1 Dokumentation - CLC3 - Projekt

Marko Gattringer, Christoph Ruhsam

1.1 Kurzbeschreibung

Im Zuge meiner Masterarbeit (Marko Gattringer) wird unter anderem ein Algorithmus entwickelt, welcher einen gezeigten Workflow automatisch aus einem Screencastvideo extrahieren kann. Da dieser Algorithmus sehr rechenintensiv ist (Neuronales Netz mit Tensorflow-GPU) und das Setup dazu nicht trivial ist (Linux, CUDA, Tensorflow-GPU, Python, ...), soll dieser als Service angeboten werden. Dazu soll das Service als Microservice angeboten werden und in Openshift (https://www.openshift.com/) gehostet werden. Weiters dient ein kleine Webapp als Frontend (Login, Verwalten seiner Videos, ...). An diesem Beispiel soll gezeigt werden, wie so eine Applikation in Openshift betrieben werden kann. Dabei werden auch Konzepte wie zBsp.: „Infrastructure-as-Code“ oder „Continuous deployment“ umgesetzt.

2 Ziel / erwartetes Ergebnis

Microservice-Architektur, welche Lokal (zu Test- und Entwicklungszwecken) als auch in Openshift lauffähig ist. Angular 8 Frontend, das auch in OpenShift läuft. Es soll einfach möglich sein, einen Serviceaufruf (auch über mehrere Services hinweg) zu tracen (Jaeger). Die Build-, Test- und Deploymentvorgänge werden mithilfe von Openshift-Pipelines und Jenkins automatisiert.

3 Projektstruktur

1. SCAAnalyzerService: Das SCAAnalyzerService ist ein PythonService zum Analysieren von ScreenCasts.

2. SCAPersistService: Das SCAPersistService ist mit Quarkus implementiert. Quarkus ist der Nachfolger von Thorntail, was eine abgespeckte Version des Wildfly war. Quarkus ist eine full-stack, kubernetes-native Java framework und ist darauf optimiert darauf, dass Quarkus-Anwendungen in Kubernetes laufen. Es ist eine effektive Plattform für serverless, cloud und Kubernetes Umgebungen. Das SCAPersistService bietet dabei die Möglichkeit bereits analysierte Videos in einer MongoDB zu speichern und wieder abzurufen.

3. SCAAngularFrontend: Das Frontend ist mit Angular 8 implementiert und bietet eine graphische Oberfläche zur Interaktion mit den beiden Service. Grundsätzlich kann sich im Frontend ein User über sein Google-Konto anmelden und dort entweder seine Google-Konto Infos abrufen, ein Video hochladen und dies analysieren oder ein analysiertes Video speichern und in einem Archiv aufrufen.

4 Architektur

In OpenShift sieht die Architektur wie folgt aus:

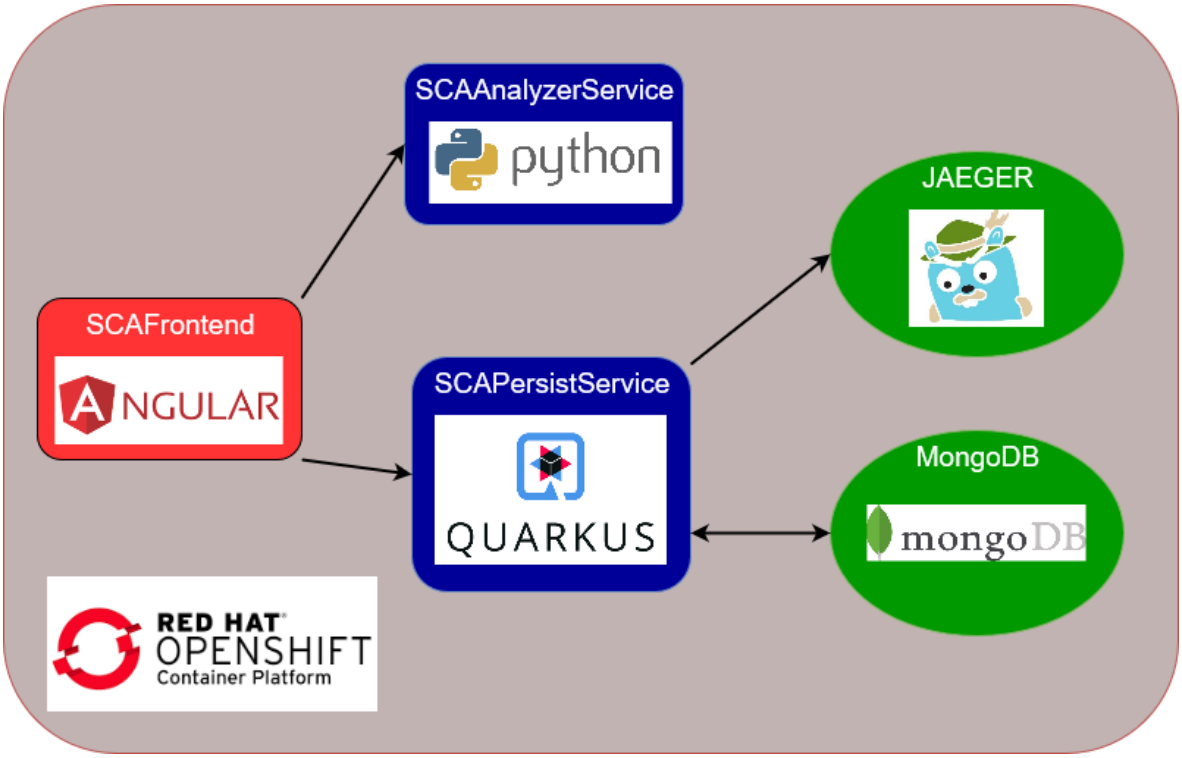

Abbildung 1:

5 Verwendete Technologien/Bibliotheken

- https://microprofile.io/: Eine Initiative um Java EE für Microsservice-Architekturen zu optimieren. Unterschiedliche Metriken, wie Health, Config, ...
- https://www.jaegertracing.io/: Ein Framework zum Monitoren von Microservices.
- https://www.openshift.com/: Ist eine Container-Plattform von Redhat, welche auf Docker und Kubernetes aufbaut.
- https://jenkins.io/: Automation Server für automatisierte Build-, Test- und Deploymentvorgänge.

6 Setup

1. TODO Codeready Container

2. In gewünschtes Verzeichnis entpacken und PATH Variable um dieses Verzeichnis erweitern

3. Docker starten (muss laufen damit Openshift Cluster gestartet werden kann, basiert auf Docker-Images)

4. **oc cluster up**: startet einen neuen Cluster

5. **oc login**: am lokalen Cluster anmelden (fabric8 deployed in den Cluster an dem man aktuell eingeloggt ist) -¿ USER/PASSWORT kann beliebig gewählt werden

6. **oc new-project [PROJEKTNAME]**: Legt ein neues Projekt an

7. **oc process -f https://raw.githubusercontent.com/jaegertracing/jaeger-openshift/master/all-in-one/jaeger-all-in-one-template.yml — oc create -f -**: Deployt ein vorgefertigtes Jaeger Template

8. **oc process -f https://raw.githubusercontent.com/sabre1041/openshift-api-swagger/master/openshift-api-swagger-template.yml — oc apply -f-**: Deployt ein vorgefertigtes SwaggerUI Template

7 OpenShift

OpenShift ist eine OpenSource Container Application Platform und setzt auf Kubernetes auf. Nach dem Befehl **oc cluster up** kann unter *localhost:8443* die grafische Oberfläche aufgerufen werden. Sind bereits Services in OpenShift deployt, sieht dies wie folgt aus: TODO image from OpenShift

Ein Service kann auf hinauf-/heruntergescaled werden. Dadurch können mehrere Pods gestartet werden. Das Load-Balancing wird von OpenShift übernommen. TODO image from OpenShift Projekt

