Virtualized Web-Application with Docker and Kubernetes

W901

Yvo Keller & Gianna Huber

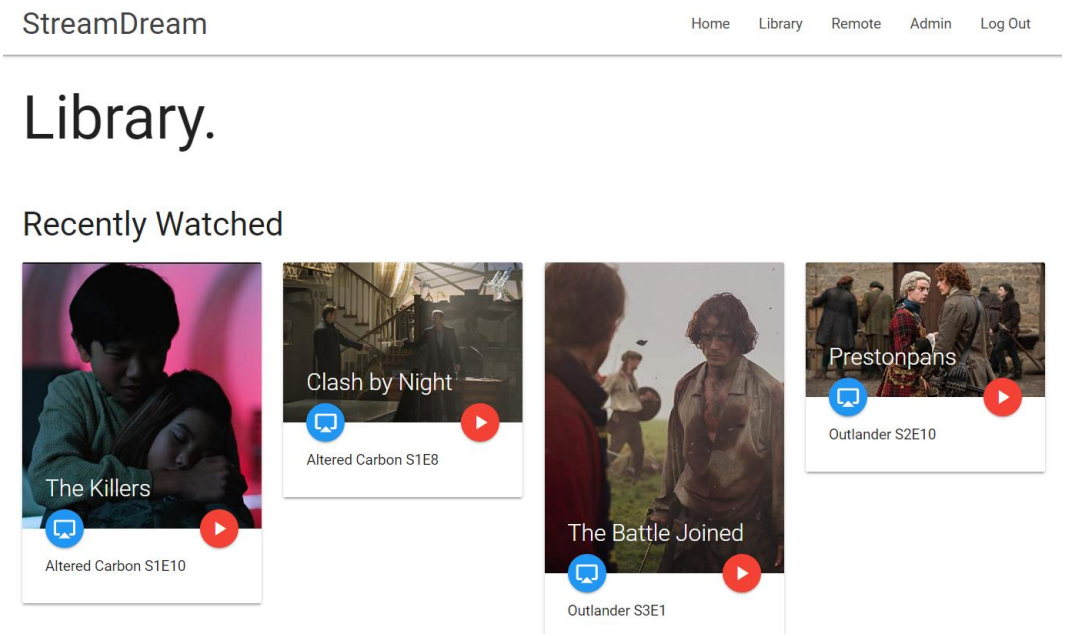
1 INHALTSVERZEICHNIS

2 INFORMIEREN

Bevor wir starten konnten, mussten wir uns zuerst in die verschiedenen Themen einlesen. Dafür haben wir die Dokumentationen vom BSCW sowie Provider Webpages gelesen und YouTube Videos geschaut. Auch haben wir immer wieder Ratschläge und Inputs von Herr Bernet bekommen, welche uns weitergeholfen haben.

2.1 IST-ZUSTAND

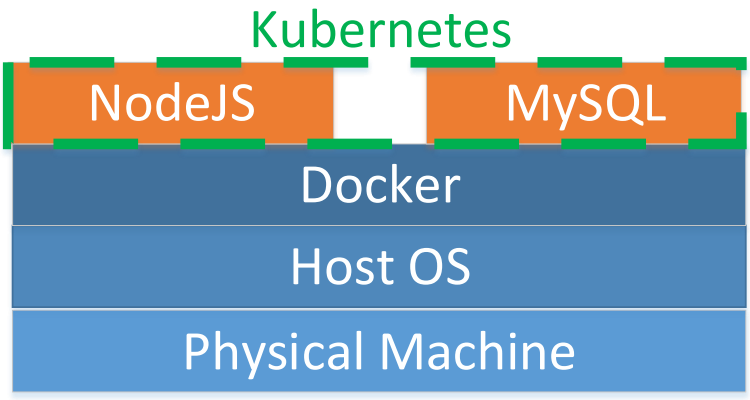
Die Applikation «StreamDream», welche von Yvo mit Node.JS entwickelt ist, verwendet eine MySQL Datenbank. Sie ermöglicht das Interaktive streamen von Filmdateien, die in einem spezifischen Ordner abgelegt sind. Eine Fernsteuerungsfunktion ermöglicht das steuern des Streams, beispielsweise vom Handy aus.



3 PLANEN

3.1.1 SOLL-Zustand

Die Webapplikation soll mit Hilfe von Docker und Kubernetes virtualisiert und skalierbar gemacht werden.



