„WolkeSieben“ - die Cloud-Native Tiervermittlung

Klaus Fuhrmeister, Arne Müller, Niklas Sauer, Savina Diez

Februar 2021

1 Motivation und Idee

Tiere aller Art verdienen ein fürsorgliches und sicheres Umfeld. Leider kann dies nicht stets garantiert werden, in anderen Fällen funktioniert das nur für eine bestimmte Anzahl von Artgenossen. Daher leisten Tiervermittlungen wie Ebay Kleinanzeigen oder Tiervermittlung.de einen wichtigen Beitrag für unsere Gesellschaft. Auch wir, ein Team von Tierliebhabern, möchten uns dem anschließen, dies jedoch mit einer ansprechenden Benutzeroberfläche, intuitiven Funktionen und einer Infrastruktur tun, die theoretisch alle Vermittlungen der Welt übernehmen kann. Angesichts dieser Vorlesung, soll letzteres mit Hilfe der Google Cloud Plattform erreicht werden.

Das Portal mit dem Decknamen „WolkeSieben“ erlaubt aktuellen Besitzern Angebote zu erstellen, diese mit Bildern und Videos zu versehen, hierzu Anfragen zu erhalten und Details mit Interessenten in Echtzeit zu besprechen. Durch das Hosting in der Cloud können daneben noch einige weitere Features effizient und skalierbar umgesetzt werden. Allgemein erlaubt diese Form des Hostings natürlich einen Geräte-, zeit- und ortsunabhängigen Zugriff, sowie bessere Kostenstrukturen, da die Hardware, sowie die Wartung der Ressourcen entfallen.

Im Rahmen dieser Arbeit erläutern wir wie eine Vielzahl von Google Cloud Services zum Funktionsumfang und der Bereitstellung unserer Plattform beitragen werden. Zur besseren Übersicht, folge eine kurze Zusammenfassung:

- **Cloud Run** zur Bereitstellung zweier Microservices
- **App Engine** zur Bereitstellung eines dritten Microservice
- **Cloud CDN** zur Bereitstellung des Frontends
- **Cloud Firestore** zur Speicherung von Chat-Daten
- **Cloud Memorystore** zur Skalierung des Echtzeit-Chats
- **Cloud BigTable** zur Speicherung von „Gefällt mir“ Angaben
- **Cloud SQL** zur Speicherung von Benutzer Profilen, Angeboten, Merklisten, sowie Rassen-Abonnements

- **Cloud Storage** zur Speicherung von statischen Dateien wie dem Frontend, Profil- / Angebots-Medien und einer Admin-Oberfläche
- **Cloud Functions** zur Benachrichtigung über neue Angebote von abonnierten Rassen
- **Cloud Pub/Sub** zum Auslösen der o.g. Benachrichtigungen

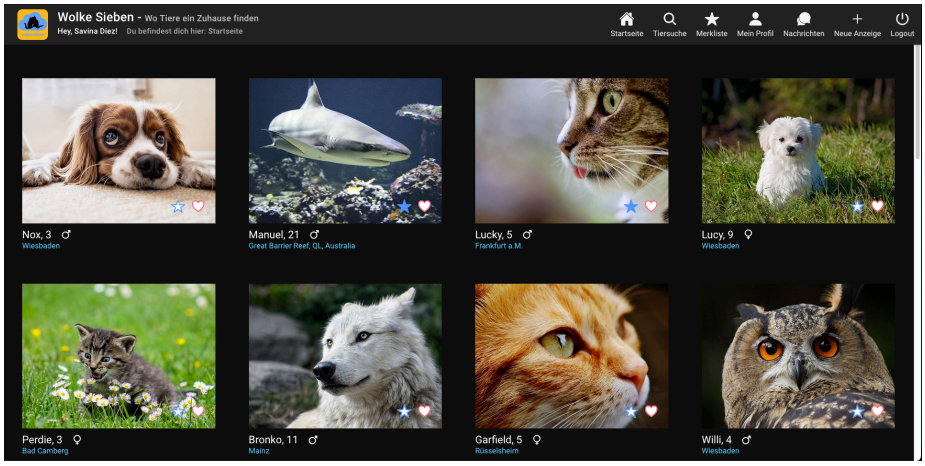

Abbildung 1: Startseite von cc-wolkesieben.de

Der Prototyp unserer Applikation, wie in Abbildung 1 demonstriert, ist ab sofort unter folgender Adresse aufrufbar: `https://cc-wolkesieben.de`.

