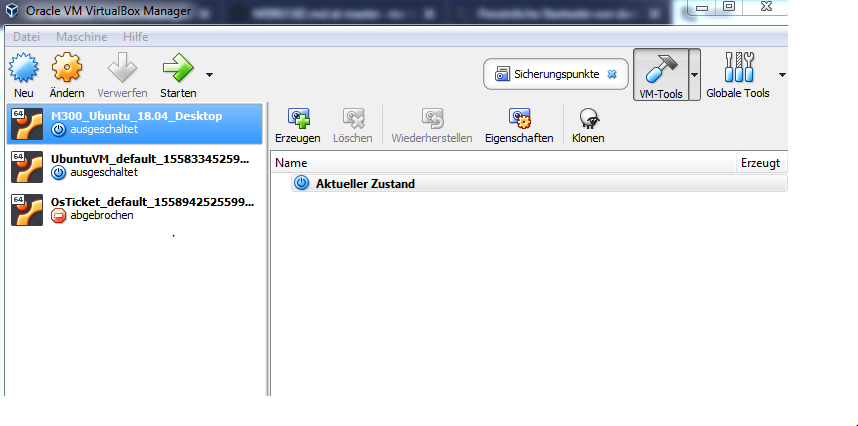
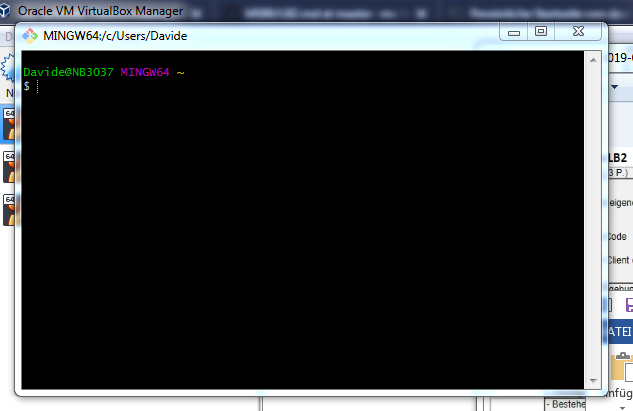
M300 LB3

**K1:**

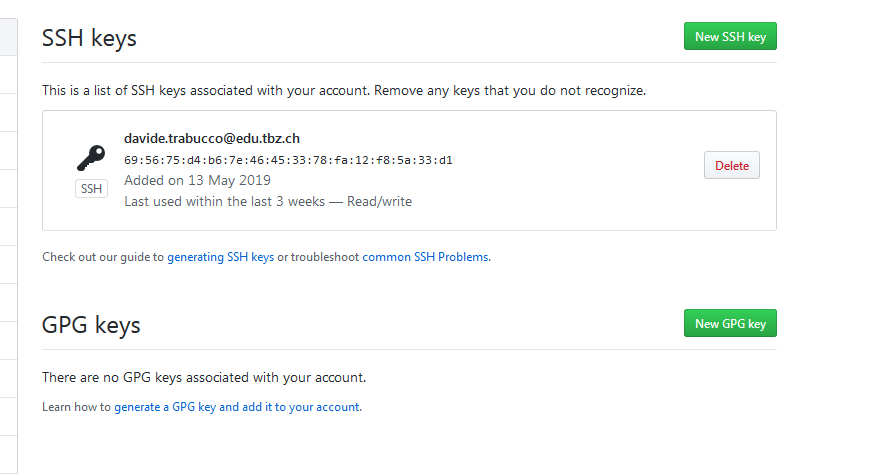
VirtualBox:



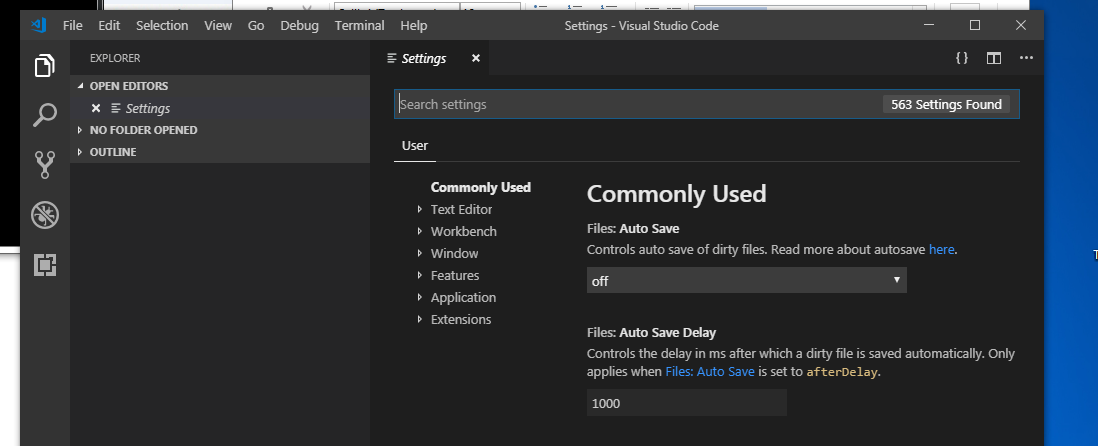
Git-Client:



Sshkey:

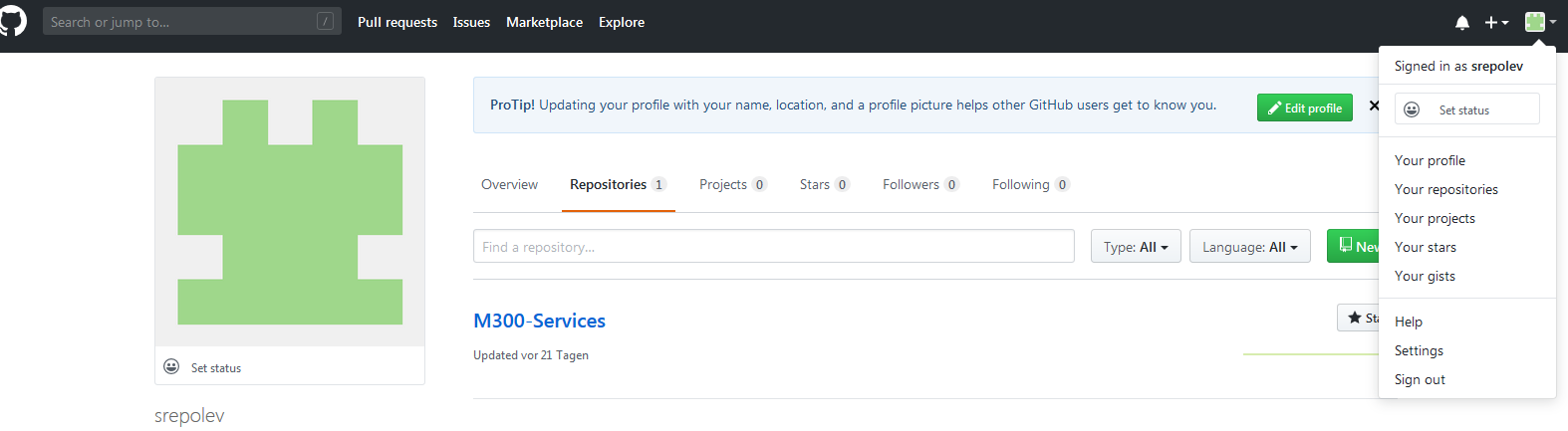


Visual Studio:



**K2:**

Github account:



Gitbash:

git config --global user.name "srepolev"

git config --global user.email [davide.trabucco@edu.tbz.ch](mailto:davide.trabucco@edu.tbz.ch)

git status # Geänderte Datei(en) werden rot aufgelistet

git add -A # Fügt alle Dateien zum "Upload" hinzu

git status # Der Status ist nun grün > Dateien sind Upload-bereit (Optional)

git commit -m "Mein Kommentar" # Upload wird "commited" >Kommentar zu ist notwendig

git status # Dateien werden nun als "zum Pushen bereit" angezeigt

git push #Upload bzw. Push wird durchgeführt

**Wissensstand**

**Containerisierung / Docker**

Die Container teilen sich mit dem host den Kernel und verwenden unter Linux ensprechende Kernelmodule um die Container von dem Host zu isolieren. Jedoch bietet dies keine hohe Isolierung wie bei Virtuellen Maschinen. Container beinhalten auch kein eigenes Betriebssystem