4 ENTSCHEIDUNG

4.1 MUSS UND KANN ZIELE

Muss	Kann
- Node Applikation (StreamDream) läuft in einem Container - MySQL Datenbankläuft in einem Container	- Kubernetes steuert die Container - Volumes werden durch Kubernetes gemanaged - CI/CD

4.2 STORAGE

Kubernetes bietet verschiedene Arten von Storage an. Wir haben hierbei die gewöhnlichen Volumes und Persistent Volumes genauer angeschaut.

Volumes sorgen dafür, dass Daten innerhalb eines Containers/Pod nicht verlorengehen, falls dieser abstürzt. Die Grundidee ist es, die Daten in einem solchen Volume unabhängig des Containers zu lagern. Hierfür bietet Kubernetes wieder verschiedene Volumes an, wie das emptydir, welches über den RAM/SSD des eigenen Nodes läuft. Auch gibt es verschiedene Cloud-Volumes, bei welchen die Daten in einer Cloud gelagert werden. Volumes haben den Nachteil, dass man alle kleinen Details, wie FS-Typ und Volume-ID kennen muss, falls man einen neuen Container konfigurieren möchte.

Dieses Problem kann umgangen werden, wenn man ein PVC – Persistent Volume Claim benutzt, welcher bei Persistent Volumes verwendet wird. Ein PVC kann ganz einfach von einem User erstellt werden. Über den PVC wird später eine Verbindung zum eigentlichen Storage, dem Persistent Volume, aufgebaut. Persistent Volumes können entweder Statisch oder Dynamisch sein, was ebenfalls im PVC definiert werden muss. Diese Verbindung wird BIND genannt und ist einzigartig. Man kann also nicht zwei PVC mit einem Persistent Volume verbinden. Man kann jedoch PVC für mehrere Container verwenden.

Da StreamDream eine stateful Application ist, haben wir uns für Persistent Volumes entschieden, da diese besser für solche Applikationen geeignet sind.

5 REALISIERUNG

5.1 BENÖTIGTE TOOLS

5.1.1 Docker Edge

Die Open-Source-Software Docker baut auf verschiedenen Linux-Techniken wie Namespaces auf mit dem Ziel, die Realisierung von Containern zu ermöglichen. Ebenfalls beinhaltet Docker den Online-Dienst Docker Hub, eine Registry für Images und Repositorys. Die Registry ist in einen privaten und in einen öffentlichen Bereich unterteilt. Im privaten Bereich lassen sich Images der Nutzer hochladen und zum Beispiel intern verteilen, ohne dass die Images öffentlich auffindbar sind. Mit Hilfe des öffentlichen Bereichs können Images auch externen Nutzern zur Verfügung gestellt werden. Mit der integrierten Versionsverwaltung von Docker, kann der aktuelle Zustand des Containers in einem Image gesichert werden. So kann für jedes Image zudem eine grobe Historie angezeigt werden.

Mit dieser Software wollen wir also versuchen unseren MySQL & Node.JS Container zu realisieren.

5.1.2 Atom

Wir haben uns dazu entschieden, Atom als unseren Editor zu verwenden.

5.1.3 Kubernetes

Kubernetes stellt alle Funktionen für die Steuerung unserer dockerized Application zur Verfügung. Dazu gehören folgende Features:

- Automatisierung von Containereinsatz und Software Rollout
- Optimierung der Nutzung von Computer-Ressourcen
- Zahlreiche Optionen für dauerhafte Datenspeicherung (Persistent Volumes)
- Integrierte Service Discovery
- (Auto-)Skalierung
- Hohe Verfügbarkeit

5.1.4 GitHub

Unser Projekt haben wir auf den GitHub Onlinedienst gestellt, um so beide darauf zugreifen zu können und immer auf dem aktuellen Stand zu bleiben.

5.2 BUILDING CONTAINERS

Um ein Image zu builden, aus dem anschliessend ein Container gestartet werden kann, wird ein Dockerfile benötigt. Dieses stellt die Grundlage des buildens dar und enthält alle relevanten Informationen dazu.

5.3 StreamDream

Im Prozess der Entwicklung haben wir durch dass, dass wir uns immer mehr Wissen aneignen konnten, mehrere Versionen des Dockerfiles für die Node.js Applikation im Einsatz gehabt. Dieses enthält die Spezifikationen zum Containerimage für die Node.js Applikation. Nachfolgend sind zwei Dockerfiles dokumentiert.