2 Architektur

2.1 Microservices

Wie für Cloud-Anwendungen üblich, setzen wir bei der Struktur des Backends auf einen Microservice-Ansatz bei dem die ganzheitliche API in verschiedene, voneinander unabhängige Programme aufgeteilt wird. Diese Form der Bereitstellung hat einige Vorteile, die hier nochmal herausgestellt werden sollen.

Zum einen können die einzelnen Funktionen auf diese Weise in Umgebungen bereitgestellt werden, welche speziell auf diese zugeschnitten sind. In unserer Anwendung sieht man dies primär anhand der Aufteilung nach den verwendeten Datenbank-Typen (Polyglott Persistency). Im Umkehrschluss, muss sich jeder Microservice somit auch nur mit denen für seine Domäne relevanten Technologien befassen. Die erleichtert die Konfiguration und trägt dazu bei, dass sich Services einfacher unter den Entwicklern eines Team aufteilen lassen, ohne dass diese sich in die Quere kommen. Zu guter Letzt lassen sich die Möglichkeiten

der Bereitstellung in der Cloud so optimal ausnutzen, da jeder Microservice auf der Infrastruktur aufgesetzt werden kann, die für ihn am geeignetsten ist, ohne den Rest der Anwendung auf einen Kompromiss zu forcieren. Dies wiederum erlaubt die unabhängige Skalierung der am meisten nachgefragten Teilbereiche, d.h. Microservices, der ganzheitlichen API.

