Cloud Infrastruktur MovieMatch

Autoren: Flavio Müller, Ronny Schneeberger

Ort, Datum: 05.01.2023, Brugg-Windisch

Graphical user interface, application

Description automatically generated

Inhaltsverzeichnis

[Einleitung 3](#_Toc124512131)

[Vorstellung MovieMatch 3](#_Toc124512132)

[Vorstellung Infrastruktur 4](#_Toc124512133)

[MovieMatch Frontend - A 4](#_Toc124512134)

[Azure Kubernetes Service - B 4](#_Toc124512135)

[Ingress Controller - C 5](#_Toc124512136)

[Middleware - D 5](#_Toc124512137)

[Watcher - E 5](#_Toc124512138)

[Logging System - F 6](#_Toc124512139)

[Azure Cosmos DB - G 6](#_Toc124512140)

[Mock Option - H 7](#_Toc124512141)

[ALBERT - I 8](#_Toc124512142)

[Speech-to-text - J 8](#_Toc124512143)

[Funktionsabläufe Abfragen 10](#_Toc124512144)

[Empfehlung durch Nutzerprofil – 1 10](#_Toc124512145)

[Empfehlung durch Sprachaufnahme – 2 10](#_Toc124512146)

[Deployment der Services auf Azure 10](#_Toc124512147)

[Fazit Entwicklungsprozess 11](#_Toc124512148)

# Einleitung

Dieses Dokument beschreibt die Cloud Infrastruktur von MovieMatch. Es entstand im Rahmen des Studienganges Data Science and der Fachhochschule Nordwestschweiz. Es wird die Aufgabenstellung der Minichallenge 2 des Moduls *Cloud Infrastructure and Computing* bearbeitet. Es soll eine Cloud-API für Cognitive Services implementiert werden. Nebst der API selbst soll ein Monitoringsystem programmiert werden, welches die Kosten der eingesetzten Cloudservices überwacht.

Der Code für die verschiedenen Services ist in folgendem Git-Repository abgelegt:

<https://github.com/ronnyfhnw/cic-mc2>

# Vorstellung MovieMatch

MovieMatch ist ein Service, welcher Filmempfehlungen erstellen kann. MovieMatch entstand im Rahmen der Challenge *cds1 – Tinder for Movies*. Die Challenge verlangt ein Recommendersystem, welches Empfehlungen für Filme erstellen kann. Das Endprodukt ist ein Service, welcher Nutzerprofile erstellt und Empfehlungen abgibt. Die Publikation dieses Service verbinden wir mit der Minichallenge für das Modul *Cloud Infrastructure and Computing*.

Abbildung , Aufbau Recommender System

Diagram

Description automatically generated

Das Recommendersystem, welches von MovieMatch verwendet wird, ist aus mehreren Teilen aufgebaut. Die Itemmatrix wird mit Metadaten und Embeddings der Filmbeschreibungen, Schauspieler und Titel erstellt.

Wie bereits erwähnt, muss MovieMatch Nutzerprofile erstellen. Das System muss die Präferenzen des Benutzers kennenlernen. Dazu werden dem Benutzer verschiedene, zufällige Filme vorgeschlagen, die er bewerten kann. Nachdem der User zehn verschiedene Filme bewertet hat, bekommt er Empfehlungen vorgeschlagen.

Da die Minichallenge verlangt, dass ein Cognitive Service in der Infrastruktur enthalten ist, wurde MovieMatch um ein Feature reicher. Der Benutzer von MovieMatch soll mittels Sprachaufnahme einen Film beschreiben, den er gerne sehen möchte. Auf diese Sprachaufnahme soll er Empfehlungen erhalten. Dies wird wie folgt umgesetzt: Die Sprachaufnahme wird an den Speech-to-text (<speech-to-text>) Service von Azure geschickt, um die Sprachaufnahme in Text umzuwandeln. Aus diesem Text wird mit einem Sprachmodell ein Embedding generiert. Schliesslich werden die Distanzen des Embeddings der Sprachaufnahme zu den Embeddings der Filmbeschreibungen berechnet. Die kürzesten Distanzen werden dem User als Empfehlungen vorgeschlagen.

MovieMatch wird als Webservice zur Verfügung gestellt. Er ist unter folgender URL erreichbar: [www.moviematch.ch](http://www.moviematch.ch).

# Vorstellung Infrastruktur

In diesem Kapitel werden die einzelnen Services vorgestellt, die von MovieMatch verwendet werden. Am Ende des Kapitels ist ein Diagramm der Infrastruktur abgebildet.