5.3.1 Version 1: GitHub als Quelle

Dockerfile

```
FROM node:latest
RUN git clone https://github.com/kelleryvo/StreamDream.git
WORKDIR /StreamDream
RUN git checkout dev --
RUN npm install
CMD node server.js
EXPOSE 8888
```

Obenstehendes Dockerfile holt sich die aktuellste Version der Applikation direkt aus dem GitHub Directory. Anschliessend werden die benötigten Node.js Pakete installiert, der Node.js Server gestartet und de Port 8888 veröffentlicht.

5.3.2 Version 2: CD (Continous Deployent)

Dockerfile

```
FROM node:latest
WORKDIR /app
COPY package.json /app
RUN npm install
COPY . /app
CMD node server.js
EXPOSE 8888
```

Obenstehendes Dockerfile liegt direkt im GitHub Repository von StreamDream und ermöglicht so ein automatisiertes builden des Images bei jedem GitHub Push. Diese Version ist am Ende zum Einsatz gekommen.

Beide des obenstehenden Dockerfiles können auch über die Konsole gebuildet werden. Das entstandene Image kann man dann starten. Dazu nutzt man die untenstehenden Befehle nutzt.

```
$ docker build -t streamdream-node .
```

```
$ docker run -it -p 8888:8888 --name streamdream-node-ct --link
streamdream-mysql-ct -e DATABASE_HOST=streamdream-mysql-ct streamdream-node
```

Der zweite Befehl startet einen Container aus dem Image streamdream-node und verbindet diesen mit dem MySQL Container, welcher zu diesem Zeitpunkt bereits laufen muss. So kann die Webapplikation (StreamDream) auf die MySQL Datenbank zugreifen. Auch der Hostname wird der Webapplikation in diesem Schritt übergeben.

5.4 MySQL

Der MySQL Container braucht kein eigenes Dockerfile, da das auf Dockerhub zur Verfügung stehende Image bereits alles unterstützt. Über Umgebungsvariablen kann das Root-Password definiert werden, welches auch der StreamDream Container kennt.

Der Befehl zum Starten sieht wie folgt aus:

```
docker run --name streamdream-mysql-ct -v
/Users/yvokeller/Development/github/StreamDream-
Virtualization/dockerized_apps/mysql/data/var/lib:/var/lib/mysql --env-file
/Users/yvokeller/Development/github/StreamDream-
Virtualization/dockerized_apps/mysql/env.list -d mysql:5.7
```

Hier ist relevant, dass ein Volume angegeben wird, also ein Datenspeicher, der im Container unter `/var/lib/mysql` gemountet wird. Dadurch speichert der MySQL Server die Daten auf dem Host-Computer. So bleiben diese auch nach einem Neustart des MySQL Containers erhalten.

5.5 Kombinieren

Diese beiden Container kombiniert haben ein starten der Webapplikation in zwei einzelnen Containern ermöglicht, genau wie vorgesehen. Über den vom Node.js Container freigegeben Port kann man unter `localhost:8888` auf die Webapplikation zugreifen und diese vollumfänglich verwenden.

5.6 Kubernetes aufsetzen

Kubernetes ist nach der Virtualisierung mit Docker das nächste Level. Sogenannte „Deployments“ Kontrollieren, wie viele Container eines bestimmten Typs laufen, startet diese im Falle von Crashes automatisch neu und ermöglicht „Rolling Updates“.

Basis der Deployments sind die yaml-Files. Hiervon mussten wir je eines für die Webapplikation und den MySQL Server erstellen.

5.6.1 StreamDream

Das Deployment File definiert in diesem Fall folgende wichtige Dinge:

- Die Applikation steht unter Port 8888 zur Verfügung
- Es sollen 5 Replikate des Containers im Einsatz sein
- Umgebungsvariablen geben der Applikation die nötigen Informationen zum Verbinden mit der Datenbank
- Das Volume der Host Machine, auf welchem die Filmdateien zur Verfügung stehen, wird gemountet.
- Quelle für den Container ist das itsfrdm/streamdream-node Image der Version 1.0.0 vom DockerHub.

