Friends and Positions – Ausarbeitung

Fortgeschrittene Internettechnologien

Westfälische Hochschule

Sommersemester 2023

Prof. Dr. Jürgen Priemer

Florian Tünte

Alexander Dall

Marek Brüning

Benedikt Schwering

1 Beschreibung der Architektur

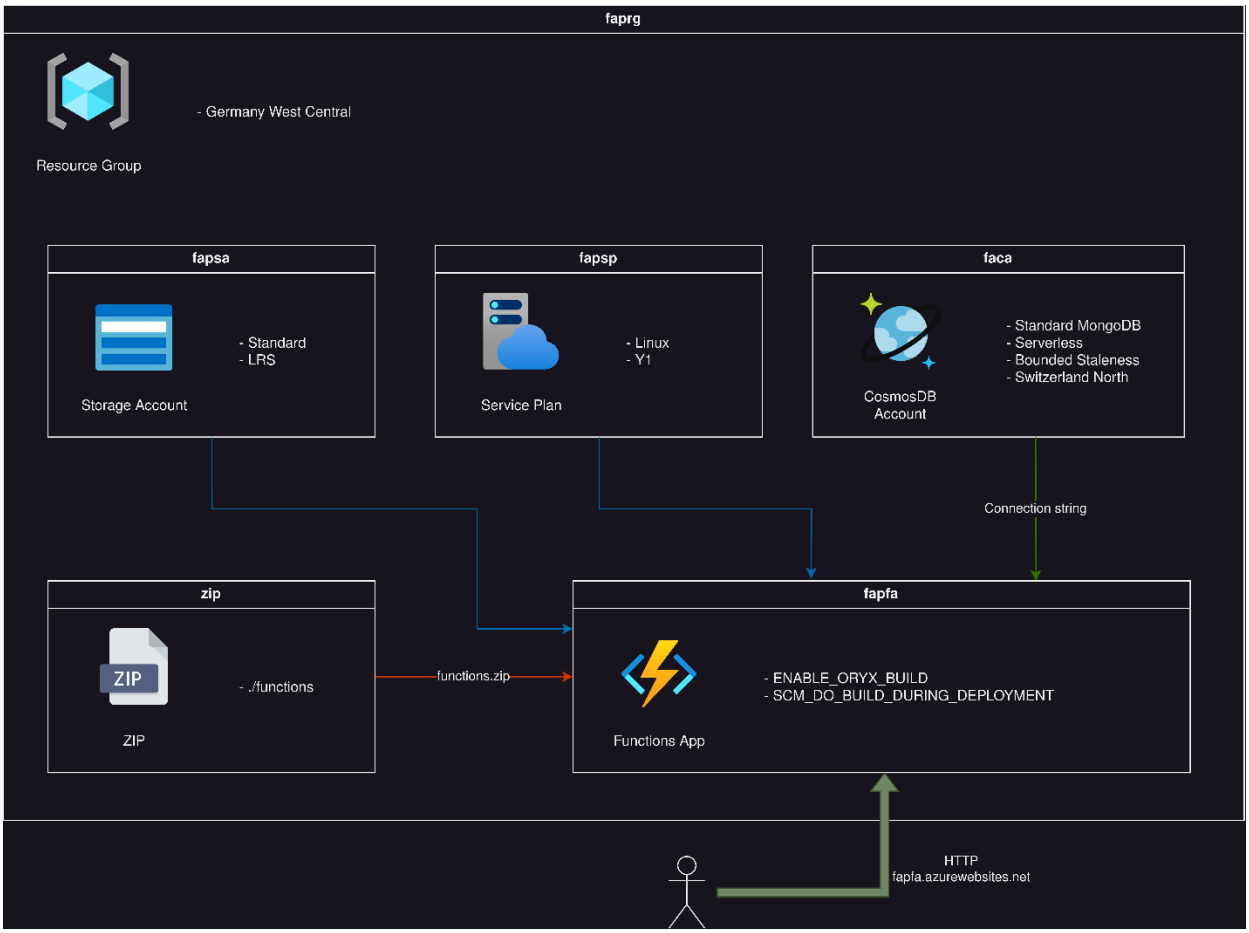

Abbildung 1

Abbildung 1 zeigt die Architektur des Friends and Positions Backend, bereitgestellt in der Microsoft Azure Cloud.