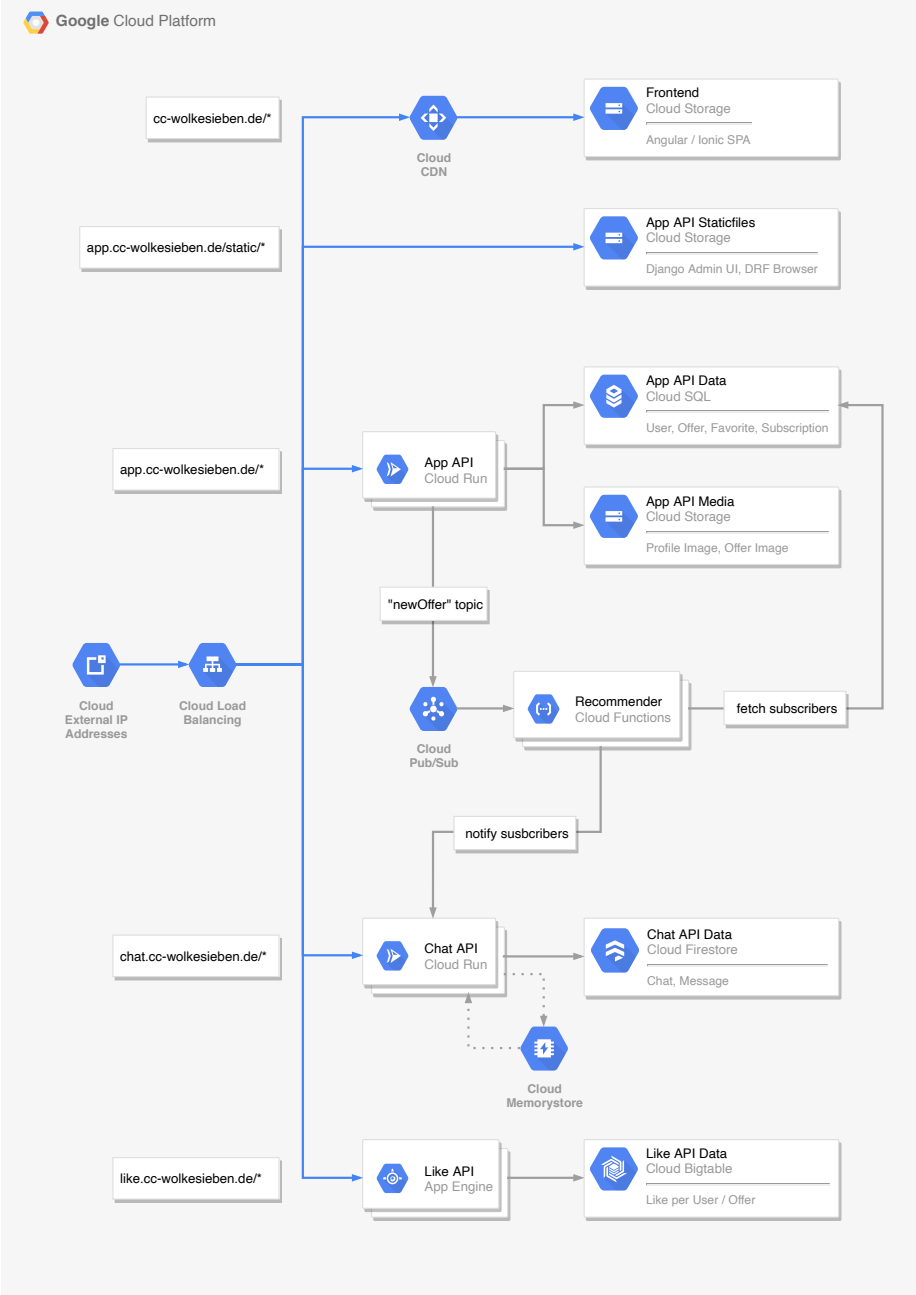

Abbildung 2: Architekturüberblick

2.2 Domain-Driven Design

Ursprünglich sollte sich die Architektur der einzelnen Microservices am Domain-Driven Design Pattern orientieren. Dieses sieht eine Unterteilung der Services wie folgt vor:

Domäne
für die Kernfunktionen des Service

API
für die Bereitstellung

Infrastruktur
für die Anbindung an externe Systeme (z.B. Datenbanken)

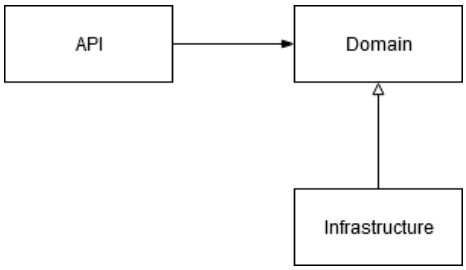

Abbildung 3: Domain-Driven Design

Im Laufe der Entwicklung stellte sich heraus, dass sich das Pattern, welches eigentlich für DotNet-Anwendungen konzipiert wurde, nicht so einfach auf die von uns verwendeten Programmiersprachen und Frameworks übertragen lässt.

Konkret, soll dies am Beispiel von Django, einem Python Web-Framework, verdeutlicht werden: Einer der Hauptzwecke der Infrastruktur-Komponente ist es die Anbindung verschiedener Datenbanken, sowie den Wechsel zwischen diesen so einfach und standardisiert wie möglich zu gestalten. In Django besteht dieses Problem allerdings garnicht, da die Datenbankanbindung komplett im Hintergrund stattfindet. Bei Verwendung des Django-REST-Frameworks entfällt außerdem ein Großteil der Arbeit, die zur Erstellung einer REST-API nötig ist, da auch hier vieles im Hintergrund passiert. Des weiteren gibt Django eine relativ umfassende Dateistruktur vor, die sich nur schwer mit dem Domain-Driven Design vereinen lässt. Aus diesen Gründen haben wir uns bereits zu einem frühen Zeitpunkt der Entwicklung entschieden dieses Pattern nicht länger als Orientierung zu nehmen.

3 Frontend

3.1 Technologie

Hinsichtlich dem Ziel unsere Plattform einer möglichst großen Masse an Benutzern zugänglich zu machen, wird das Frontend mit Ionic (Version 5) entwickelt. Obwohl augenscheinlich nur Browser Technologien zum Einsatz kommen (HTML, CSS, JavaScript), kann die mit Hilfe von Ionic entwickelte Single-Page-App anschließend auch in eine native iOS bzw. Android App konvertiert werden. Intern fußt Ionic auf dem von Google propagierten Angular Framework, welches das „Two-way Binding“ Paradigma umsetzt. Es unterstützt auch die von uns bevorzugte Programmiersprache TypeScript, ein Superset von JavaScript, sowie SASS als erweiterte Stylesheet-Sprache.