5.6.2 streamdream.yaml:

```
apiVersion: v1
kind: Service
metadata:
  name: streamdream-node
  labels:
    app: streamdream
spec:
  type: LoadBalancer
  ports:
  - port: 8888
    protocol: TCP
  selector:
    app: streamdream
---
apiVersion: apps/v1beta2 # for versions before 1.8.0 use apps/v1beta1
kind: Deployment
metadata:
  name: streamdream-node
  labels:
    app: streamdream
spec:
  replicas: 5
  selector:
    matchLabels:
      app: streamdream
      tier: nodejs
  template:
    metadata:
      labels:
        app: streamdream
        tier: nodejs
    spec:
      containers:
      - name: streamdream
        image: itsfrdm/streamdream-node:1.0.0
        env:
        - name: DATABASE_HOST
          value: streamdream-mysql
        - name: DB_PASSWORD
          value: pw1mysql
        ports:
        - containerPort: 8888
          name: streamdream
        volumeMounts:
        - name: streamdream-data
          mountPath: /StreamDream/data
      volumes:
      - name: streamdream-data
        hostPath:
          path: /Users/yvokeller/Development/StreamDream/data
          type: Directory
```

5.6.3 MySQL

Das Deployment File definiert in diesem Fall folgende wichtige Dinge:

- Der MySQL Server steht unter dem Port 3306 zur Verfügung
- Umgebungsvariablen definieren das Root-Passwort
- Ein Persistent Volume Claim der Grösse von 5GB wird vom MySQL Container verlangt. Darin werden alle Datenbankrelevanten Daten gesichert.
- Quelle für den Container ist das mysql/mysql:5.7 Image vom DockerHub.

5.6.4 apiVersion: v1

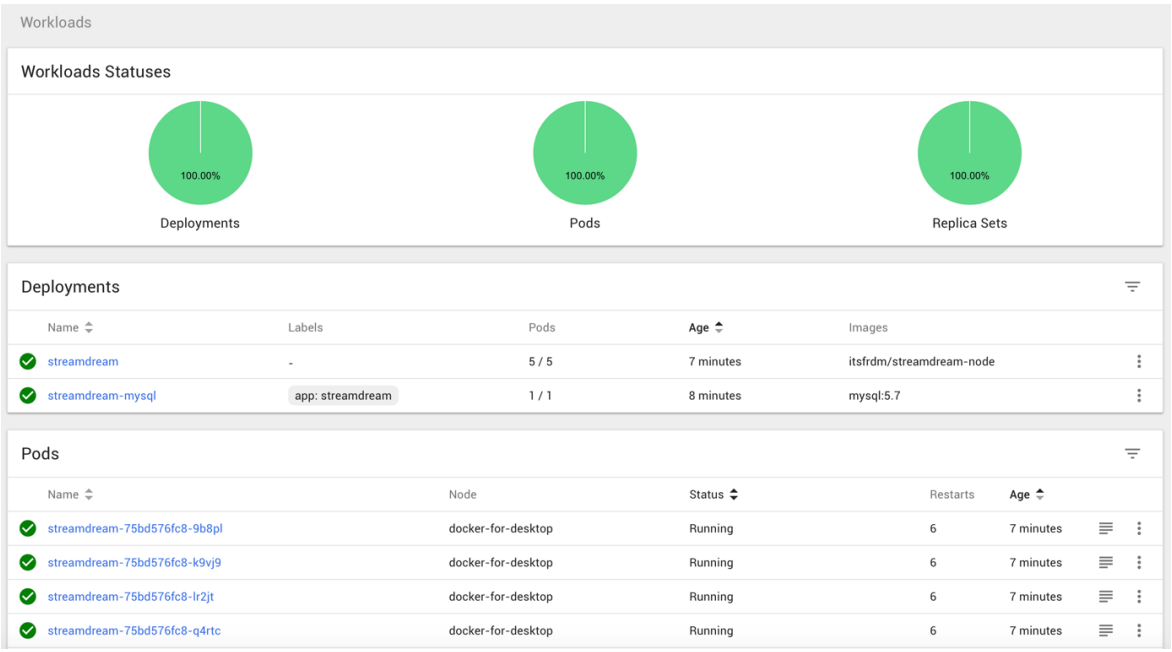
```
kind: Service
metadata:
  name: streamdream-mysql
  labels:
    app: streamdream
spec:
  ports:
    - port: 3306
  selector:
    app: streamdream
    tier: mysql
  clusterIP: None
---
apiVersion: apps/v1beta2 # for versions before 1.8.0 use apps/v1beta1
kind: Deployment
metadata:
  name: streamdream-mysql
  labels:
    app: streamdream
spec:
  selector:
    matchLabels:
      app: streamdream
      tier: mysql
  strategy:
    type: Recreate
  template:
    metadata:
      labels:
        app: streamdream
        tier: mysql
    spec:
      containers:
      - image: mysql:5.7
        name: mysql
        env:
        - name: MYSQL_ROOT_PASSWORD
          value: pw1mysql
        ports:
        - containerPort: 3306
          name: mysql
        volumeMounts:
        - name: streamdream-mysql-storage
          mountPath: /var/lib/mysql
      volumes:
      - name: streamdream-mysql-storage
        PersistentVolumeClaim:
          claimName: streamdream-mysql-pv-claim
---
```