Auf oberster Ebene befindet sich die Ressourcen Gruppe „faprg“, der alle Services untergeordnet werden. Ressourcen Gruppen dienen der Gliederung der einzelnen Cloud-Services und bieten in einer Betriebs- und Betreuungssituation die Möglichkeit, Kosten zu analysieren und mehrere Services gleichzeitig zu modifizieren.

Auf der nächsten Ebene folgt der CosmosDB Account „fapca“. Dieser wird so konfiguriert, dass er sich wie eine native MongoDB Datenbank verhält. Zu den Konfigurationsargumenten gehört auch die Eigenschaft Serverless. Das bedeutet, dass die Abrechnung der Operationen nicht anhand der zu Verfügung gestellte Leistung, sondern anhand der tatsächlich in Anspruch genommenen Leistung erfolgt.

Neben der Datenbank wird eine Functions App, mit dem Namen „fapfa“, benötigt. Eine Functions App beinhaltet mehrere Functions. Eine Function ermöglicht es, kleine Codeabschnitt in der Cloud auszuführe, ohne die zugrunde liegende Infrastruktur zu planen und zu betrieben. Es wird lediglich

der Code bereitgestellt und mit einem Trigger, in diesem Fall HTTP, verknüpft. Außerdem skalieren diese Functions automatisch mit dem aktuellen Bedarf an Rechenleistung.
Um diesen Functions Service bereitzustellen, sind zwei weitere Ressourcen notwendig.

Der Storage Account „fapsa“ wird mit der Functions App verknüpft, um als Speicher für den Quellcode zu dienen.

Der Service Plan „fapsa“ dient der Functions App als Plattform der Laufzeitumgebung. Hier wird zum Beispiel konfiguriert, dass als OS Linux verwendet wird, und der Service Plan je nach Nutzung automatisch erweitert werden darf (Y1).

Zusammenfassend besteht die Infrastruktur in Azure aus einer Functions App, welche für jeden API-Endpunkt die jeweilige Funktionalität „serverless“ bereitstellt. Für die Persistenz der Daten wird die Applikation mit einer MongoDB Datenbank verknüpft. Diese wird durch den eigenen Azure Service CosmosDB bereitgestellt.

2 Beschreibung der wesentlichen Teile

Nachdem die Architektur erläutert wurde, werden in diesem Kapitel die wesentlichen Teile der Lösung vorgestellt. Dazu gehören zum einen die Entwicklung der Azure Functions und zum anderen die Bereitstellung der Cloud-Infrastruktur mit Terraform.

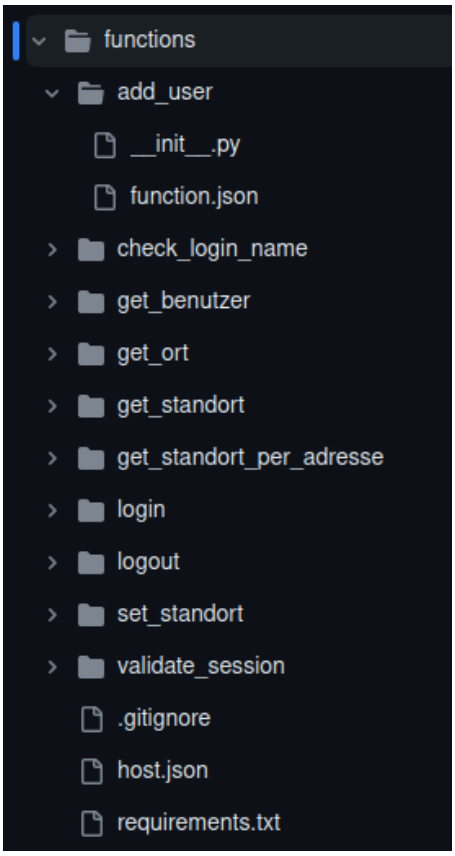

Abbildung 2

2.1 Functions

Alle Functions werden in Python entwickelt. Bevor eine Function den jeweiligen Anwendungsfall erfüllen kann, muss diese passend konfiguriert werden.

Der Aufbau eines typischen Functions Projekts ist in Abbildung 2 dargestellt.

Im Ordner „functions“ liegen die jeweiligen Functions, ausgelagert in eigene Unterordner.

Zudem wird das Verhalten der gesamten Functions App durch die Datei host.json und requirements.txt konfiguriert.

Die host.json Datei dient der Konfiguration der Laufzeitumgebung und der allgemeinen Verfügbarkeit über HTTP. In diesem Fall wird zum Beispiel die Version des unterstützten Azure Environments auf v3 und v4 gesetzt. Zudem wird für alle Funktionen das Prefix

„FAPServer/service/fapservice/“ an die HTTP Routen angehängt, um mit den Routen des originalen FAP-Servers kompatibel zu sein.