## MovieMatch Frontend - A

Das Frontend besteht aus einer Angular 15 Single Page Application. Diese wird als statische Website auf einem Azure Storage Container gehosted und via Azure Content Delivery Network verteilt. Um auf das Mikrofon des Endgerätes zugreifen zu können muss die Website über HTTPS erreicht werden. Dies wird mit einem Reverse Proxy von Cloudflare erreicht.

Graphical user interface

Description automatically generated

Abbildung : Moviematch Frontend

## Azure Kubernetes Service - B

Der Kubernetes Cluster bildet das Herzstück der Infrastruktur. Er beinhaltet die Middleware, den cic-watcher und den cic-logger. Alle Anfragen auf diese drei Workloads werden von einem nginx Ingress Controller geroutet. Auch hier wurde wieder ein Reverse Proxy von Cloudflare davor geschaltet, um die Webschnittstellen über HTTPS aufrufen zu können.

Graphical user interface, text, application, email

Description automatically generated

Abbildung : Konfiugration des Kubernetes Clusters

## Ingress Controller - C

Der Ingress Controller wurde auf (<https://api.moviematch.ch>) gemappt und verteilt alle Anfragen basierend auf der Route an die drei verschiedenen Workloads. Der Ingress Controller bildet den einzigen Einstiegspunkt in den Kubernetes Cluster.

Table

Description automatically generated

Abbildung 4: Liste der ClusterIPs / LoadBalancer

## Middleware - D

Die Middleware ist das zentrale Element der Infrastruktur. Alle Abfragen vom Frontend laufen über die Middleware. Die Middleware koordiniert die Überwachung der Kosten mit dem Watcher, wandelt Sprachaufnahmen über den speech-to-text Service in Texte um, generiert Embeddings für Texte mit dem ALBERT Service und gibt Empfehlungen zurück an das Frontend.

## Watcher - E

Der Watcher überwacht die anfallenden Kosten des Clusters. Bei jeder Anfrage an die Middleware, wird von dieser geprüft, ob noch Budget vorhanden ist. Die Anfragen, welche an den Watcher gestellt werden, werden in einer Azure Cosmos DB persistiert. Damit können später auch Analysen und Auswertungen durchgeführt werden. Administratoren haben die Möglichkeit über einen API-Client (z.B. Postman) Einstellungen am Watcher vorzunehmen oder das aktuell verbrauchte Budget einzusehen. Diese Endpunkte benötigen eine speziellen Admin-Token um aufgerufen werden zu können.

## Logging System - F

Das Loggingsystem besteht aus zwei verschiedenen Use Cases. Zum einen kann über (<https://api.moviematch.ch/cic-logger/status>) der Status und die Latenz der anderen beiden Workloads abgefragt werden. Zusätzlich wird alle fünf Minuten probiert eine Verbindung zu den beiden Workflows aufzubauen. Dabei wird die Latenz der beiden Workloads gemessen. Diese Ergebnisse werden auch in die Azure Cosmos DB geloggt.

## Azure Cosmos DB - G

Der Azure Cosmos DB Cluster wird als serverless NoSQL-Variante betrieben und besteht aus zwei Datenbanken (cic-watcher und cic-logger). Auf den beiden Datenbanken befinden sich jeweils Collections, welche auf die jeweiligen Use Case des Workloads zugeschnitten sind.

Graphical user interface, text, application

Description automatically generated

Abbildung : Screenshot Azure Cosmos DB mit 2 Datenbanken

## Mock Option - H

Graphical user interface, application

Description automatically generatedDie Mock Option beinhaltet eine vereinfachte Variante des Cognitive Services Speech-to-Text. Sie wurde zur Ersparnis von Kosten bei der Entwicklung verwendet. So müssen nicht immer Anfragen an den teureren Cognitive-Service geschickt werden. Sie können einfach an die Mock Option umgeleitet werden, welche die Anfrage überprüft und bei Erfolg immer den gleichen Wert zurückschickt.

Abbildung , Mock Option Deployment

## ALBERT - I

Graphical user interface, application

Description automatically generatedDas Sprachmodell ALBERT wurde in eine Flask-API integriert. Die API erledigt das Preprocessing der Texte und wandelt sie in Embeddings um. Die API ist in einem Dockercontainer verpackt. Dieser Container wurde als Azure App Service eingesetzt. Die API ist unter folgender URL erreichbar: <https://azure-containerized-albert.azurewebsites.net>. Der Container wurde wegen folgenden Gründen nicht auf Kubernetes eingesetzt: Die Studentenversion erlaubt nur zwei Pods mit öffentlichen IP-Addressen. Wir entschieden uns deshalb die zwei wichtigeren Services, die Middleware und den Watcher, auf Kubernetes einzusetzen. Wir versuchten auch die API mit ALBERT in die Middleware zu integrieren. Da das Sprachmodell aber so gross ist, dass eine stärkere und teurere Maschine nötig gewesen wäre, entschieden wir uns dagegen. Die generierten Kosten der teureren Maschine wären mit dem 100 CHF Kredit des Studentenaccounts nicht tragbar gewesen.