3.2 Authentifizierung

Die Authentifizierung eines Benutzers wird wie in Grafik XY durchgeführt. Im ersten Schritt wird überprüft, ob bereits eine Session vorhanden ist. Diese ist durch das Vorhandensein eines gültigen JSON Web Token (JWT), ausgestellt vom App-Microservice (siehe Abschnitt 4), gekennzeichnet. Andernfalls ist der Benutzer nicht eingeloggt. In diesem Fall, kann der Nutzer sich mit einem Social Provider (Facebook, Google, GitHub etc.) einloggen. Aktuell wird nur Google unterstützt. In jedem Fall wird jedoch eine OAuth 2.0 Anfrage an den jeweiligen Provider abgesetzt, die das Öffnen dessen Single Sign-On Fenster zur Folge hat. Bei erfolgreichem Ausweisen der Identität (normalerweise durch Benutzername und Passwort, ggf. mit OTP) liefert dieses ein sogenanntes ID Token zurück, das auch dem JWT Standard folgt. In dessen Payload findet sich neben den verifizierten Benutzer Details, wie Name oder Email Adresse, auch die in der Domäne des jeweiligen Provider eindeutige ID des Nutzers. Um den Login Vorgang abzuschließen, wird das ID Token an den App-Microservice gesendet, welches die Gültigkeit anhand der Signatur prüfen kann und dieses gegen ein neues JWT austauscht. Die Notwendigkeit für ein eigens ausgestelltes JWT wird in Unterabschnitt 4.3 erläutert.

Genau genommen, handelt es sich bei der Antwort des App-Microservice um ein JWT-Paar, nämlich einem Access- und Refresh-Token. Das Access-Token wird für künftige Requests an jeden der Microservices als `Bearer` Token im `Authorization` Header mitgesendet, worauf basierend die Authentifizierung und Authorisierung in jedem Microservice stattfindet. Das Refresh-Token hingegen erlaubt es dem Frontend ein neues Access-Token zu erhalten, wenn dieses seine zeitliche Begrenzung überschritten hat. Refresh-Token sind in der Regel ein vielfaches länger gültig.

Um die Login-Session auch zwischen Neuladen der Webseite beizubehalten, werden beide Tokens im `localStorage` persistiert. Damit muss der oben beschriebene Login-Prozess erst wiederholt werden, wenn sich der Nutzer längere Zeit nicht im Portal bewegt hat und somit auch beide Tokens abgelaufen sind. Bei einem manuellen Logout werden beide Tokens zwangsläufig entfernt.

3.3 Google OAuth Integration

Zur einfachen Integration mit Angular, stellt Google das `gapi` NPM Package bereit. Es verwaltet den Zustand der Authentifizierung mittels eines `GoogleUser` Objektes, das auch eine Methode exponiert (`.getAuthResponse().id_token`), um das in Unterabschnitt 3.2 erwähnte ID Token abzurufen. Sollte der Benutzer nicht angemeldet sein, kann der OAuth 2.0 Prozess durch die statische Methode `gapi.auth2.init()` gestartet werden. Diese Methode sollte mit der in der Google Developer Konsole konfigurierten `CLIENT_ID` aufgerufen werden, um sicherzustellen, dass das ID Token aus dieser Login-Session und nicht der einer anderen Applikation stammt.

3.4 Deployment

Wie in Unterabschnitt 3.1 erläutert, besteht dieser Teil des Projekts lediglich aus einem Superset von Browser-Technologien (HTML, CSS, JavaScript). Es wird kein serverseitiger Code ausgeführt, somit ist auch kein Webserver erforderlich, um die Anwendung zu hosten. Stattdessen ist es ausreichend, das Frontend in einem öffentlichen read-only Dateisystem zu speichern. Hierfür eignet sich der Google Cloud Storage ideal. Um die Latenz noch weiter zu verringern, kann der Google Cloud CDN Service eingesetzt werden. Dies wiederum setzt einen HTTP(S) Load Balancer voraus, wie im Folgendem beschrieben.

3.4.1 Google Cloud Storage

Der Google Cloud Storage ist ein Cloud Speicher, um beliebige Binary Large Objects (BLOBs) zu speichern und in sogenannten Buckets (vgl. Ordner) zu verwalten. Es gibt kein Speicherlimit, stattdessen wird die Benutzung pro belegtem Gigabyte abgerechnet. Datensicherheit, Langlebigkeit sowie Verfügbarkeit werden durch die Google SLAs garantiert.

Mit dem Programm `gsutil` stellt Google eine Kommandozeilen-Schnittstelle zur Verfügung, über die auch der Frontend Bucket dieses Projektes regelmäßig aktualisiert wird.