In der requirements.txt werden die benötigten Pakete für die Python-Umgebung definiert, sodass diese automatisch installiert werden können.

Auf der Ebene der Functions lassen sich zwei Dateien finden.

functions.json beinhaltet die Konfiguration der jeweiligen Function. Diese umfasst beispielsweise den HTTP-Trigger, die HTTP Method sowie den Namen der Route.

In der __init__.py Datei befindet sich der wichtigste Teil, der Quellcode der Funktion.

2.2 Bereitstellung

Die Cloud-Infrastruktur, beschrieben in Kapitel 1, wird mit Terraform bereitgestellt. Terraform bietet die Möglichkeit, Infrastruktur durch deklarativen Quellcode bereitzustellen, ohne dass es ein manuelles „zusammenklicken“ der Cloud-Komponenten in der Weboberfläche benötigt. Zudem vereinfacht es das Einbringen von Änderungen in der Infrastruktur, da dies nicht im laufenden Betrieb durch Operationen im Portal des Cloudanbieters, sondern durch einfaches Anpassen des deklarativen Codes geschieht.

In unserem Fall wird nicht nur die Infrastruktur mit Terraform bereitgestellt. Es erfolgt zudem noch das Deployment des Quellcodes für die Functions. Dazu wird der Ordner „functions“ von Terraform in ein ZIP-Archiv gepackt, um dieses bei der Erstellung der Function App einzubetten.

Abbildung 3 zeigt beispielsweise den Code für die Bereitstellung der Functions App mit dem Packen der zugehörigen ZIP-Datei.

Außerdem fällt die Zeile 37 des Codeausschnittes ins Auge. Hier wird der ConnectionString der MongoDB Datenbank aus einem vorangehenden Bereitstellungsschrittes genommen und als Environment Variable in die Functions App eingesetzt. Dies erleichtert die Handhabung der produktiven Passwörter der Datenbank und erhöht gleichzeitig die Sicherheit des bereitgestellten Services.

```
data "archive_file" "zip" {
    type = "zip"
    source_dir = "../functions"
    output_path = "functions.zip"
}

resource "azurerm_linux_function_app" "function_app" {
    name                        = "${var.project}fa"
    location                    = azurerm_resource_group.resource_group.location
    resource_group_name         = azurerm_resource_group.resource_group.name

    storage_account_name        = azurerm_storage_account.storage_account.name
    storage_account_access_key  = azurerm_storage_account.storage_account.primary_access_key
    service_plan_id             = azurerm_service_plan.service_plan.id

    app_settings = {
        ENABLE_ORYX_BUILD              = true
        SCM_DO_BUILD_DURING_DEPLOYMENT  = true
        MONGO_URI                      = azurerm_cosmosdb_account.cosmosdb_account.connection_strings[0]
    }

    site_config {
        application_stack {
            python_version = "3.9"
        }
        cors {
            allowed_origins = [
                "*"
            ]
        }
    }

    zip_deploy_file = data.archive_file.zip.output_path
}
```

Abbildung 3

2.3 Herausforderungen

Als besonders schwieriger Teil erwies sich die lauffähige Bereitstellung der Functions App. In den ersten Versionen meldete der Service einen internen Serverfehler (HTTP Code 500), da die Abhängigkeiten des Python Projektes nicht installiert wurden. Der Grund für das Fehlen war nicht ersichtlich. Die Lösung bedurfte die Parameter ENABLE_ORYX_BUILD = true und SCM_DO_BUILD_DURING_DEPLOYMENT = true damit die Abhängigkeiten aus der requirements.txt geladen werden.

Ebenso musste getestet werden, wie sich die API verhält, um die tatsächliche Funktionalität nachzubilden. So ist die Grenze für die Mindestlänge von Nutzernamen beispielsweise 5. Beim Anlegen eines Benutzers ist diese aber egal. Dort half es auch den dekompilierten Code anzusehen. Es war uns sehr wichtig, vollständig und integer zur originalen Java Server Version zu sein, damit andere Gruppen ihren Client entweder gegen unser Backend oder das Java Server Backend entwickeln können und diese auch nach Austauschen des Backends vollständig ihre Funktionalität behalten.