Vorteile: kurzen Aufstartzeiten, geringerer CPU-Overhead, Portierbar

**Microservices**

Microservices führen Applikationen/Services aus. Eine Applikation wird so weit aufgeteilt, dass viele kleine Services entstehen. Microservices können beliebig gestartet oder gestoppt werden ohne dass gleich die ganze Website betroffen ist. Z.B. wird bei einer Google Anfrage ein Microservice gestartet, wenn die Suche fertig ist wird er wieder gestoppt.

**K3**

Docker Befehle (Beispiele):

| **Befehl** | **Erklärung** | |
| --- | --- | --- |
| docker run | Container starten | |
| docker stop | Container Stoppen | |
| docker-compose up | Docker-compose Umgebung starten | |
| docker-compose down | | Docker-compose Umgebung stoppen. Entfernt anschliessend alle Container. |
| docker images | Heruntergeladene Images auflistet | |
| docker log | Logs eines Containers anzeigen | |
| docker ps | Listet container auf | |
| docker rm | Container Löschen | |

Voraussetzungen, damit die Container funktionieren:

**Host File** (da kein DNS)

192.168.60.101 test.ch

192.168.60.101 wordpress.test.ch

192.168.60.101 owncloud.test.ch

192.168.60.101 localhost

**Reverse Proxy erstellen**

docker network create reverse\_proxy

**Container starten**

docker-compose up –d

**Volumen einrichten**

docker volume create "Name\_des\_Volumen"

**K4**

1. Nestarts begrenzen:

restart: on-failure somit startet der Container nur neu wenn er crasht

1. Reverse Proxy  
   Reverse Proxy (Traefik) wurde für die verwendet.
2. Netzwerkzugriff beschränken  
   Bis auf die vom Reverse Proxy benötigten Ports keine weiteren Ports freigegeben. Die Kommunikation von aussen zu den Container geht somit ausschliesslich über den Reverse Proxy:

nmap localhost

Ausgabe:

Starting Nmap 7.70 ( https://nmap.org ) at 2019-07-01 10:44 CEST

Nmap scan report for localhost (127.0.0.1)

Host is up (0.000052s latency).

Not shown: 996 closed ports

PORT STATE SERVICE

80/tcp open http

443/tcp open https

8080/tcp open http-proxy

**K5**

**Vorwissen – Wissenszuwachs**

Grundsätzlich war das meiste neu. Ich weiss jetzt was ein Container ist und wie dieser aufgebaut ist. Ebenfalls habe ich gelernt was Microservices sind und wie sie eingesetzt werden. Ich kann eigene Container erstellen, habe Docke kennengelernt und weiss nun wie Docker verwendet wird um Container zu erstellen. Ich habe gelernt was Kubernetes ist und konnte einen eigenen kleinen Cluster erstellen. Auch habe ich gelernt wie man ein Reverseproxy einrichtet. Die ufw war etwas vom wenigen was ich schon kannte.

**Reflexion**

Das Modul hat mir Spass bereitet. Ich konnte einige neue Dinge lernen, die mir auch in der Praxis weiterhelfen können. Leider ging es bei mir vor allem am Anfang recht schleppend voran, weshalb ich gegen Ende ziemlich gestresst war.

**K6**

**Kubernetes Übung**

1. Github Repository lernkube klonen.

git clone https://github.com/mc-b/lernkube

1. config.yamk file ensprechend bearbeiten. Als Master node:

use\_dhcp: true

network\_type: public\_network

1. Mit dem Master node per ssh verbinden.

vagrant ssh master-01

1. Befehl anzeigen der auf dem Worker node ausgeführt werden muss:

sudo kubectl token create --print-join-command

1. VM verlassen
2. Skript ausführen, damit die Umgebungsvariablen so angepasst werden, dass der Master node auch von ausserhalb gesteuert werden kann.

source kubeenv

1. Deployment starten z.B. mit Node red.

kubectl apply -f duk/iot/nodered.yaml