3.4.2 Cloud CDN & Load Balancer

Das Cloud Content Delivery Network (Cloud CDN) ist ein Netzwerk von Google, welches überall auf der Welt Knotenpunkte verteilt hat und damit auf die schnelle Auslieferung von Inhalten optimiert ist. Um dieses zu nutzen, muss ein HTTP(S) Load Balancer eingerichtet werden, der Anfragen an eine Domain erhält und diese an den jeweiligen Google Cloud Storage Bucket weiterleitet. Das Cloud CDN ist dazwischengeschaltet und fungiert als unabhängiger, automatischer Cache in mehreren Regionen der Welt.

Für die Skalierung bedeutet dieses Konstrukt, dass der Bucket nicht skalieren muss, denn Anfragen entstehen lediglich während dem CDN Cache-Refill bzw. beim Upload in den Bucket selbst. Bei einer steigender Anzahl von Nutzern muss nur das Cloud CDN skalieren. Hierfür sorgt Google vorbildlich.

4 App-Microservice

Der App-Microservice stellt den Großteil der Backend-Funktionen unserer Anwendung bereit. Er ist in Python geschrieben, verwendet das Django Web-Framework und wird über Cloud Run bereitgestellt.

4.1 Technologie

Die Programmiersprache Python ergab sich aus der Wahl des Web-Frameworks. Django bietet einige entscheidende Bausteine, welche die Entwickelung von Web-Anwendungen vereinfachen und von denen auch wir großen Gebrauch gemacht haben.

Der vorzugsweise größte Vorteil ist die vollständige Abstraktion des Datenbankmodells durch das eingebaute Object-Relational Mapping (ORM). Dies ermöglicht das Anlegen und Verwalten einer SQL-Datenbank ohne echten SQL-Code schreiben zu müssen. Modelle und Relationen können mit Python-Code definiert werden, aus denen per Konsolen-Befehl sogenannte Migrations generiert, die anschließend von Django durch Umwandlung in SQL-Code auf die Datenbank übertragen werden. Entwickler können sich dadurch auf die reine Programmierung in Python konzentrieren.

Bei Zuhilfenahme des Django-REST-Frameworks werden auch die Standard REST-Funktionen (`GET`, `POST`, `DELETE`, etc.) für alle eigens angelegten Datenbank Modelle vor-implementiert, wodurch sich Entwickler umso mehr auf die für die Anwendung benötigten Funktionen konzentrieren können. Dieses Plugin erleichtert auch das Testen der Anwendung, in dem es eine Web Oberfläche zur Interaktion mit der API bereitstellt.

Daneben haben sich die eingebauten Konzepte, die Anwendung produktionsreif zu machen, schon mehrfach in der Industrie bewährt.

4.2 Datenbank

Entsprechend dem Polyglott Persistency Prinzips, setzt der App Microservice auf eine Cloud SQL Instanz für relationale Daten, sowie einen Cloud Storage Bucket für Medien sonstiger Art (z.B. Fotos und Videos). Hinsichtlich dem SQL-Datenbank Typen, genügt MySQL (Version 8) unseren Anforderungen.

An dieser Stelle sollte erwähnt werden, dass zum Zeitpunkt der Entwicklung kein offizieller Emulator für den Google Cloud Storage vorliegt. Daher wurde zum Testen der Datenbankanbindung auf eine Open-Source Lösung namens `fake-gcs-server` zurückgegriffen.

4.3 Authentifizierung

Wie bereits in Unterabschnitt 3.2 dargelegt,

Bei der Nutzerauthentifizierung haben wir uns dazu entschieden, selbst keine Nutzerkonten zu verwalten und so Entwicklungszeit zu sparen, die wir in die eigentliche Anwendung stecken konnten. Stattdessen authentifizieren wird unsere

Nutzer über Google OAuth und JWT-Tokens. Dabei verlassen wir uns nur bei der Anmeldung auf die von Google ausgestellten Token. Der App-Microservice legt nach der Bestätigung der Identität ein eigenes JWT-Token an, welches daraufhin verwendet wird, um einen Nutzer zwischen unseren Microservices zu authentifizieren.

4.4 Deployment

Des Weiteren haben wir uns dazu entschieden, den App-Microservice mit Cloud Run in Docker-Containern bereitzustellen. Indem wir auf Container zurückgreifen, machen wir uns die flexible Struktur der Cloud-Umgebung zu Nutzen, in der Anwendungen ortstransparent bereitgestellt werden können. Durch die Verwendung von Containern machen wir uns von der zu Grunde liegenden Hardware unabhängig und können die Möglichkeiten der Cloud voll ausschöpfen. Weiterhin erledigt sich das Thema Skalierbarkeit so quasi von selbst, da das System auf Seiten des Anbieters in der Lage ist, jederzeit neue Container zu starten oder zu stoppen, um mit den Anforderungen des eingehenden Datenverkehrs mithalten zu können.

