MobaX-Terms aus dem Windows Host heraus geschehen. Der Grund weshalb MobaX-Term als SSH Client verwendet wird ist da hierin für die Windows Nutzer eine freundliche Möglichkeit zum Datei-Exchange zwischen Windows und Ubuntu Host besteht. Somit können auch die Testdatenbanken in das Test-Lab zum Ubuntu Host übertragen werden. Ein weiterer zentraler Vorteil ist darin zu finden, da durch die zentrale Verwaltung nun auch befehle im Windows kopiert und direkt auf CLI- Ebene eingefügt werden kann, sowie dass das Tastaturlayout sich dem gegebenen Layout des Hostsystems anpasst. Somit wird eine lästige Verwaltungsarbeit des Linux Servers umgangen.



An dieser Stelle empfiehlt es sich in der VM-Ware als Sicherheitsmaßnahme einen Snapshot zu erstellen

## Docker Installation

Docker in einer Linux-Environment kann auf unterschiedlichste Weisen installiert werden. Hierbei spielt jedoch immer das zugrundeliegende OS und die entsprechende Hardwarekonstellation eine zentrale Rolle. Es gibt nun die Möglichkeit sich anhand der detaillierten Docker-Dokumentation durch das Deployment dieser Applikation „hindurch zu hangeln“, oder mittels des Docker-Scriptes die entsprechende Applikation zu installieren. Mittels des Scriptes werden zunächst die Umgebungsparameter geprüft (OS des künftigen Docker Servers/ Version OS / Etwaige HW-Komponenten…) und auf Basis dieser Informationen automatisiert das beste Docker für das etwaige System ausgewählt und installiert. Im weiteren Schritt wurde sich für das Deployment mittels des Scriptes entschlossen. Das Script ist einsichtlich unter:

<https://get.docker.com/> .

Als Vorbereitung für das Script-Deployment ist auf Ubuntu die Applikation CURL notwendig. Diese wird zunächst ausgerollt:

**sudo apt install curl**

Im nächsten Schritt wird das Script heruntergeladen:

**curl -fsSL https://get.docker.com -o get-docker.sh**

und mit dem Befehl

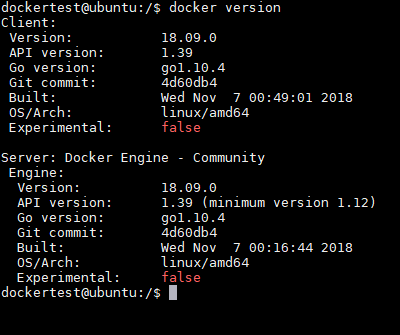
**sh get-docker.sh**

wird das Script ausgeführt und somit Docker „optimal“ für die jeweilige Umgebung ausgerollt.

Mittels des Befehls:

**docker version**

ist erkenntlich, dass Docker nun auf dem Server installiert und betriebsbereit ist:



Alternativ Docker direkt in windows mittels der Toolbox

Hier brauch ich für den Doker dienst keine eigenständige Virtuelle maschine und rei mal nicht einen Hyper v, das ist für den Windows User optimal.

Netzer Install Docker Toolbox

Das ist Docker ohne Hyper V für Windows

<https://docs.docker.com/toolbox/toolbox_install_windows/>

Nachdem das setup durchgelaufen ist / braucht einige berechtigungen für gewisse adapter von der Oracle box / sollte parallel zur VM laufen //

Danach öffnen der Docker Quickstart



## Docker arbeiten

Der folgende Abschnitt wird zunächst auf das konzeptionelle Arbeiten mit Docker und im Anschluss das operative Arbeiten mit Docker eingehen

##### Konzeptionell

Wie bereits einleitend beschrieben ist Docker ein Tool zu Containerverwaltung und zum Ausführen dieser Container. Diese Container können entweder selbst erzeugt werden oder direkt von einer entsprechenden Plattform bezogen werden. Im Zuge dieser Seminararbeit werden mit Docker Container der Datenbanktechnologien

1. MS-SQL-Server
2. MySQL
3. Cassandra
4. Neo4J
5. MongoDB

arbeiten.

Eine Möglichkeit wäre diese Container selbst zu erstellen und mit diesem zu arbeiten. Dies erfordert jedoch vertiefte Kenntnisse mit dem Programm Docker. Die vorliegende Seminararbeit hat jedoch den Fokus auf das Zusammenspiel zwischen Docker und jeweiligen Technologien als auf tieferes Docker Verständnis. Daher verwenden wir im weiteren Schritt die bereits bereitgestellten Image-Dateien, aus welchem wir einen Container erzeugen.

**Was geschieht, wenn ein Container bereitgestellt wird?**

Hierzu führen wir einen weiteren Begriff „Image“.ein. Diese Image-Dateien (Binaries/Dependencies/Metadaten für Applikation) werden in ihrer Grundkonfiguration auf unterschiedlichen Plattformen ausgetauscht und speichern die Grundsettings eines Containers ab. Aus diesen Grundsettings kann man dann laufende Instanzen sogenannte Container erstellen. Dieses Verhalten lässt einen Vergleich zur Objektorientierten Programmierung zu. Hierbei werden aus Klassen (Docker-Terminologie: Images) einzelnen Instanz (Docker-Terminologie: Container/Instanz) erstellt. Die jeweiligen Instanzen können im nächsten Schritt individualisiert und modifiziert werden. Dadurch können aus einem Image beispielsweise mehrere Container erzeugt werden.

Somit wird beim ersten Aufrufen einer Instanz zunächst geprüft ob die jeweils zugrundeliegende Image-Datei bereits lokal auf dem Hostserver Ubuntu/„Docker-Server“ bereits existiert. Ist diese noch nicht vorhanden wird die nötige Image-Datei von dem eingestellten „Image-Hub“ bezogen. Default (docker-Hub <https://hub.docker.com/>). In unserem Falle Verwenden wir sowohl dieses default Docker Hub für den Bezug der Image.

## Exemplarische Befehlsaufschlüsselung am Beispiel MS-SQL Server

Bei der Instanz-Erstellung muss außerdem über Schalter der Container soweit mit Umgebungsvariablen konfiguriert werden sodass dieser entsprechend mit seiner Umwelt kommunizieren kann. Konkret an einem Beispiel veranschaulicht bedeutet dies, dass mittels des Schalters:

docker container run -e 'ACCEPT\_EULA=Y' -e 'SA\_PASSWORD=yourStrong(!)Password' -p 1433:1433 -d -name MSSQLServerDOCKER -v ~/SQLData:/home/dockertest/SQLData microsoft/mssql-server-linux:2017-CU8

Der Befehl docker container run ist der Basic Command um eine Instanz eines Images / einen individuellen Container zu starten. Wie beschrieben wird etwaig das Image heruntergeladen und dann entsprechend aus dem Image heraus ein individueller Container erzeugt.

Der Schalter Environment / -e 'ACCEPT\_EULA=Y' besagt lediglich, dass die End User Licence von Microsoft ausführenden humanen Person akzeptiert wird.

Der User System Administrator (SA) spielt in der Umgebung des MSSQL Servers eine sehr zentrale Rolle. Daher muss dieser beim Erzeugen einer Instanz vorab mit einem entsprechenden Passwort konfiguriert werden. Dies wird mittels des Schalters - e SA\_PASSWORD=yourStrong(!)Password' erreicht-

Der grüne Sektor bedeutet, dass der Host, auf dem für MS-SQL-Server defaultmäßigen Port 1433 an der Hostinternen FW lauscht und diesen Port bei Ansprache an den MSSQL-Server defaultmäßigen Port 1433 an den Container weiterleitet. Somit ist gewährleistet, dass auf einem running Container von dem Zugrundeliegenden Host-Operating System durch das Virtuelle Ubuntu auf dem Port 1433 hin zudem Container auf dem Port 1433 zugegriffen werden kann.

