Hauptstudie Umsetzung VA

Niklas Liechti

Inhaltsverzeichnis

1. Management summary

Das Projekt soll Lehrpersonen das benützen der vorhandenen Kubernetes Cluster vereinfachen.

Das Projekt ist auf Github gehostet, wo auch diese Dokumentation als Asciidoc verfügbar ist.

Dort finden sich auch noch weitere dokumentationen dazu, wie die Applikation zu Deployen, zu benutzen und mit Inhalt zu beliefern ist.

1.1. Funktion

Die Applikation selber erlaubt ein einfaches bereitstellen von Applikation auf pro Schüler ebene. Das heisst die Lehrperson kann mit wenigen klicks 25 OS Ticket instanzen hochfahren.

1.2. Technologien

Das Projket wurde mit folgenden Technologien umgesetzt:

- Frontend: Vue.js
- Backend: Javalin
- Datenbank: Nitrite

1.3. Deployment

Die applikation kann als Docker Container von Dockerhub heruntergeladen und deployt werden.

Dies kann in dem zu managenden Kubernetes Cluster geschehen, oder auch Standalonw von aussen.

2. Architektur

Die Architektur ist eine recht einfach gehaltenes Frontend welches über eine Rest API mit dem Backend kommuniziert.

2.1. Frontend

Da ich das Frontend recht simple halten wollte, habe ich mich dafür entschieden die Komponenten fast komplett voneinander zu trennen. Die einzigen Komponenten, die mehrfach verwendet werden, sind die Navigations komponenten. Auch verwende ich nicht den Vue router, daher wird ganz klassisch per "full page reload" navigiert.

2.2. Backend

Das Backend ist mehr oder weniger klassisch aufgebaut. Im Main.kt sind alle rest endpoints in einzelnen Funktionen definiert. Dort sind diese dann mit Controllern verlinkt.

Die Controller übernehmen dann die basic funktionalitätet und einen Teil der Logik.

Komplexe und ab kapselbare Logik ist dann in Services ausgelagert, die von den Controllern aufgerufen werden.

Der Persistenz Layer ist über sogenannte Repositories verfügbar. Dies können Implementationen sein, welche auf eine DB gehen, oder auf den Kubernetes cluster.

3. Technologien

3.1. Frontend

Aktuell ist das Frontend immer noch massiv von Javascrip dominiert. Langsam kommen Konkurenten wie Web Assembly hervor. Diese sind aber noch viel zu Jung und zu wenig verbreitet um momentan für eine Applikation ein zu setzten.

Daher habe ich mich dazu entschieden noch mit einem Javascript Framework weiter zu machen.

Frontend Javascript Frameworks sind seit Jahren im starken Wandel.

3.1.1. Framework

Im Frontend bereich habe ich von den "modernen" Frameworks nur Erfahrung in Angular.js, welches aber schon seit einigen Jahre eigentlich überholt ist.

Daher habe ich mich bei der Entscheidung an einen Blogpost von dev.to und Erfahrung von Arbeitskollegen angelehnt.

Angular 2+

Angular ist das nachfolge Framework von angular.js, welches weiterhin von Google maintained wird.

Mit Angular habe ich minimale Erfahrung durch einen 1 Monatigen einsatz in einem kleineren Projekt.

Wie unten ersichtlich ist das Lernen und aufsetzen eines Angular Projektes relativ komplex. Grundsätzlich ist es extrem gut einsetzbar für grössere Singlepage apps. Die vorgegebene Struktur ist ein grosser Vorteil für Projekte mit vielen Personen, was in diesem Projekt aber nicht relevant ist.

Figure 1. Angular

Vue.js

Vue.js ist das neuste aller Frameworks. Es ist sehr einfach zu lernen und grundsätzlich sehr leichtgewichtig. Es ist ähnlich reaktiv wie react.js und nutzt Web components excessiv.