5 Like- und Chat-Microservices

5.1 Like-Microservice

5.2 Chat-Microservice

6 Recommender Cloud Function

Wenn ein neues Angebot erstellt wird, sollen diejenigen Nutzer per Chat-Nachricht benachrichtigt werden, die die jeweilige Rasse abonniert haben. Dies erfolgt mithilfe einer Publisher/Subscriber-getriggerten Google Cloud Function. Wir haben uns für dieses Modell entschieden, da es sich bei der Recommendation um ein nicht-zeitkritisches Feature handelt. Um die Nachricht zu senden, wurde ein Recommendation-Bot implementiert, der den jeweiligen Benutzern per Chatnachricht die Rasse sowie den Link zum neuen Angebot schickt. Das Publishen erfolgt durch den App Microservice. Die Recommender Function stellt anschließend einen GET-Request an den App Microservice, um die betroffenen User zu bekommen. Dann wird für jeden betroffenen User ein GET-Request an den Chat Microservice gestellt, um zu prüfen, ob bereits ein Chat zwischen dem Bot und dem User existiert. Falls nein, wird ein neuer Chat angelegt. Anschließend wird die Nachricht in diesem Chat gesendet.

7 Security

Wir haben bei der Implementierung darauf geachtet, dass wir als Betreiber sowie die Nutzer gegen die gängigsten Angriffsmethoden geschützt sind, sodass

ein Abhören, Manipulieren oder Beschädigen nicht möglich ist. Im Folgenden werden unsere Lösungskonzepte vorgestellt.

7.1 Verschlüsselung der Kommunikation

Eine unverschlüsselte Kommunikation der Nutzer mit einem Webserver öffnet Angriffen wie Man-in-the-Middle Attack, Cross-Site-Scripting (XSS) und Session Hijacking alle Türen. Wir nutzen ausschließlich HTTPS, wobei Webbrowser (Client) und Server mit einem nur für sie bekannten Schlüssel arbeiten. Somit läuft sämtliche Kommunikation verschlüsselt ab und ein Angreifer kann keine Informationen extrahieren, sofern er versuchen sollte, die Kommunikation abzuhören.

7.2 Authentifizierung und Autorisierung

Nutzer, die unsere Applikation verwenden möchten, müssen sich als erstes mit ihrem Google-Konto anmelden. Hier macht Google Gebrauch von dem OAuth 2.0 Protokoll. Das von Google ausgestellte JWT-Token verwenden wir dazu, um eine authentifizierte Anfrage an den App Microservice zu stellen, der ein eigens erstelltes, neues JWT Token zurückgibt. Dieses Verfahren macht unsere Applikationen flexibler im Umgang mit potenziellen anderen Login-Verfahren, die in Zukunft implementiert werden könnten. Außerdem können wir unsere eigene Payload, zum Beispiel die selbst erstellte UUID des Nutzers codieren. Wir erstellen nämlich eine neue UUID, da es theoretisch möglich ist, dass eine von Google zugewiesene ID identisch mit einer ID sein kann, die eine andere Plattform ausstellt.

Sämtliche Kommunikation mit den Services erfordert das Senden des von uns erstellten Tokens im `Authorization` Header. Für zusätzliche Sicherheit stellen wir außerdem ein Refresh-Token aus, mit dem nach Ablauf der Gültigkeit des Access-Tokens (fünfzehn Minuten) über den App Service ein neues Token angefragt werden kann. Mit einem User-JWT-Token können Anfragen an den App Microservice, den Like Microservice und den Chat Microservice geschickt werden, beispielsweise um passende Tierangebote zu finden, Angebote zu erstellen oder Profilinformationen zu ändern. Ein Nutzer ist nicht dazu autorisiert, im Namen eines anderen Nutzers oder des Recommendation Bots Nachrichten zu versenden oder ein Angebot hochzuladen. Für administratorische Zwecke existiert ein entsprechendes Service Token. Der Recommendation Bot hat ebenfalls ein eigenes Token, hauptsächlich dafür, um eine Anfrage an den App Service zu stellen, der die Abonnentenliste für eine bestimmte Rasse aus der Datenbank zurückgibt.