Mittels des Schalters detach| -d wird festgelegt, dass dieser Container zwar auf der CLI aufgerufen wird, aber im Hintergrund läuft. Somit kann die aktuelle CLI Sitzung weiter genutzt werden.

Mittels des Schalters -name MSSQLServerDOCKER wird festgelegt unter welchem Namen die Instanz angesprochen werden kann. Der Status einer Instanz kann ein und ausgeschaltet sein. Wie bereits oben beschrieben haben jedoch laufende Container für die Kommunikation mit der Umwelt entsprechend eine IP-Adresse. Diese Adresse kann jedoch mit einem „reboot“ eines Containers erneuern. Aus diesem Grunde ist innerhalb der Docker Container-Verwaltung die Namenszuweisung der Container in Zusammenhang mit einem internen DNS-Dienst fest verankert.

Ein weiteres zentrales Element ist das Einbinden von einem Shared Laufwerk zwischen dem Hostsystem(Ubuntu) und dem Container. Dies wird mittels v ~/SQLData:/home/dockertest/SQLData erledigt. Hierbei muss davor beachtet werden, dass das Verzeichnis ~/SQLSData auf Ubuntu eben angelegt wird.

Im letzten Schritt muss der Name der Imagedatei und das dazugehörige Repository angegeben werden. In diesem Falle laden wir das Image aus dem Default-Repository (Docker Hub) herunter. Für diese Aufgabe ist die Zeile: microsoft/mssql-server-linux:2017-CU8 verantwortlich.

##### Operativ

Bevor wir die Images herunterladen und erzeugen wird zunächst das spätere Sharelaufwerk mittels des Befehles:

mkdir ~/SQLData

in dem Home-Laufwerk des Users, in meinem Fall „labadmin“ erzeugt. An diesem zentralen Verzeichnis sollen später die Daten zwischen dem Host und den Docker Containern ausgetauscht werden.

Außerdem gewähren wir dem user docker in unserem Lab volle administrationsrechte mittels:

sudo usermod -a -G docker $USER

Dies ist in unseren Lab-Szenario bedenklos umzusetzen, da wir mit keinen sensiblen/produktiven/schützenswerten, sondern ausschließlich mit fiktiven Daten arbeiten. Für Produktivumgebungen ist Docker grundsätzlich nicht gedacht, da hier auch das Sicherheitskonzept nicht „state oft he art“ ist. Aber auch für Entwicklungsumgebungen könnte man über ein geeignetes Rechtesystem verwaltet in Linux etwas Sicherheit in das Lab bringen. Der Einfachheit zu liebe wurde dieser Schritt übersprungen und der Verwaltungsuser Docker und labadmin gänzlich mit den nötigen sudo-Rechten ausgestattet.

Für unser Test-Lab beschaffen wir uns einen unabhängigen Container, von

**MS-Hub**

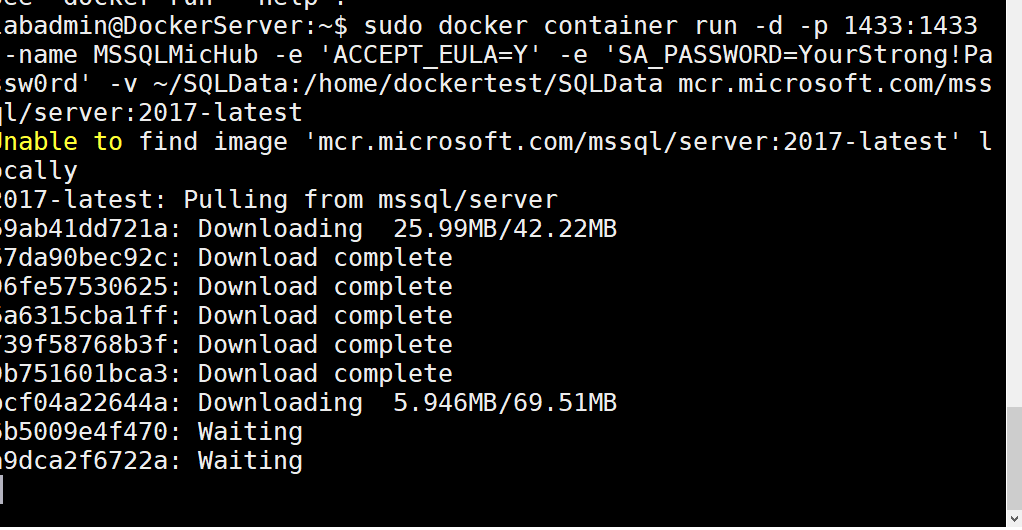
sudo docker container run -d -p 1433:1433 --name MSSQLMicHub -e 'ACCEPT\_EULA=Y' -e 'SA\_PASSWORD=YourStrong!Passw0rd' -v ~/SQLData:/home/dockertest/SQLData mcr.microsoft.com/mssql/server:2017-latest

(Developer-Edition)

(Die übrigen Container der MySQL, Cassandra, Neo4J und MongoDB werden künftig vom Default Docker hub bezogen, hier wurde exemplarisch von einem anderen Verteilungshub das Image bezogen)

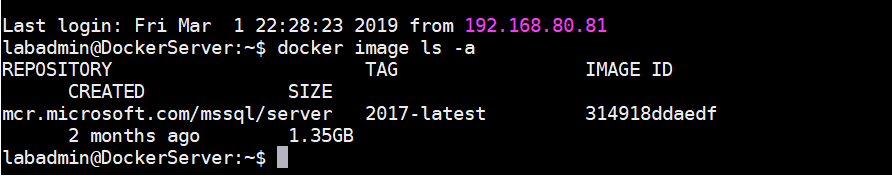
Beim erstmaligen Aufruf dieser Befehle werden die Image Dateien vom jeweiligen-Repository heruntergeladen und eine laufende Instanz/Container mit den gegebenen Environment-Variablen gestartet.

Hier der initiale Download:



Die bestehenden Images, welche auf dem Docker-Server hinterlegt sind können mit dem Befehl:

**docker image ls -a**



Angezeigt werden.

Hieraus ist zu sehen, dass das Image vom Microsoft Hub erst vor 2Monaten (relativ zur Erstellung dieser Dokumentation auf dem jeweiligen Repository veröffentlicht wurde.

Der Status der existenten Container im Server wiederum kann über den Befehl:

**docker container ls:**



Hierbei erhält man folgende Informationen:

Container ID: Zur Abgrenzung und Ansprache einzelner Container

IMAGE: Das zu grundliegende Image für die Erstellung des Containers

CREATED: Zeitpunkt wann der Container auf diesem Server erzeugt wurde

STATUS: Status ob der Container laufend oder abgeschaltet ist

PORTS: Auf welche Ports im Host bzw. im Containermanagement wird gelauscht

NAMES: Namensgebung des Containers, wichtig für die DNS Ansprache

## Interaktion mit MS-SQL-Server Containern

Wie der Status einer Imagedatei bzw. eines Containers ist wurde im oberen Kapitel bereits erläutert. In diesem Teilbereich wollen wir auf die Befehle eingehen mit welchen Container gestoppt und gestartet werden können.

Da durch den oberen Befehls direkt nach dem Download des Images ein Container mit entsprechenden Umgebungsvariablen erzeugt wird besteht ein laufender Container, welcher über **docker container ls** -a angezeigt werden kann. Dieser laufende Container kann mit dem Befehl

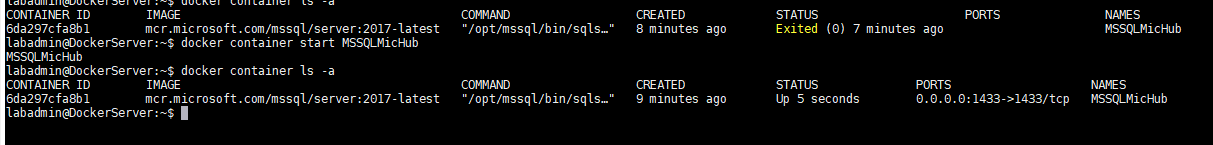
**docker container stop „Name“**

gestoppt werden. Ab diesem Zeitpunkt sind die Dienste des Containers nicht mehr erreichbar.