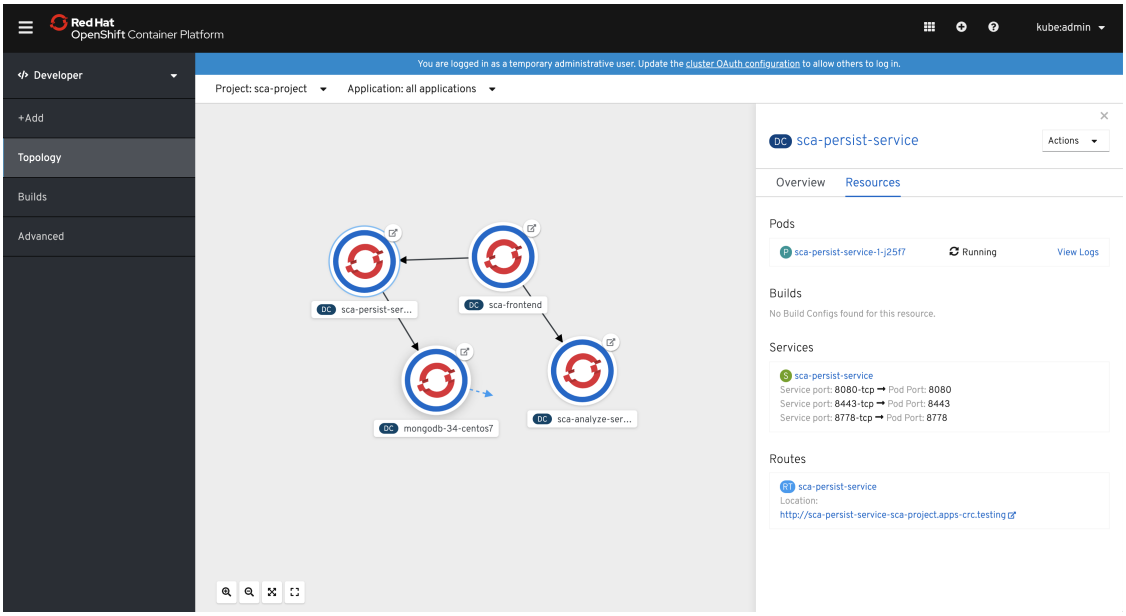

Abbildung 2:

8 OpenTracing mit Jaeger

Jaeger ist ein Tracing System, das OpenSource zur Verfügung steht. Es wird für Monitoring und Troubleshooting bei Microservice-Architekturen verwendet. Um Jaeger mit Thorntail und JavaEE verwenden zu können, werden folgende Teile benötigt:

- io.quarkus.quarkus-smallrye-opentracing Dependency
- Im application.properties folgende Konfigurationsparameter:
  - service-name: Service Name der mit den Spans assoziiert wird
  - agent-host: Host unter dem der jaeger-agent erreichbar ist
    - Dieser ist für die lokale Entwicklung localhost
    - Und in OpenShift **NICHT** die Route von Jaeger, sondern der Container Name von Jaeger. In unserem Fall jaeger-agent.
  - agent-port: Port unter dem der jaeger-agent erreichbar ist.
  - reporter-flush-interval: Definiert, wie oft Jaeger die Spans flusht
  - sampler-type: Definiert den Typ des Samplers, z.B.: probabilistic oder const
  - sampler-parameter: Configurations-Wert für den Sampler, z.B.: probabilistic = 0.001

Ist Jaeger und ein Service deployed und eine getracte REST-Resource dieses Service wird aufgerufen, wird der Span an Jaeger gesendet. Mittels der JaegerUI, deren URL in OpenShift zu sehen ist, können die Traces angesehen werden.

9 MongoDB

Deployen und Starten der MongoDB in OpenShift:
oc new-app -e MONGODB_USER=test -e MONGODB_PASSWORD=test -e MONGODB_DATABASE=sc
-e MONGODB_ADMIN_PASSWORD=test openshift/mongodb-24-centos7
Exposen der Route:
oc expose svc/mongodb-24-centos7

10 Fragestellungen

10.1 Automated Infrastructue Provisioning/(Infrastructure-as-Code). Wie wurde im vorliegenden Projekt Automated Infrastructure Provisioning berücksichtigt?

Beim Deployen der Services erzeugt OpenShift ein **deployment.yaml** mit Standardwerten. Dieses kann je nach Bedarf modifiziert werden. Werden nun System mit diesem deployment.yaml erzeugt, bleiben diese konsistent. Solange Änderungen auch nur in diesem Konfigurations-File durchgeführt werden, bleibt das System auch konsistent und kann jederzeit reproduziert werden. Dadurch werden die 4 folgenden Grundsätze von Infrastructure as Code in unserem Projekt umgesetzt.