7.3 Weitere Lösungen

Unsere Applikation implementiert weitere Mechanismen zur Gewährleistung der Sicherheit.

- Der Datenbankzugriff erfolgt ausschließlich vom App Microservice über Django Queries. Die Querysets von Django sind vor SQL-Injektion geschützt, da ihre Abfragen mit Hilfe von Query-Parametrisierung konstruiert werden. Der SQL-Code einer Abfrage wird getrennt von den Parametern der Abfrage definiert. Da Parameter vom Benutzer bereitgestellt werden können und daher unsicher sind, werden sie vom zugrunde liegenden Datenbanktreiber escaped. [2]

- Im App Microservice haben wir die `CSRF-Middleware` manuell aktiviert, die einen Schutz gegen Request Forgeries von anderen Seiten implementiert.

- Da wir grundsätzlich keinen User-Input in das Dateisystem einfügen, können Directory Traversal-Angriffe in unserer Applikation nicht stattfinden.

8 Scaling

9 Continuous Integration und Continuous Delivery

Bei Bau und Bereitstellung der verschiedenen Teile unserer Anwendung haben wir uns die Möglichkeiten der Continuous Integration sowie der Continuous Delivery (CI/CD) zu Nutze gemacht. Dies erschien uns als natürlicher Schritt, da GitHub, welches wir zur Versionsverwaltung genutzt haben, diese Konzepte selbst unterstützt. Über die sogenannten GitHub Actions kann der Code per CI/CD gebaut und bereitgestellt werden, ohne dass hierfür eine separate Plattform eingerichtet werden muss. Wir haben uns dazu entschieden, die notwendigen Dateien (Dockerfile u.ä.) im selben Projekt wie den Quellcode zu halten. Diese Entwicklungsmethode ist als "Docker-Native" bekannt und zielt darauf ab, die Docker-Dateien an den Quellcode zu koppeln und die Verwaltung zu vereinfachen.

Für die Bereitstellung selbst haben wir das YAML-Format verwendet, mit dem sich die gewünschten Ressourcen als eine Art Skript beschreiben lassen. Dieses können die Cloud-Dienste dann für die automatische Provisionierung der Hardware benutzen.

Eine Schwierigkeit, die uns beim Bereitstellen der Microservices, die Cloud Run benutzen, aufgefallen ist, ist dass das System es offenbar nicht erlaubt, eine SSH-Sitzung in einem laufenden Container zu starten. Für die Bereitstellung ließ sich dieses Problem umgehen, da alle hierfür erforderlichen Konsolenbefehle sowieso in der YAML-Datei definiert werden müssen, um automatische ausgeführt werden zu können. Für die Wartung im laufenden Betrieb ergibt sich jedoch das Problem, dass keine Fehleranalyse stattfinden kann, da nicht auf die Fehlermeldungen zugegriffen werden kann. Hier wird eine ausgefeiltere Logging-Methode benötigt werden, um den reibungslosen Betrieb sicherzustellen. Uns ist Weiterhin bekannt, dass andere Cloud-Anbieter diese Möglichkeit sehr wohl zur

Verfügung stellen. Dies wird sicherlich einen Einfluss auf die Wahl der Plattform für zukünftige Cloud-Projekte haben.

10 Kostenanalyse

Um die Kosten für eine bestimmte Anzahl Benutzern zu berechnen, müssen zuvor einige Annahmen getroffen werden.

10.1 Annahmen zum Nutzerverhalten

Verglichen mit anderen Plattformen, treffen wir die Annahme, dass es im Wesentlichen zwei Nutzergruppen gibt. Eine Gruppe nutzt die Anwendung hauptsächlich zum Stöbern und Kaufen, die andere Gruppe nutzt die Anwendung zum Anbieten von Tieren. Ein kleiner Anteil der Benutzer wird sowohl kaufen als auch verkaufen. So kommen wir zu der Annahme, dass ein Benutzer im Durchschnitt zwei Inserate online hat.

Weiterhin gehen wir davon aus, dass jedes Angebot im Durchschnitt zwei Bilder und ein Video beinhaltet, was bei einer Größe von zwei MB pro Bild und 16 MB pro Video zu einem Gesamtspeicherplatz von etwa 20 MB pro Angebot führt.