Um einen gestoppten bereits existierenden Container wieder zu starten kann mittels

**docker container start „Name“**

der ausgewählte Container wieder gestartet werden



### Zugriff aus Linux mittels der Shell in den SQL Server hinein

Ziel ist es hier auf Basis der Host Bash in den Container hinein die Befehlszeile zu erlangen um somit TSQL Befehle(SQLServer spezifischer Dialekt) abzufeuern

**docker exec -it MSSQLMicHub "bash"**

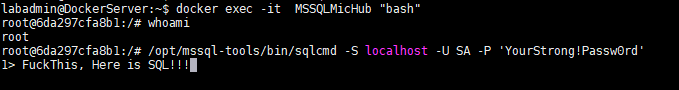
nun befinden wir uns auf der Kommandozeile des SQL-Servers und mittels



melden wir uns am Server direkt an um TSQL Scripte auszuführen

**/opt/mssql-tools/bin/sqlcmd -S localhost -U SA -P 'YourStrong!Passw0rd'**

Bitte beachte die Roten „ ‚‘ “



Ab diesem Zeitpunkt können TSQL Scripte ausgeführt werden und somit ist das erste Ziel von dem Ubuntu Host auf den MS-SQL Container zuzugreifen erreicht

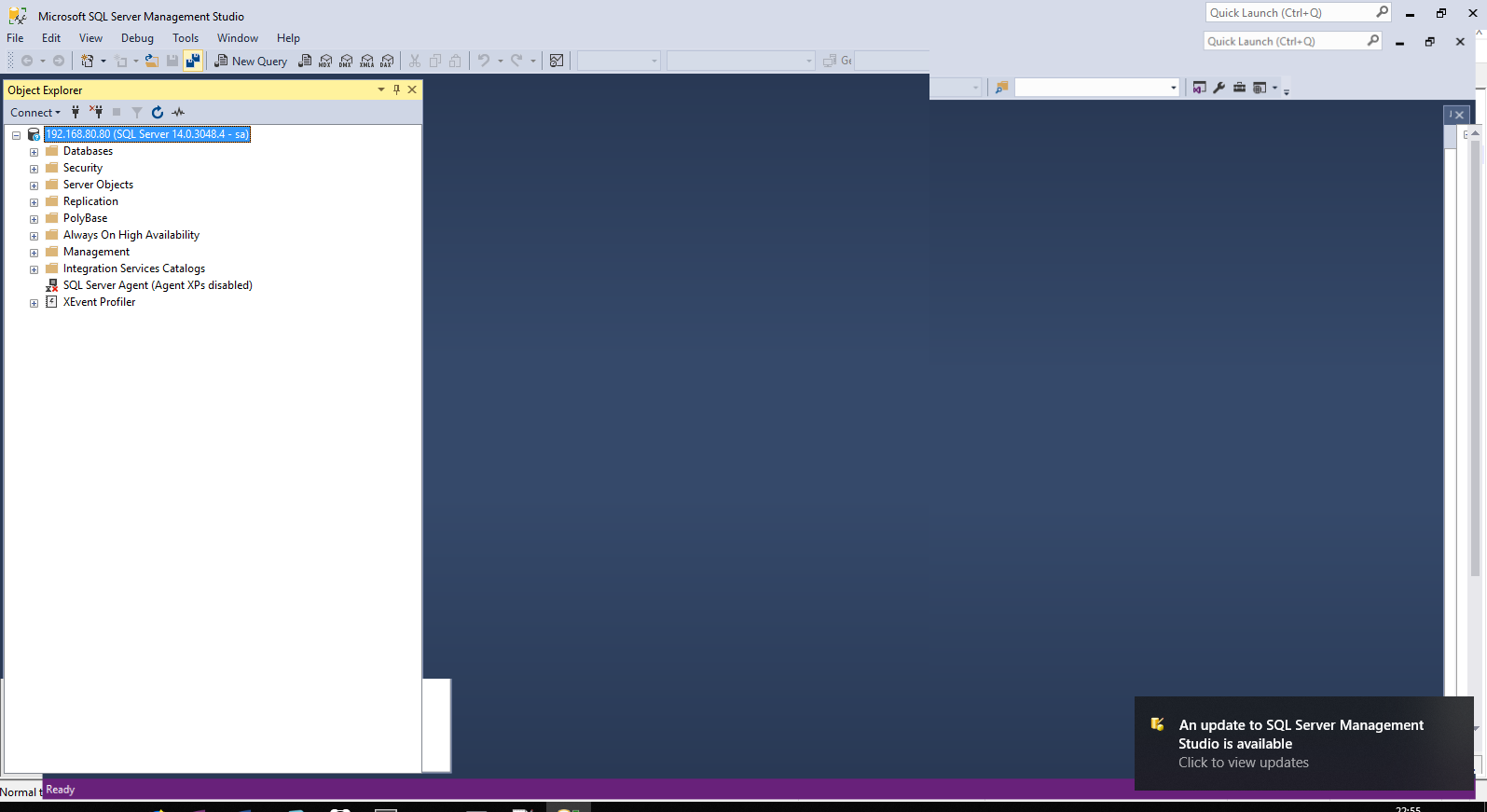
### Zugriff aus dem Windows Wirtssystem in den SQL Server mittels SSMS

Bedingt durch die Port Bezeichnung beim erstellen einer Instanz wird dem Host System Ubuntu von Beginn an mitgeteilt, dass eingehende Anfragen auf dem Port 1433 direkt an den Container weitergeleitet werden. Auf dem Container gehen diese Anfragen ebenfalls auf dem Port 1433 ein:

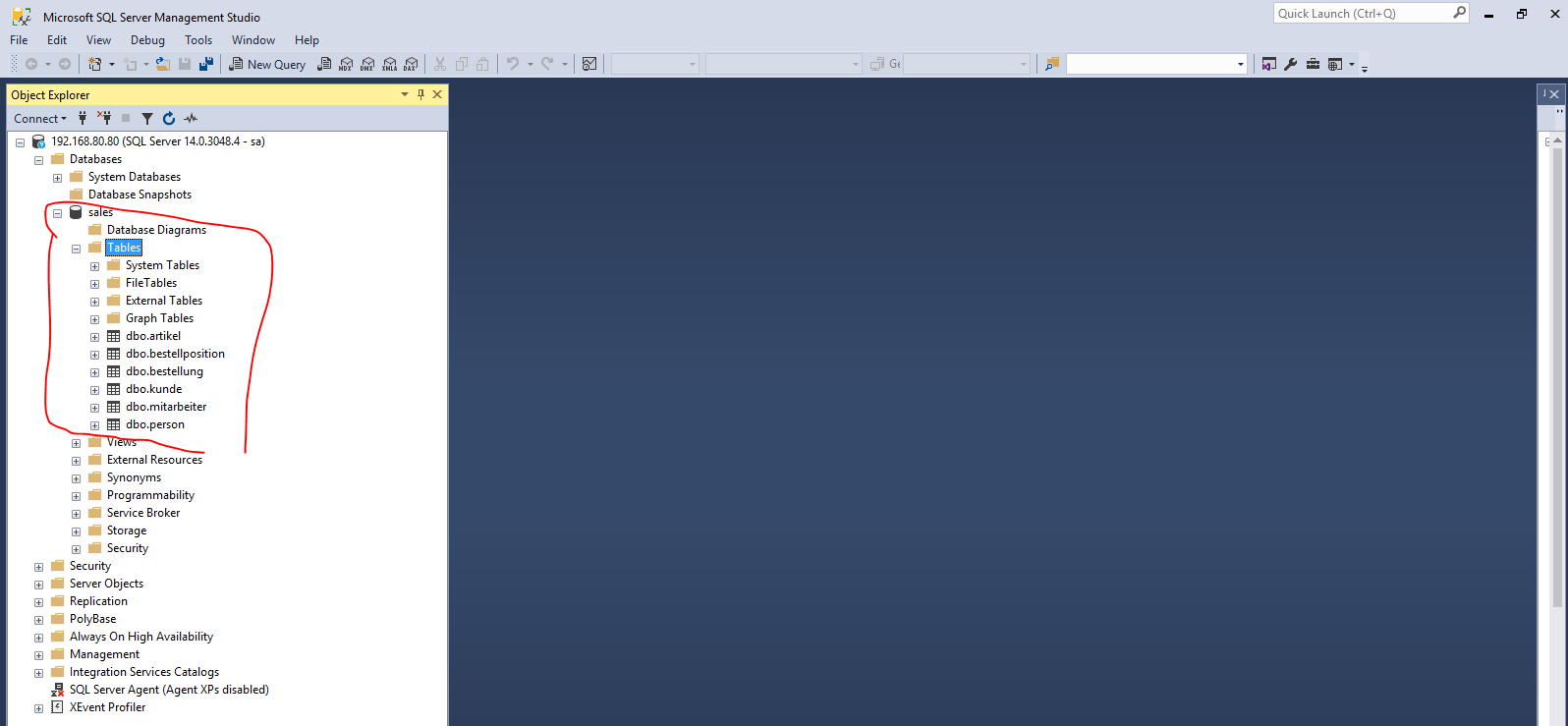
docker container run -e 'ACCEPT\_EULA=Y' -e 'SA\_PASSWORD=yourStrong(!)Password' -p 1433:1433 -d -name MSSQLServerDOCKER -v ~/SQLData:/home/dockertest/SQLData microsoft/mssql-server-linux:2017-CU8

Somit ist eine Verbindung aus dem Basis Windows Host direkt zu dem Docker Container erreicht:

Computergenerierter Alternativtext:
Microsoft SQL Server Management Studio 
Edit View Debug Tools Window Help 
, v' New Query 
Object Explorer 
Connect • 
Quick La 
Connect to Server 
Server type 
Server name 
Password 
SQL Server 
Database Engine 
192 1688080 
SQL Server Authentication 
Remember password 



Hier kann dann das bereitgestellte [TSQL-DDL File](https://gitlab.web.fh-kufstein.ac.at/kohleggermichael/de4ds/raw/master/Mega-Tutorial___SQL/sales-db___DDL-DML___SQL-Server.sql) ausgeführt werden, welches im Umkehrschluss die DB mit den entsprechenden Tabellen erstellt:



### Docker verteilen

#### Remove

Mittels des Befehles rm kann man Container oder Images aus dem System wieder entfernen. Hierzu muss man die Container / Image ID Kennen

Docker container/image rm “Container/ImageID”

#### Push

Docker ist ein Tool um Container schnell und effizient zwischen unterschiedlichen Teilnehmern, in einem privaten Netzwerk mittels eines eigenes Docker Servers oder einem öffentlichen Docker Hub zu verteilen. Hierzu muss aus einem laufenden Container zunächst mittels des Befehles

**docker commit „Container-id“ „Image-Name“**

**docker commit c52186187013 jhc1990/test**

Eine Image Datei erstellt werden. Diese neu erstellte Image Datei ist nun auch unter den lokalen Image Dateien mit folgendem Befehl ersichtlich. Wichtig an dieser Stelle ist die Namensgebung des Images. Jhc1990 ist mein Benutzername an der Docker Plattform und alle meine öffentlich geteilten Image Dateien müssen somit der Namenkonvention „jhc1990/\*“ folgen.

**docker image ls -a**

Für den Upload dieser Image Datei in den Docker Hub hinein muss man sich zunächst mit den individuellen Docker Credentials anmelden (Hierzu muss man den Registrierungsprozess unter XYZ durchführen)

**docker login**

Zuletzt wird mittels

**docker push „Containername “**

**docker push jhc1990/test**

Das eben erstellte Image auf die entsprechende Plattform hinaufgeladen.

#### Pull

Diese erzeugte Imagedatei ist nun unter <https://hub.docker.com/r/jhc1990/test/> öffentlich ersichtlich und kann von jedem Interessenten, der eine entsprechende Docker-Umgebung besitzt mittels des Befehles

**docker pull jhc1990/test**

in die externe individuelle Docker-Umgebung übernommen werden.

### Abschluss SQL-Server

Somit haben wir einen ersten vertieften Einblick in die Zusammenarbeit zwischen Docker und der Relationalen Datenbanktechnologie eingesehen. In den nächsten schritten werde ich mein Vorgehen weniger detailliert dokumentieren, da häufig Befehle ähnlich bis nahezu identisch wie bisher geschrieben werden

## MySQL

Ein geeigneter MySql Container kann analog wie der MSSQL-Container mittels des Befehles:

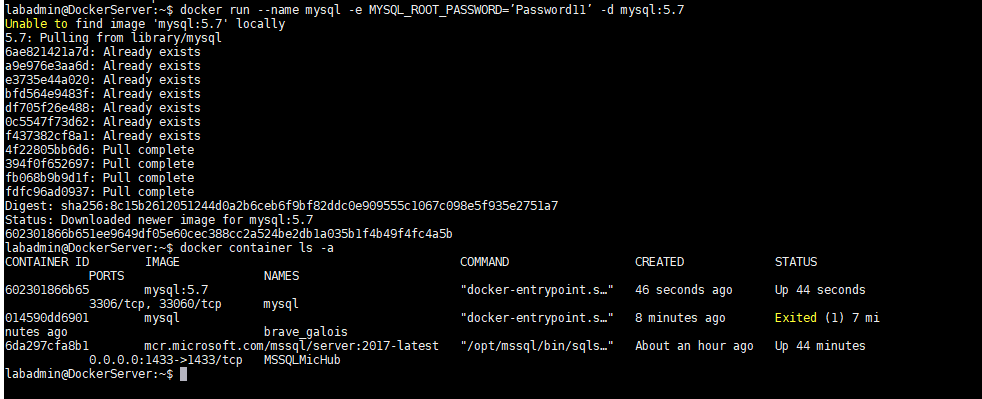
docker run -d -p 3306:3306 --name mysql2 -e MYSQL\_ROOT\_PASSWORD=’Password11’ mysql:5.7

angelegt werden.

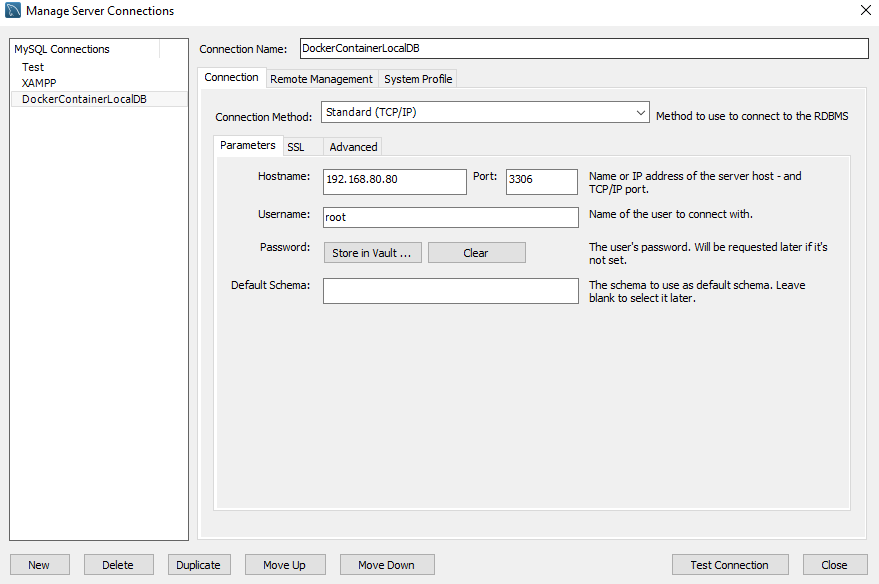
Im Anschluss ist dieser Container über

Docker Container ls -a

In der Container Ablage ersichtlich und kann entsprechend gestartet und wenn gewünscht gestoppt werden:



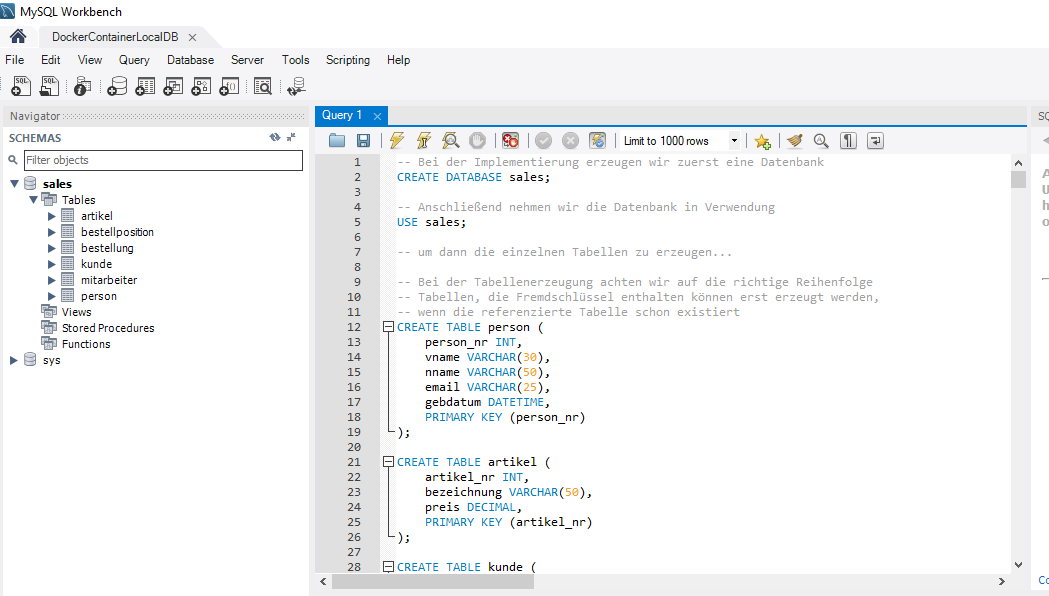
Ausgehend aus dem laufenden Container in der virtuellen Instanz kann nun aus dem Windows Host im lab mittels des MySQL Verwaltungstools Workbench auf die DB zugegriffen werden:



Der Standarduser ist root, und das Passwort wurde in der Instanzierung des Containers gewählt (docker run -d -p 3306:3306 --name mysql2 -e MYSQL\_ROOT\_PASSWORD=’Password11’ mysql:5.7

)

Sobald die Anmeldung erfolgreich geschehen kann mit dieser Datenbankinstanz über die Management Software interagiert werden:

.

Auch diese Instanz wurde auf das Docker Hub hochgeladen und ist unter:

<https://hub.docker.com/r/jhc1990/mysql_lab_dataeng>

ersichtlich. Die dafür nötigen Schritte sind gänzlich analog dem vorgehen wie es beim MSSQL Server vorgenommen wurde.

## Cassandra

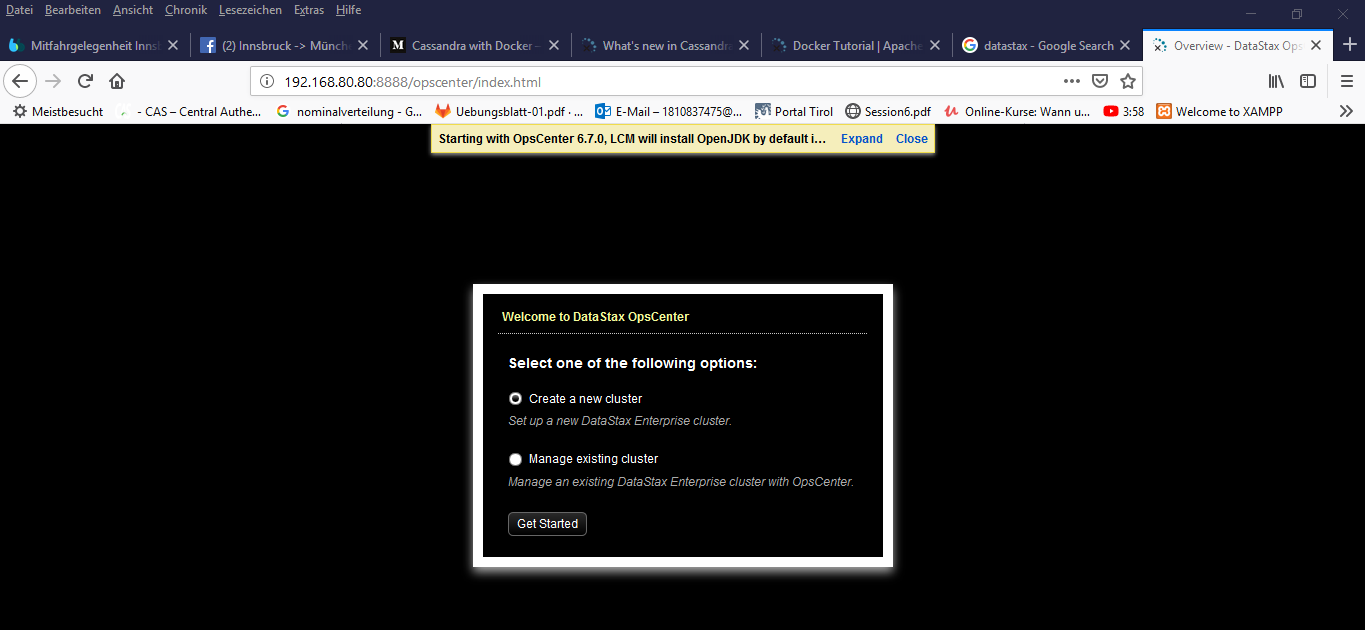
Um die Cassandra Technologie zu implementieren wurde nach folgendem Tutorial vorgegangen: <https://academy.datastax.com/content/docker-tutorial> .

Hierbei wird zunächst DataStax, welches die Appache Cassandra beinhaltet ausgerollt und anschließend mittels DataStax Studio auf die eigentliche Datenbank zugegriffen. Mittels des DataStaxOpsCenter wird eine Management Umgebung mittels Webtechnologien bereitgestellt.