Figure 2. Vue.js

React.js

React.js ist eine möglichst schlank gehaltene Frontend Library, welche von Facebook maintained wird. Das Konzept ist wie der Name schon sagt reaktives verhalten. Das heisst man beschreibt grundsätzlich welche Daten dargestellt werden sollen und das Framework kümmert sich dann um das "reagieren" auf changes des stetes.

Figure 3. React.js

3.1.2. Entscheidung

Das am einfachsten auf zu setzende Framework ist in meinem Fall Vue, da es direkt von meinem Backend Framework unterstützt wird. Da sich alle Frameworks für meinen Anwendungsfall nicht viel nehmen, habe ich mich für Vue.js entschieden.

3.2. Backend

Das Backend ist grundsätzlich seit jeher stabiler, da diese Platform meist voll unter Kontrolle des betreibenden liegt. Diese wahl wurde in den Letzten Jahren aber auch immer flexibler, da mit Docker fast jede beliebige Betriebssystem Umgebung gebaut werden kann.

Die meiste Erfahrung habe ich selber mit Java EE Applikationne, welche aber hier ein zu grosses Kaliber wären.

Zusätzlich habe ich in letzter Zeit viel Positives über Kotlin gehört. Kotlin ist eine auf der Jvm basierte Sprache, die nahe an Java ist, aber sehr viel Syntactic Sugare anbietet und das Entwickeln von genau solchen kleinen Applikationen extrem einfach macht.

3.2.1. Framework

Als Framework habe ich mir nach kurzem recherchieren und nach Feedback von Arbeitskollegen Javalin ausgesucht.

Javalin ist ein leichtgewichtiges Microservice Framework, welches mit nativ Kotlin unterstützung kommt.

Auch bietet Javalin integrierter Support für Vue.js komponenten, was natütlich ein sehr praktischen Vorteil ist.

3.3. Database

Um Datenbank migrationen für so ein kleines Projket zu Vermeiden setzte ich auf ein NosqlDB.

Nach kurzer suche bin ich auf Nitrite gestossen

3.4. Deployment

Da das Deployment der Applikation auf einem Standart TBZ Cluster laufen muss, war das Deployment als Docker Image fix.

3.5. Buildsystem

Als Buildsystem für das Projekt habe ich Gradle mit der Kotlin DSL eingesetzt. Dieses Buildsystem ist sein einigen Jahren Standard im Java Universum. Mit gradle builde ich ein FatJar mit dem ShadowJar plugin, welches dann in ein Docker Image verpackt wird.

3.5.1. Dependencies

Gradle macht auch das Dependencie management. Die dependencies sind hier zu sehen.

3.6. CI / CD

Da bei einer so komplexen Umgebung ein Continous integration nur mit viel Aufwand möglich ist, gibt es dies hier nicht.

Für den Continous delivery Teil sorgt Travis, welcher bei jedem Push auf das Repo ein latest Docker image baut und dieses zu Dockerhub pusht. Sobald ein Commit ge tagt ist, wird zusätzlich ein fixe Version released.

4. Technische Voraussetzungen

4.1. Kubernetes Cluster

Um die Applikation zu deployen müssen Persistent Volumes mit der storage class `local-storage` vorhanden sein.

Von diesen müssen genug vorhanden sein, damit jede Instanz ein volume claimen kann. Das heisst `Anzahl Deployments x Anzahl VolumeClaims pro deployment`