Abbildung , ALBERT Deployment

## Speech-to-text - J

Graphical user interface, application

Description automatically generatedSpeecht-to-text ist ein Cognitive Service von Azure und kann als Cognitive Service in Ressourcengruppen gestartet werden. Danach können http Abfragen an den Service geschickt werden, die ein .wav File enthalten. Als Antwort kommt dann der erkannte Text zurück.

Abbildung , Speech-to-Text Deployment

Diagram

Description automatically generated

J

I

H

G

F

E

D

C

B

A

# Funktionsabläufe Abfragen

Wie bereits erwähnt gibt es zwei mögliche Abfragen, die vom Frontend an die Middleware geschickt werden. Die Abläufe der beiden Abfragen werden detailliert beschrieben.

## Empfehlung durch Nutzerprofil – 1

Wenn ein Nutzerprofil erstellt worden ist, können die ids der Filme, die der Benutzer als positiv bewertet hat, an die Middleware geschickt werden. Die Antwort enthält Empfehlungen. Die Anfrage wird wie folgt bearbeitet:

1. Anfrage an Watcher, Überprüfung Kostenlimit
2. Überprüfung des Keys
3. Überprüfung der Daten
4. Berechnung und Rückgabe der Empfehlungen

## Empfehlung durch Sprachaufnahme – 2

Wenn der Benutzer im Frontend eine Sprachaufnahme erstellt hat und sie abschickt, geschieht in der Middleware folgendes:

1. Anfrage an Watcher, Überprüfung Kostenlimit
2. Überprüfung des Keys
3. Anfrage an speech-to-text Service
4. Text von Speech-to-text an ALBERT schicken
5. Distanzen zwischen Embedding von ALBERT und Filmbeschreibungen berechnen
6. Empfehlungen zurückgeben

# Deployment der Services auf Azure

Die folgende Tabelle liefert einen Überblick, wo welche Services eingesetzt wurden.

|  |  |
| --- | --- |
| **Servicename** | **Verwendeter Service** |
| azure-containerized-albert | Azure App Service |
| Speech-to-text | Azure Speech Service |
| Middleware | Kubernetes Workload |
| Watcher | Kubernetes Workload |
| Logger | Kubernetes Workload |
| Ingress Controller | Kubernetes Ingress |

Die folgende Tabelle liefert einen Überblick, auf welche Art die Services eingesetzt wurden.

|  |  |
| --- | --- |
| **Servicename** | **Beschreibung** |
| azure-containerized-albert | Flask-API in Dockercontainer. Deployment und Konfiguration im Azure-Portal. |
| Speech-to-text | Kein Programmieraufwand. Deployment und Konfiguration im Azure-Portal. |
| Middleware | Flask-API im Dockercontainer. Deployment und Konfiguration via deployment.yml Datei. |
| Watcher | Flask-API im Dockercontainer. Deployment und Konfiguration via deployment.yml Datei. |
| Logger | Flask-API im Dockercontainer.  Deployment und Konfiguration via deployment.yml Datei |
| Ingress Controller | Installiert via Helm direkt auf dem Cluster. Konfiguriert via .yml File |

# Reflexion Entwicklungsprozess

Die Entwicklung der Middleware, des Watchers, der Mock-Option und des Loggers lief relativ einfach und ohne grössere Probleme ab. Die Verpackung der Applikationen in Dockercontainer lief ebenfalls ohne Probleme ab.

Beim Versuch das Sprachmodell ALBERT als Service anzubieten, gab es jedoch Probleme. Zuerst wurde versucht, ein ONNX-Modell zu generieren und dieses über einen Azure Machine Learning Workspace zu deployen. Das Deployment funktionierte zwar, jedoch kann mit ONNX-Modellen der Hidden State von ALBERT nicht erzeugt werden bzw. die Hidden States befinden sich nicht im Output. Nach längerer Suche tauchte ein offenes Issue auf, welches genau dieses Problem beschrieb. Deshalb konnten wir ALBERT nicht auf einem Azure Machine Learning Workspace laufen lassen. Die beste Alternative war, das funktionierende Modell in eine API zu packen und diese als Container in einem Azure App Service laufen zu lassen. Dies funktionierte ohne weitere Zwischenfälle.