1. Beschaffen der nötigen Images
2. Bevor der eigentliche Datenbankdienst muss ein Container für das Managementstudio erstellt werden
3. docker run -e DS\_LICENSE=accept --name my-opscenter -d -p 8888:8888 datastax/dse-opscenter

Der Grund hierfür liegt daran, dass die einzelnen Cassandra Nodes im Anschluss erstellt werden. Bei der Instanziierung dieser individuellen Nodes wird jedoch der Management Node direkt an die Individual Nodes mitgegeben

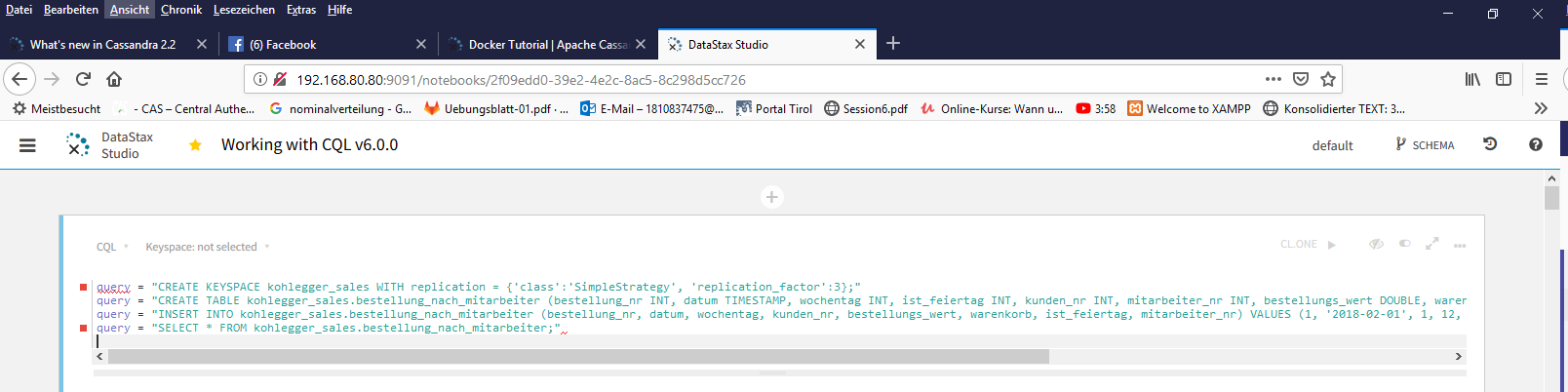
1. Starten eines einzelnen Data-Stax(Cassandra included) Server  
   docker run -e DS\_LICENSE=accept --link my-opscenter:opscenter --name my-dse -d datastax/dse-server  
   Dieser Schritt kann beliebig oft wiederholt werden, um beliebig viele DataStax Container dem Management node hinzuzufügen, sodass diese von dort aus verwaltet werden können, wichtig hierbei ist lediglich dass immer ein anderer Name der einzelnen Nodes verwendet wird
2. Ab nun ist aus dem Cluster aus dem Host über die Webtechnologien erreichbar



Über dieses Webinterface kann nun ein Cassandra Cluster entsprechend erzeugt, konfiguriert und gewartet werden.

Zugriff auf einen einzelnen Singelnode/

Wichtig ist, dass ein Container bereits läuft, jedoch kann dann über dieses Web-Interface mit den Nodes kommuniziert werden:



Die entsprechenden Container können unter

<https://hub.docker.com/r/jhc1990/dse-server>

und

<https://hub.docker.com/r/jhc1990/dse-studio>

eingesehen werden.

## Neo4j

Ein geeignete Neo4J Instanz kann analog wie der MSSQL-Container mittels des Befehles:

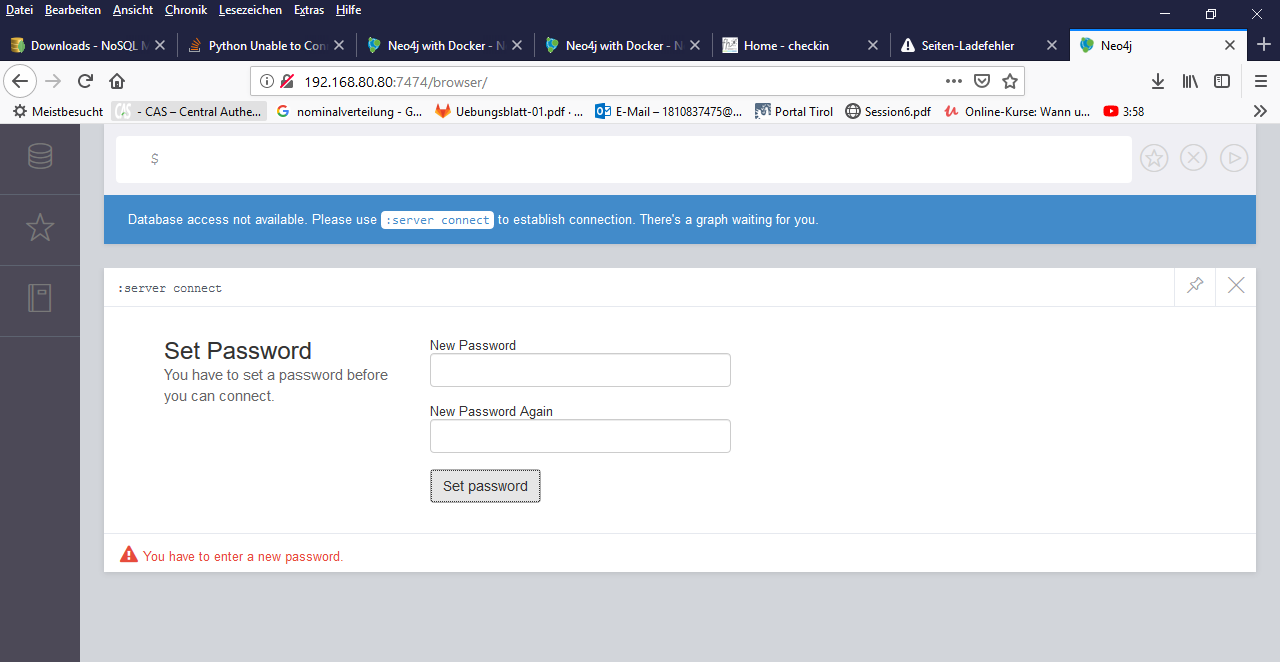
docker run \

--publish=7474:7474 \

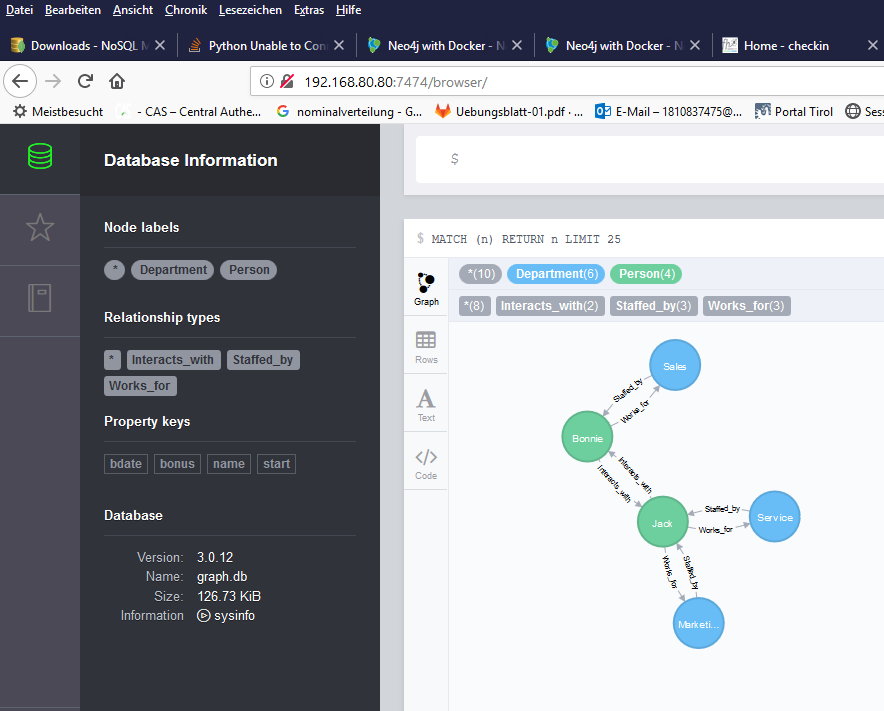
--volume=$HOME/neo4j/data:/data \

neo4j:2.3

Danach ist aus einem üblichen Browser die Management-area vom Neo4J Container ersichtlich



Sobald dieser Knoten steht, kann auch normal mit diesem interagiert werden:



Das entsprechende Image kann unter:

<https://hub.docker.com/r/jhc1990/neo4j_lab_dataeng>

eingesehen und heruntergeladen werden

MongoDB

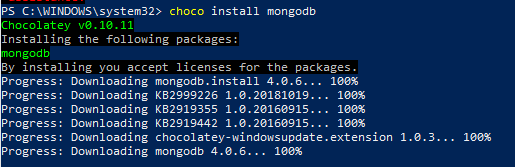
Als letzte Datenbanktechnologie hatten wir uns mit MongoDB im Lab beschäftigt. Analog zum bisherigen vorgehen können wir nun wieder das Image starten:

docker run -p 27017:27017 –-name mongodb -d mongo

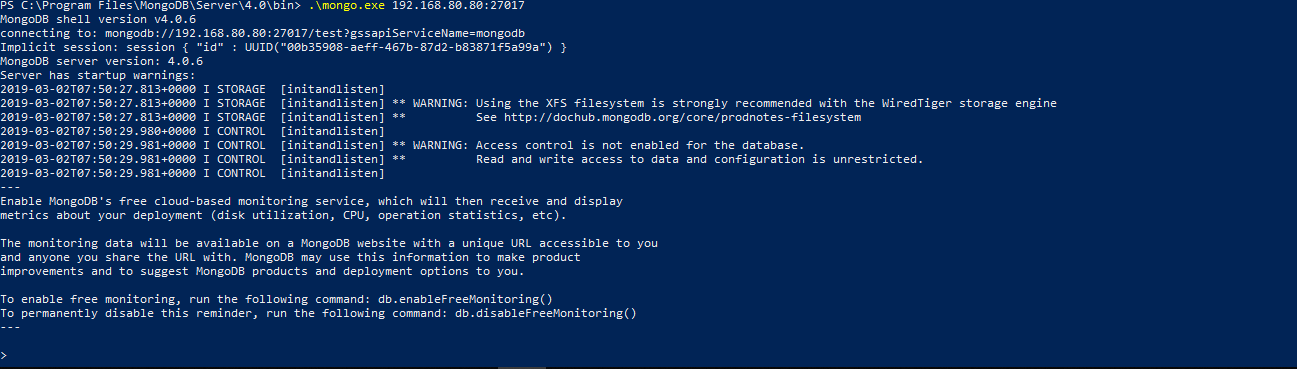
Für den Zugriff aus der Windows Powershell auf den MongoDB Server und den dortigen Container brauchen wir das CLI Programm MongoDB welches über die Zeile und dem Packetmanager Choco

choco install mongodb

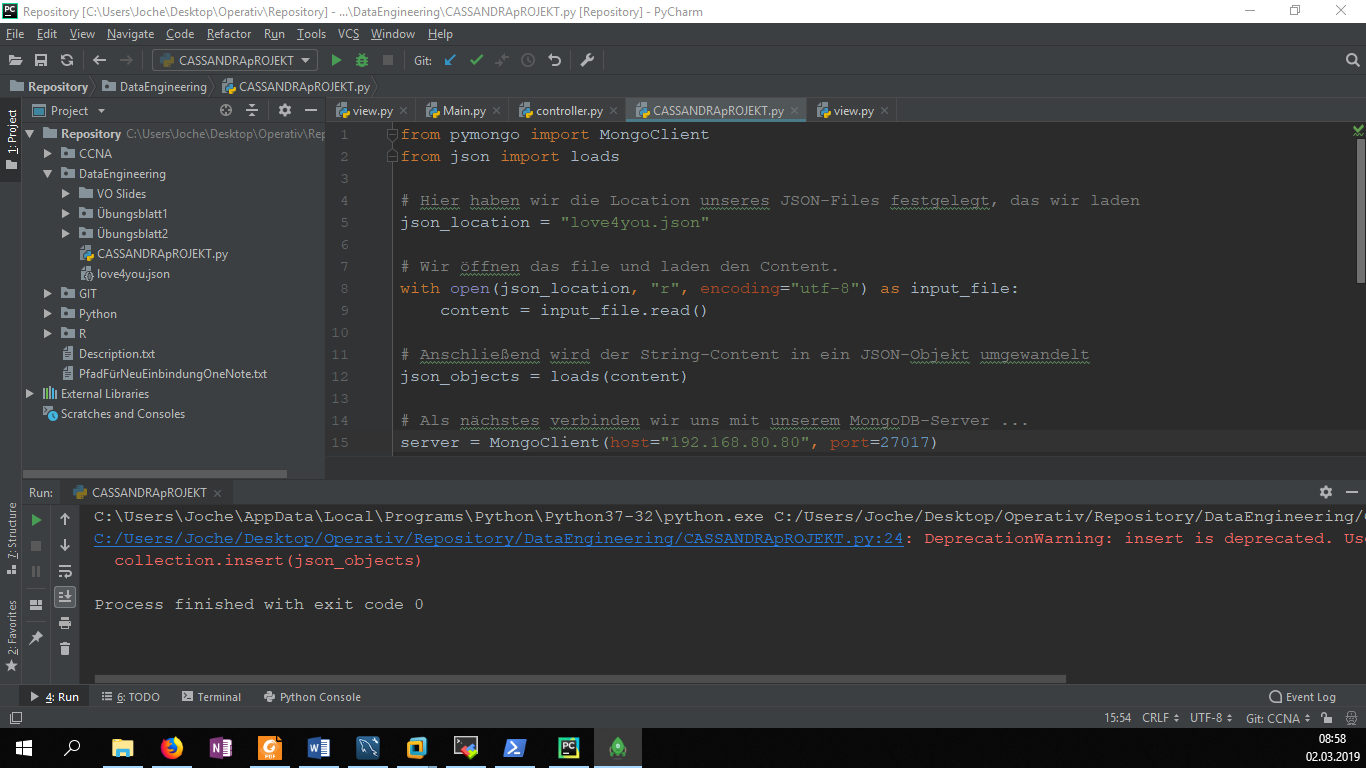
Beschafft wird:

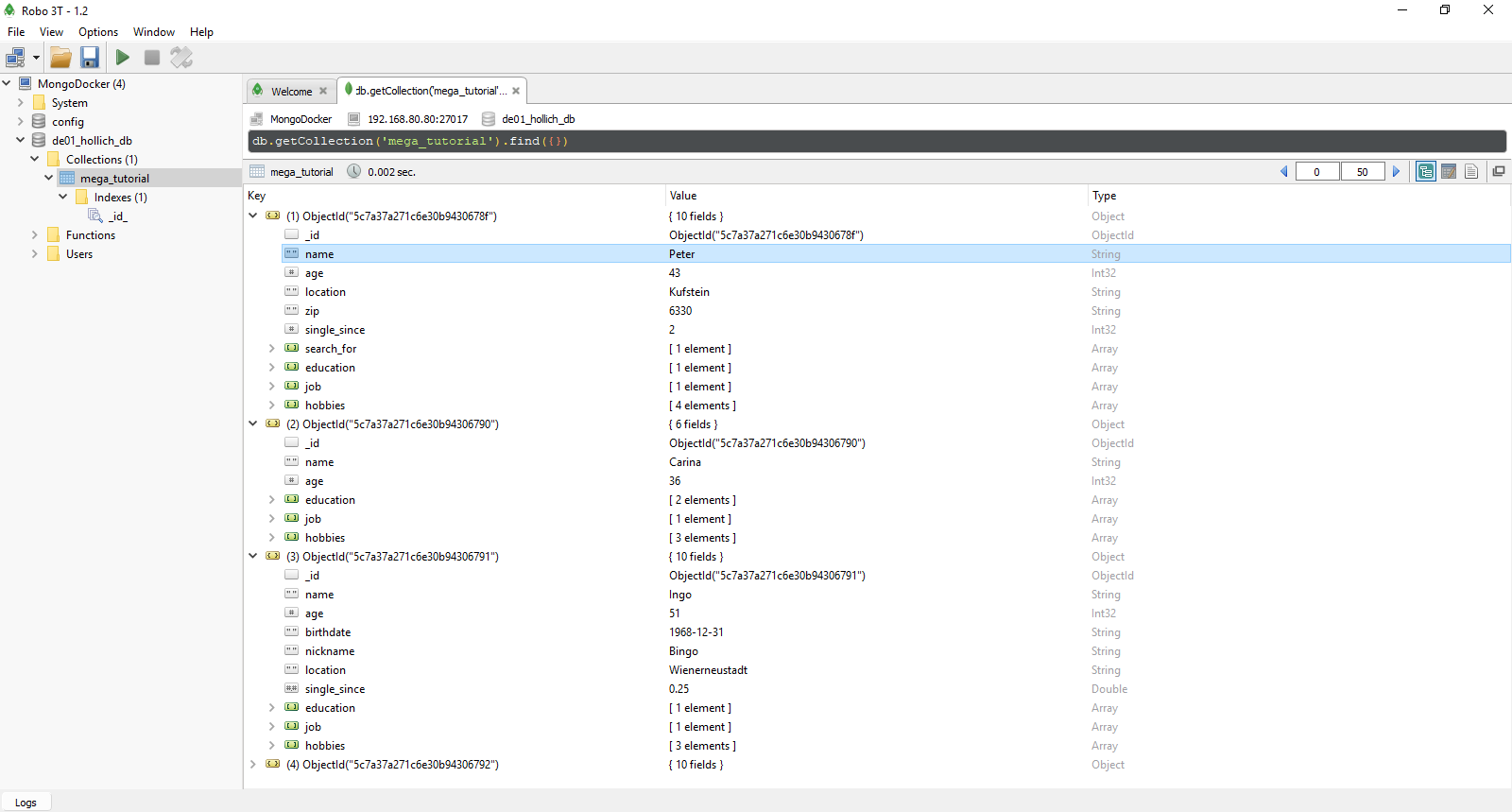


Nach dem erfolgreichen Start des Containers und der gegebenen lokalen Installation von MongoDB kann eine Verbindung hergestellt werden:



Außerdem kann nun auch via PyCharm die Verbindung vorgenommen werden:



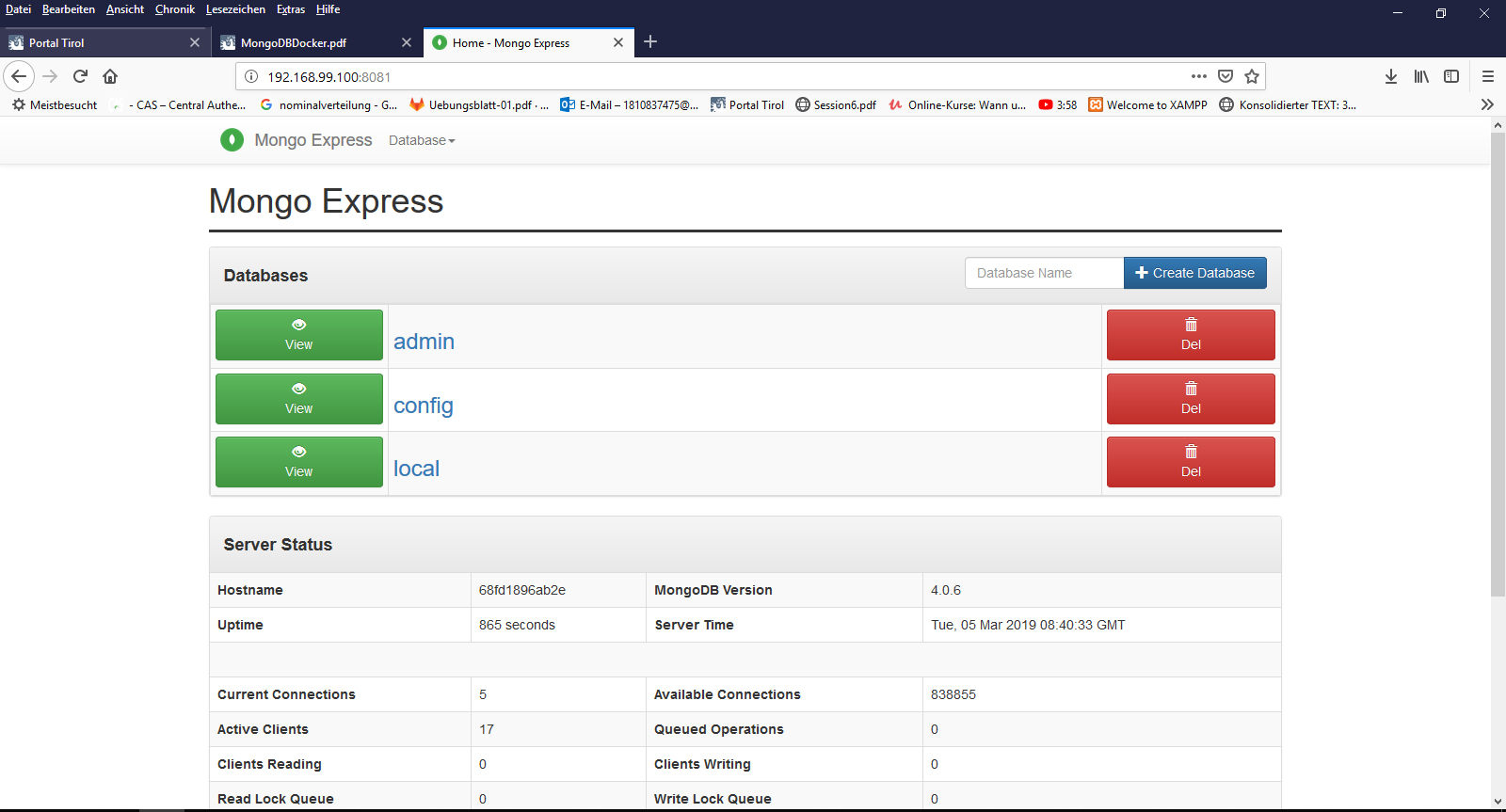
Und mittels des Managementtools „Robo-3T“: 

Netzer:

Es gibt auch ein Managementtool namens Mongo-express welches ebenfalls in einem Container gestartet wird und dann von außerhalb für die Verwaltung erreichbar ist:

docker run ––name mongoexpress --link mongodb:mongo -p 8081:8081 -d mongo-express

Davor muss ein Containerdb mit dem Namen Mongodb bereits laufen



Erstellung von Random JSON-files = JavaScriptObjectNotation

<https://www.json-generator.com/>

# Ausblick

In dieser Seminararbeit wurde ein Grundlegender Umriss des Themas „Umsetzung unterschiedlicher Datenbanktechnologien mittels Docker in einer Linux Umgebung“ umrissen. Eine vertiefte Anwendung der Thematik wäre das Docker Swarm Verhalten, in welchem mehrere Docker Container als Netzwerk fungieren und somit eine Ideale Grundlage für ein SQL-Server bzw. auch für eines Cassandra-Cluster bilden. Sowohl die traditionelle Technologie der relationalen SQL-Serverund die NoSQL Lösungen als auch die Innovative Container-Lösung Docker werden in der zunehmenden Datenflut des 21 Jahrhunderts eine immer zentralere Rolle spielen. Daher war es für den Studierenden durchaus sinnvoll sich mit diesen Fachgebieten näher zu beschäftigen.

# Quellen

1. Udemy Kurs

<https://www.udemy.com/docker-mastery/>

Netzer Install Docker Toolbox

Das ist Docker ohne Hyper V für Windows

<https://docs.docker.com/toolbox/toolbox_install_windows/>

Ergänzung mit den Slides vom Luaks // SoftDev Semester 2 Slides