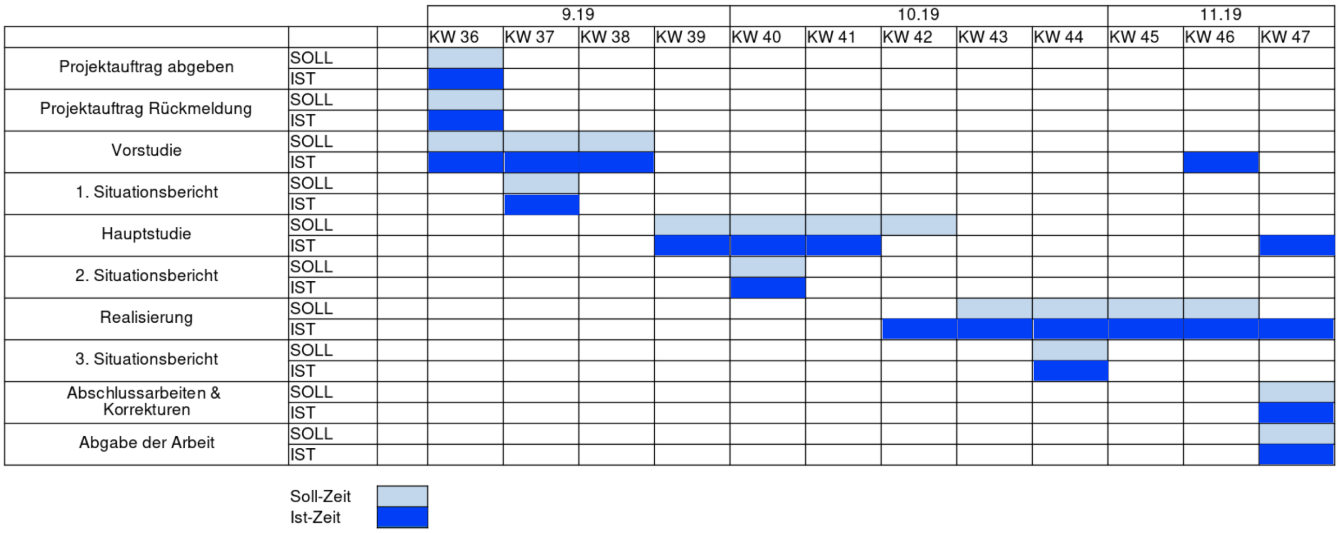
```
kind: PersistentVolume
apiVersion: v1
metadata:
  name: rwo-volume-02
  labels:
    type: local
spec:
  storageClassName: local-storage
  persistentVolumeReclaimPolicy: Recycle
  capacity:
    storage: 50Gi
  accessModes:
    - ReadWriteOnce
  hostPath:
    path: "/data"
```

5. Controlling

5.1. Zeitplan

Grundsätzlich konnte der Zeitplan relativ gut eingehalten werden. Aufgrund von feedback vom Auftraggeber habe ich die Vorstudie und Hauptstudie immer wieder leicht angepasst.

Die Dokumentation wurde im ininitalen Zeitplan nicht wirklich berücksichtigt, erfolgete aber währen der gesammten umsetzungszeit fortlaufend.



5.2. Ticket Tracking

Ich habe bei dem Projekt auch versucht mit github tickets den Fortschritt des Projektes zu Tracken.

Dies hat in diesem Projekt mit sehr beschränkten Zeit mitteln nur bedingt funktioniert. Es war aber grundsätzlich nicht schlecht als Motivation, zu sehen was ich doch bereits alles gemacht habe. Vor allem im Infrastrukturellen bereich am anfang ist es schwierig den Überblick über den Fortschritt zu halten.

Auch konnte diese Variante nicht 100% ausgespielt werden, da ich komplett alleine an dem Projekt ge arbeitet habe.

5.3. Applikationstest

Table 1. Testfälle

Testfall	Test Konfiguration	Ausgang
Mails verschicken	Gmail smtp mit privatem account verwendet.	Funktioniert mit der über die ENV mitgegebene config für einen SMTP Server
CSV Von klasse hochladen	Testfile mit anonymisierten daten getestet. Direkt von ecolm.com	Verschickt Mail einwandfrei
Deployment auf Kubernetes	Deployt auf einem Minikube lokal	Deployment funktioniert mit den Zugangsdaten über ENV gesetzt

6. Schwierigkeiten bei der Umsetzung

6.1. Travis CI

Die Dokumentation, wie Credential hochgeladen werden können hat micht doch eine gute Stunde gekostet. Gut versteckt im UI können diese z.B. zum pushen von images hinterlegt werden.