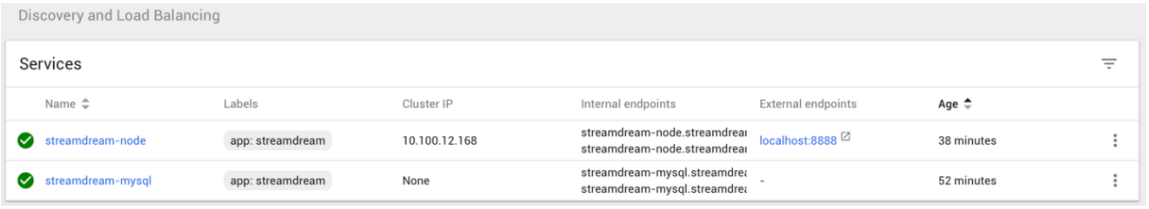
```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: streamdream-mysql-pv-claim
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 5Gi
```

5.7 Applikationen „Deployen“

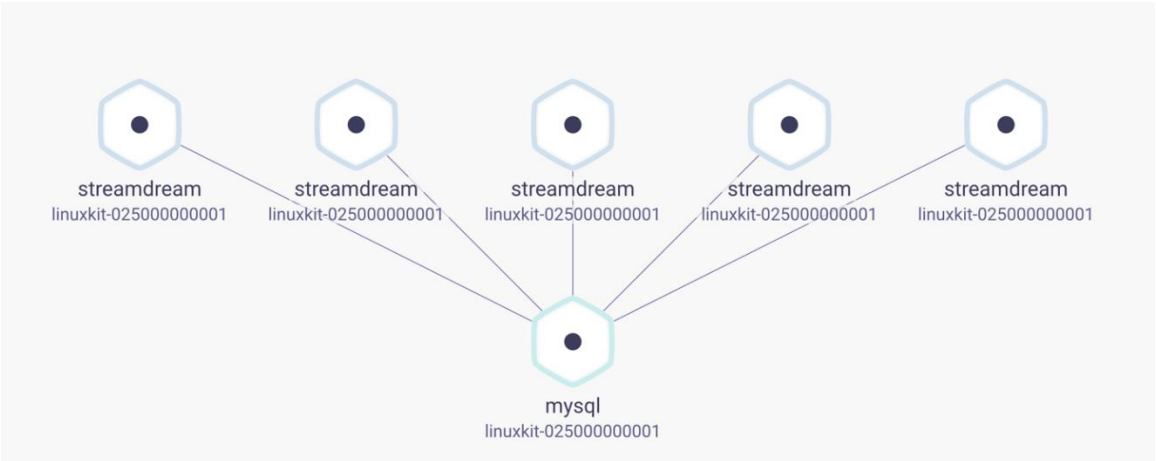
Deployt man die Applikationen in Kubernetes auf Grundlage der .yaml-Files, wird wie vorgegeben alles gestartet.

Das Resultat sieht man im Kubernetes Control Panel:



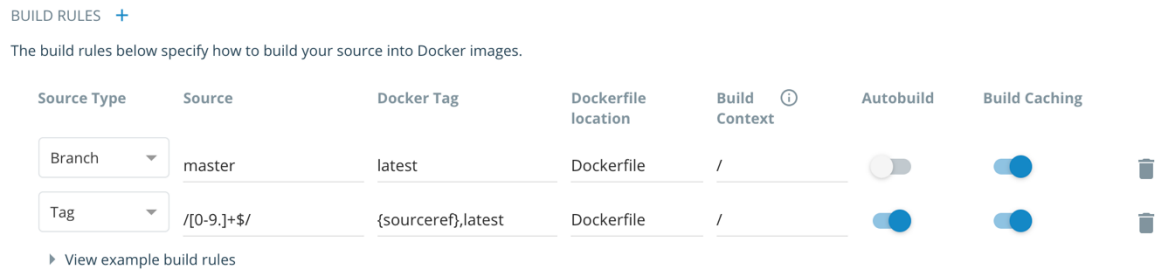


WeaveScope visualisiert die Verbindung der Container Verbindung sehr gut sichtbar:

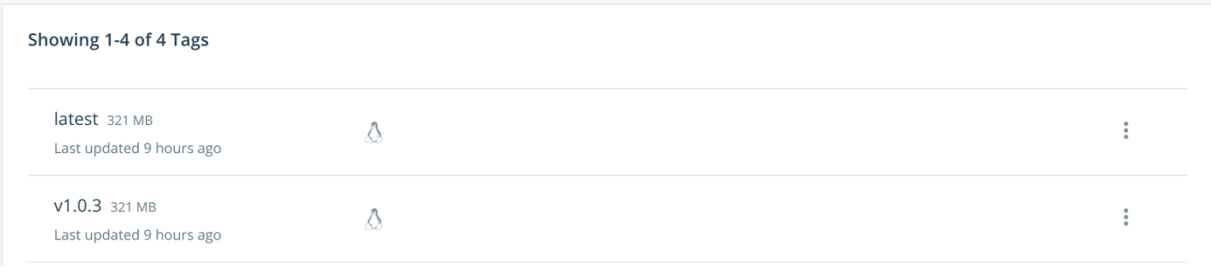


5.8 CI/CD

Damit das Docker Image bei jedem Push mit Versionstag auf das GitHub Directory von StreamDream automatisch gebuildet wird, haben wir im Docker Repository „Automated Builds“ aktiviert. Einstellungen siehe Screenshot.



Dank dieser Konfiguration wird nun, sobald eine neue Version von StreamDream gepusht wird, das dazugehörige Image gebuildet. In Kubernetes kann man anschliessend problemlos mit Hilfe eines Rolling Updates auf die neue Version wechseln.



6 KONTROLLE

Testfall / Erwartetes Resultat	Resultat
Webapplikation StreamDream kann über Container gestartet und verwendet werden	OK
MySQL kann über Container gestartet und verwendet werden, die Daten sind dabei persistent.	OK
Kubernetes kontrolliert mit Hilfe von Deployments die Container und ermöglicht Rolling Updates, Skalierung sowie Ausfallsicherheit durch selbstheilende Container.	OK

7 AUSWERTUNG

7.1 WO STEHEN WIR

7.1.1 Implementierung & Problemlösung

Anfangs haben wir viel Zeit benötigt um uns über die verschiedenen Technologien und Tools zu informieren, da wir mit diesen noch nicht vertraut waren. Erst nach dieser Phase haben wir uns für das effektive Projekt, Dockerized Web Application entschieden. Anschliessend haben wir die Aufgaben aufgeteilt und begonnen unsere Umgebung aufzubauen. Probleme hatten wir vor allem bei der Entscheidung, ob wir jetzt Volumes oder Persistent Volumes verwenden sollen, dies hat sich nach Recherche und erstmaliger Implementierung aber ergeben. Ansonsten verlief es mehr oder weniger reibungslos. Daraus kann man schliessen, dass das Projekt erfolgreich war.

7.1.2 Kollaboration

Die Zusammenarbeit hat gut funktioniert. Wir haben vor allem darauf geachtet, dass jeder das macht was seinen Stärken entspricht.
Yvo hat bereits mehr Verständnis für solche Umgebungen, weshalb er mir viel erklären musste, damit ich es verstehe. Nachdem er mir die mir unbekannten Dinge erklärt hat, habe ich mein Bestes gegeben um bei der Umsetzung des Projekts zu helfen. Dies hat so ganz gut funktioniert, auch wenn Yvo nicht so viel von meinem Wissen profitieren konnte, wie ich von seinem.

7.1.3 Vorgehen Lernen

Um an die benötigten Informationen zukommen, haben wir viel selber recherchiert. Das meiste Wissen haben wir uns also durch online Dokumentationen und YouTube Videos angeeignet. Bei Unsicherheiten konnten wir uns aber immer auf die Hilfestellung von Herr Bernet verlassen. Mit seinen Tipps und Inputs sind wir immer gut weitergekommen und hatten neue Ideen für unser Projekt.