Um den Datenverkehr zu berechnen, gehen wir davon aus, dass ein registrierter Benutzer pro Tag ein Angebot aufruft sowie fünf mal die Startseite besucht. Diese Zahl haben wie etwas höher angesetzt, da auch nicht registrierte Benutzer die Startseite aufrufen können und die Anzahl dieser Aufrufe schwierig vorherzusagen sind.

Zusammengefasst haben wir folgende Parameter festgelegt:

Anzeigen pro Benutzer	2
Speicherplatz pro Anzeige	20 MB
Datenverkehr pro Nutzer pro Monat	600 MB
Anzahl Likes pro Anzeige und Benutzer	2
Anzahl Chats pro Benutzer pro Monat	2
Anzahl Nachrichten pro Chat	6
Größe einer Nachricht	20 KB

Aus den Werten der Tabelle 10.1 kann man mithilfe des Google Cloud Platform-Preisrechners berechnen, welche Cloud Services welche Kosten verursachen. [1]

Abbildung 10.1 zeigt den Kostenanteil der Cloud Services im Verhältnis zu den Gesamtkosten.

Literatur

[1] Google cloud platform pricing calculator. `https://cloud.google.com/products/calculator`. Accessed: 2021-02-20.

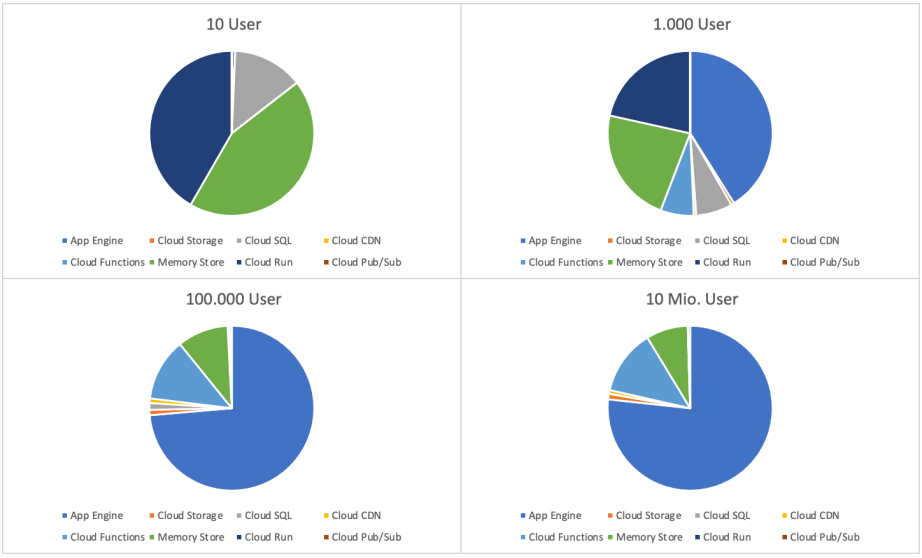

Abbildung 4: Kostenanteil der einzelnen Services

Abbildung 5: Gesamtkosten gruppiert nach Nutzerzahl

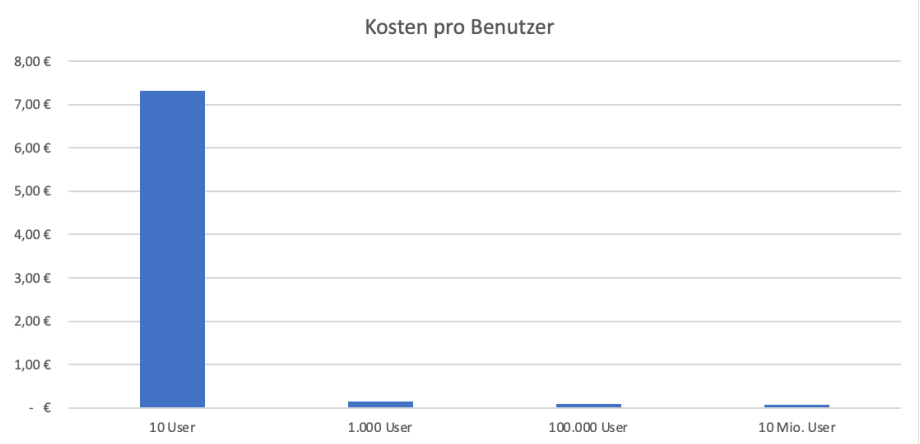

Abbildung 6: Gesamtkosten gruppiert nach Nutzerzahl

[2] Security in django. `https://docs.djangoproject.com/en/3.1/topics/security/`. Accessed: 2021-02-14.