3 Erfahrungen und Bewertung

Das Analysieren der Java Anwendung und das Dekompilieren dieser war für uns nicht intuitiv. Trotzdem konnten wir eine Anwendung schreiben, die sich zur Ausgangsversion nur geringfügig anders verhält. Zudem war uns neu, das Fehlschlagen einer Abfrage mit einem HTTP Code 200 und dem Wert { ergebnis: false } zurückzumelden. Außerdem war es interessant, sich mit Terraform im Zusammenhang mit Azure auseinander zu setzen. Im Team hatten wir unabhängige Erfahrungen mit Azure und Terraform und konnten unser Wissen gut ergänzen.

Auch sind wir sehr glücklich, dass unser Cloud-Service bereits von einigen anderen Gruppen adaptiert wurde. Es konnte sogar kurzfristig eine neue Funktionalität hinzugefügt werden, sodass es einer Gruppe nun möglich ist, zu überprüfen, ob eine angegebene Session noch gültig ist.

4 Kurzbeschreibung eines Geschäftsmodells

Ein mögliches Geschäftsmodell ist, dass Eltern ihre Kinder mit einer App ausstatten, die in regelmäßigen Intervallen den aktuellen Standort meldet. Somit können die Eltern verfolgen, wo sich ihre Kinder aufhalten. Natürlich wird diese Funktion nur mit vorheriger Absprache und nur zur Gewährleistung eines sicheren Weges nach Hause, zum Beispiel bei Nacht, angewendet. Im Notfall oder bei ungewöhnlich später Heimreise könnten Eltern sichergehen, dass ihre Kinder sicher nach Hause kommen. Das könnte man um eine Benachrichtigung bei der Ankunft an einem Standort erweitern. Auch beim Antritt der Reise könnte man eine Benachrichtigung erhalten, um ihnen entgegenzukommen. Für die Kinder vermeidet das die Peinlichkeit von den Eltern abgeholt zu werden, als Elternteil hat man trotzdem die Chance auf seine Kinder aufzupassen.

Das Geschäftsmodell lässt sich auch auf ältere Menschen erweitern. Bei diesen kann ebenso sichergestellt werden, wo sie sich gerade aufhalten. In dieser Hinsicht ist es weniger die Sicherheit des Nachhause Weges, sondern viel mehr, darum dass diese nicht umständlich erklären müssen, wo sie sich aufhalten. Auch ohne tiefe Technologie Kenntnisse können diese sich melden, dass sie abgeholt werden wollen. Keiner muss sich daran erinnern, wo an diesem Sonntag Kuchen gegessen wird und mit wem die Oma morgen Kaffee trinken geht.

Ein weiterer Anwendungsfall sind Freunde. Wenn man mal mit seinen Freunden wieder zu tief ins Glas geschaut hat, kann es eine große Tortur bedeuten alle Mitglieder der Freundesgruppe in ihren jeweiligen Verfassungen wieder einzusammeln. Oder man hat einen Spezialisten dabei, der sich nie verabschiedet, den man am Ende aber genau deswegen noch sucht. Da er entweder in einer Ecke schläft oder bereits seit vielen Stunden zu Hause ist, jedoch sein Handy lautlos gestellt hat. Alle

installieren sich die App und teilen sich gegenseitig den Standort. Nie wieder Warten auf unauffindbare Freunde.

Um aus dieser Idee ein Geschäftsmodell zu erschaffen, sollte ein Abosystem eingeführt werden. Es existieren die Mitgliedschaften Basic mit 2€ pro Monat und Premium mit 5€ pro Monat. Als Basic Member kann man seinen Standort mit 20 Personen teilen und hat ein Intervall von mindestens einer Minute. Als Pro Member hat man die Möglichkeit bis zu 100 Freunde einzuladen und sekündlich den Standort zu teilen.

Auch sind einige weitere Schritte notwendig, um diese Funktionalität erfüllen zu können. Dazu gehört zum Beispiel die Möglichkeit, Standort nur mit bestimmten Personen zu teilen. Auch sollte ein nativer Client für die gängigen mobilen Endgeräte (Apple IOS und Android) entwickelt werden. Zudem wird eine Live Sharing Funktion benötigt, welche automatisch die GPS-Position des Handys ausliest und an den Server sendet. Zudem sollten bei neuen Positionen Push Benachrichtigungen an die verknüpften Personen gesendet werden. Als letzter Anforderungspunkt kann das Definieren und Erkennen von Orten, wie zum Beispiel „Schule“ oder „Zuhause“, im Rahmen von Geofencing hinzugefügt werden.