- Systeme sind reproduzierbar
- Systeme sind wegwerfbar
- Systeme sind konsistent
- Aktionen sind wiederholbar

10.2 Skalierbarkeit. Wie wurde im vorliegenden Projekt Skalierbarkeit berücksichtigt?

In der Oberfläche von OpenShift kann ein Service per Hand rauf- und runterskaliert werden. Dies kann natürlich auch automatisch je nach Auslastung eingestellt werden. OpenShift übernimmt dabei selbst das Scheduling und das Load Balancing.

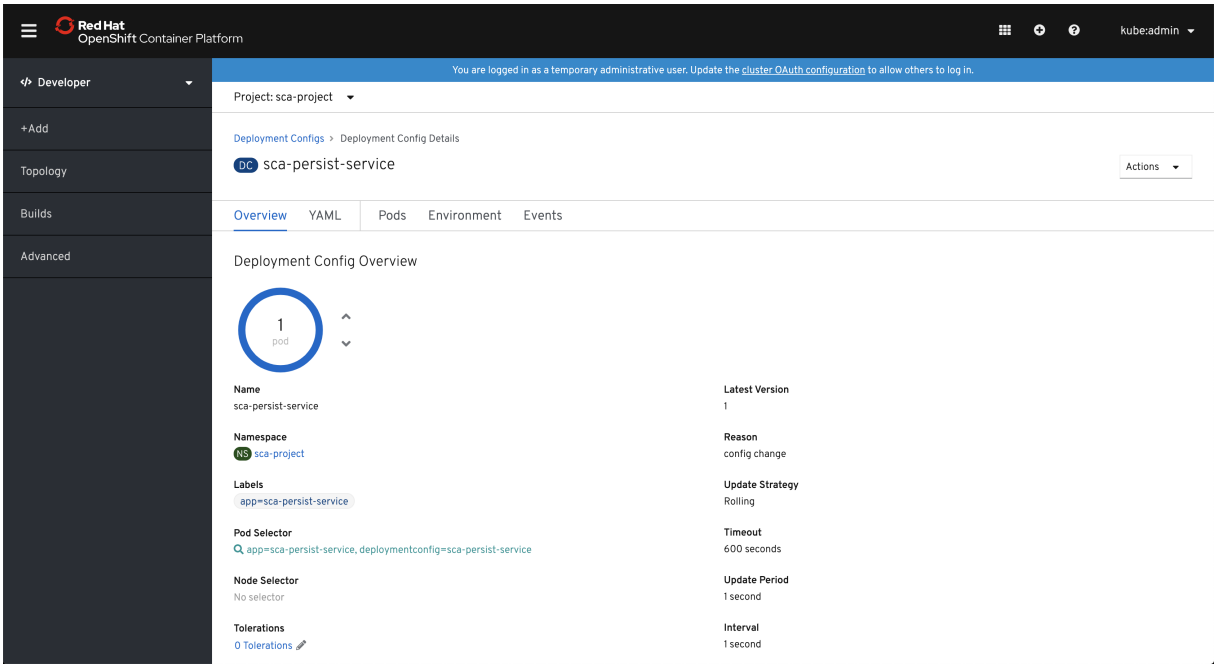

Abbildung 3: Scale Up- Funktion

10.3 Ausfallssicherheit. Wie wurde im vorliegenden Projekt Ausfallssicherheit berücksichtigt?

Grundsätzlich regelt die Ausfallsicherheit der Service OpenShift selbst. Das größte Problem hierbei wäre wenn die bestimmte Region ausfällt. Dazu müsste man ein Replika auf einer anderen Region deployen.

10.4 NoSql. Welchen Beitrag leistet NoSql in der vorliegenden Problemstellung?

Bei unseren Daten handelt es sich nicht um heikle Daten und auch nicht um Daten, wo zwingend ein Schema benötigt wird. Da bietet sich eine MongoDB gut an, da grundsätzlich unsere Daten auch schon in Form von JSON vorhanden sind und MongoDB auch bei größeren Daten auch noch performant ist.

10.5 Replikation. Wo nutzen Sie im gegenständlichen Projekt Daten-Replikation?

Replikation wird lediglich bei der Datenbank benötigt. Alle anderen Services, sowie die GUI sind stateless. Dies wurde in unserer Anwendung jedoch nicht umgesetzt.

10.6 Kosten. Welche Kosten verursacht Ihre Lösung? Welchen monetären Vorteil hat diese Lösung gegenüber einer Nicht-Cloud-Lösung?