6.2. Deployment von Javalin in Docker

Aus bisher immer noch unerklärlichen gründen, hat Javalin im Container die resourcen im Classpath an einem anderen Ort als auf einem Mac oder einem normalen Linux.

Dies konnte ich aber mit einem workaround fixen.

```
[main] INFO io.javalin.Javalin - Starting Javalin ...
[main] INFO io.javalin.Javalin - Listening on http://localhost:7000/
[main] INFO io.javalin.Javalin - Javalin started in 193ms \o/
[qtp715521683-36] WARN io.javalin.Javalin - Uncaught exception
java.nio.file.NoSuchFileException: src/main/resources/vue
at java.base/sun.nio.fs.UnixException.translateToIOException(UnixException.java:92)
at java.base/sun.nio.fs.UnixException.rethrowAsIOException(UnixException.java:111)
at java.base/sun.nio.fs.UnixException.rethrowAsIOException(UnixException.java:116)
at java.base/sun.nio.fs.UnixFileAttributeViews$Basic.readAttributes(UnixFileAttributeViews.java:55)
at java.base/sun.nio.fs.UnixFileSystemProvider.readAttributes(UnixFileSystemProvider.java:145)
at java.base/sun.nio.fs.LinuxFileSystemProvider.readAttributes(LinuxFileSystemProvider.java:99)
at java.base/java.nio.file.Files.readAttributes(Files.java:1763)
at java.base/java.nio.file.FileTreeWalker.getAttributes(FileTreeWalker.java:219)
at java.base/java.nio.file.FileTreeWalker.visit(FileTreeWalker.java:276)
at java.base/java.nio.file.FileTreeWalker.walk(FileTreeWalker.java:322)
at java.base/java.nio.file.FileTreeIterator.<init>(FileTreeIterator.java:71)
at java.base/java.nio.file.Files.walk(Files.java:3820)
at io.javalin.plugin.rendering.vue.JavalinVue.walkPaths$javalin(JavalinVue.kt:33)
at io.javalin.plugin.rendering.vue.VueComponent.handle(JavalinVue.kt:63)
at io.javalin.core.security.SecurityUtil.noopAccessManager(SecurityUtil.kt:22)
at io.javalin.http.JavalinServlet$addHandler$protectedHandler$1.handle(JavalinServlet.kt:116)
at io.javalin.http.JavalinServlet$service$2$1.invoke(JavalinServlet.kt:45)
at io.javalin.http.JavalinServlet$service$2$1.invoke(JavalinServlet.kt:24)
at io.javalin.http.JavalinServlet$service$1.invoke(JavalinServlet.kt:123)
at io.javalin.http.JavalinServlet$service$2.invoke(JavalinServlet.kt:40)
at io.javalin.http.JavalinServlet.service(JavalinServlet.kt:75)
at javax.servlet.http.HttpServlet.service(HttpServlet.java:790)
```

6.3. Libraries

Initial hatte ich einige Version konflikte, was ich mit Gradle noch nie hatte. Auch dies hat mich mindestens 1-2 Stunden gekostet, bis ich jede library zufriedenstellen konnte.

```
Yaml parsing without reflection with `https://github.com/charleskorn/kaml`
`https://github.com/Kotlin/kotlinx.serialization/`
Did not work in combination with `io.fabric8:kubernetes-model:4.6.0`
```

Gelöst mit commit und dem `constraints` block gleich darunter.

6.4. Kubernetes models

Das Typensichere laden eines Kubernetes configfiles war schwerer als gedacht, da die Offiziellen models kein wirklich generisches Parsen zulassen.

Dafür ist der fabric8io client mit models um so besser, wenn man das konstrukt mal verstanden hat.