SA4E U2

Karim Amri

January 2025

1 Modellierung

Man betrachte Abbildung 1. Alle Wünsche werden zunächst in einer **global verteilten Cloud-basierten Datenbank** gespeichert. Ein Mensch kann über einen API-Aufruf seinen Wunsch an einen geographisch naheliegenden **Load-Balancer (LB)** senden, der die Unmengen an Wünschen auf eine horizontal skalierbare Menge von **WishServices (WS)** weiterleitet. Ein WishService sendet den Wunsch dann samt Meta-Daten und initialen Wunsch-Status (1: Nur gewünscht, 2: In Bearbeitung, etc.) in die Cloud-Datenbank. Die Cloud-Datenbank ist regional aufgeteilt, um Latenz zu minimieren. Ist in einer Region gerade wenig los, zum Beispiel um 4 Uhr morgens, werden die Daten der letzten 24h durch einen **Scheduler** an die **Nordh Pole Database** gesendet. Auf dieser zentralen Datenbank können die Elfen über Terminals in ihren Workshops dann Queries abfragen und Updates auf den Wünschen durchführen.

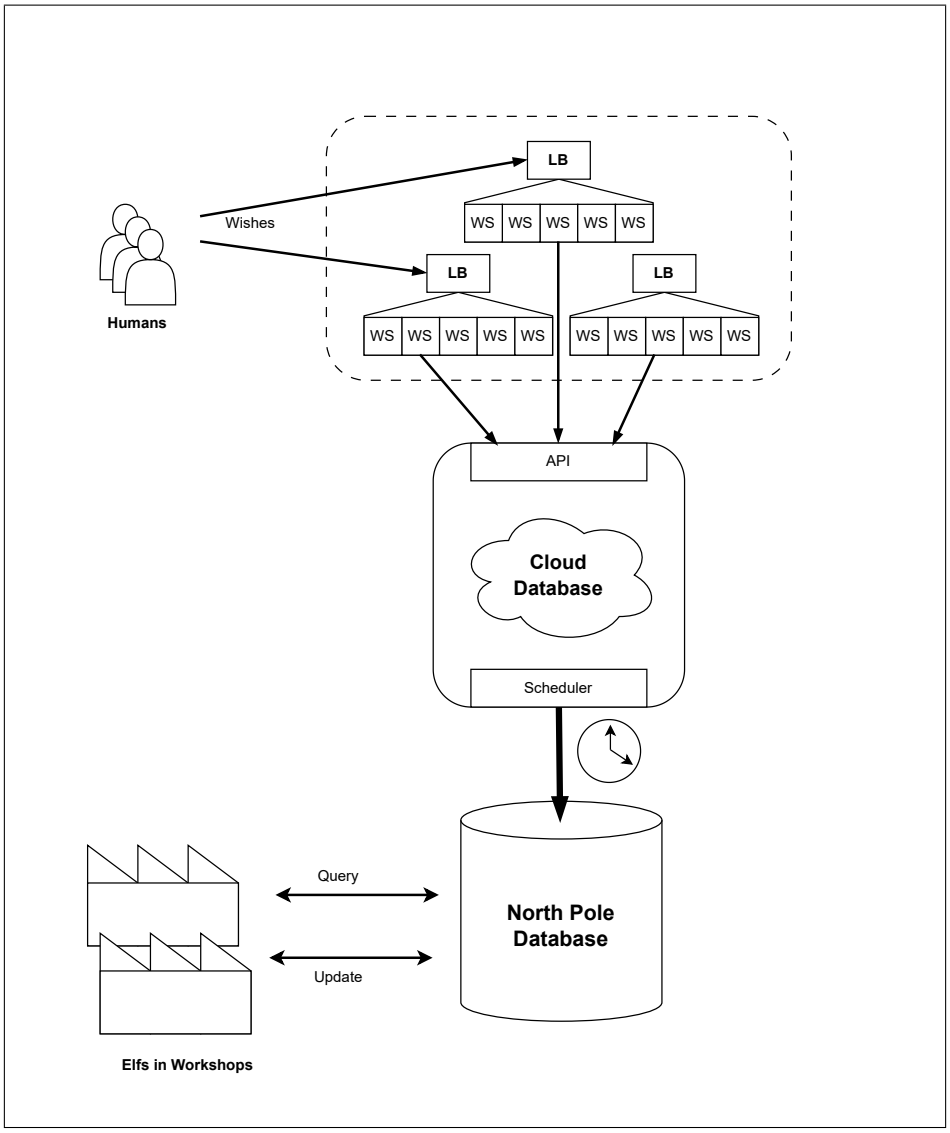

Abbildung 1: Systemarchitektur von XmasWishes

2 Konkretisierung

Für die einzelnen Microservices kann man Spring Boot verwenden. Man braucht APIS 1) von Mensch zu WishService, 2) von WishService zur Cloud-Datenbank bzw. von Cloud-Datenbank zu Nordpol Datenbank und 3) von Workshop zu Nordpol Datenbank (siehe Aufgabe 3). Über Kubernetes kann man horizontal und automatisch skalierbare Docker-Cluster generieren. Mithilfe der Google Kubernetes Engine kann man diese Cluster global verteilen. Mit Google Spanner kann man ein global verteiltes Cloud-basiertes Datenbanksystem führen. Eine automatische geschedulte Weitergabe der Daten ist bestimmt auch möglich, ich habe es in Aufgabe 3 aber in einer Platzhalter API implementiert. Für die Nordpol Datenbank reicht eine relationale Datenbank, die für Queries auf großen Datenmengen optimiert ist.

3 Prototyp

Das Repository befindet sich auf `https://github.com/amri-amri/XmasWishes`

3.1 Inhalt

Im folgenden wird der Inhalt der Packages erklärt. Für package `camel` siehe Aufgabe 4.

3.1.1 `model`

In diesem package befinden sich Klassen zur Modellierung der Wünsche. Ein Wunsch-Record hat 4 Felder:

1. Name: Name d. Wünschenden (eindeutiger Primärschlüssel)
2. Text: der formulierte Wunsch
3. Status: einer der 4 in der AUfgabenstellung genannten Zustände
4. etc: Platzhalter für etwaige Meta-Daten

Das Interface `WishRepository` erbt von Spring Boots `JpaRepository` und dient als Platzhalter für tatsächliche Datenbanksystem-Schnittstellen.

3.1.2 `northpole`

Die zwei auf „Controller“ endenden Klassen definieren die Endpunkte für die `Northpole`-API:

- `/persist`: POST-request zum Speichern eines neuen Wunsches in der Nordpol-Datenbank (wird von Cloud-Datenbank angesteuert)
- `/get/by/status`: GET-request als Beispielquery: Gibt alle (oder einen Teil aller) Records mit übergebenem Status aus (wird von Elf angesteuert)
- `/update/status`: PUT-request zum Updaten eines Wunsch-Status anhand des Namens (wird von Elf angesteuert)

In `DatabaseLogic` wird die Business Logic implementiert und `Northpole` ist der Eintrittspunkt zum Starten der API.

3.1.3 `datacloud`

Die `DatabaseService`-API ist eine Art Platzhalter bzw. umschließt die Cloud-Datenbank logisch gesehen. Business Logic ist in `DatabaseLogic` und die Endpunkte werden in `DatabaseController` definiert:

- `/perist`: POST-request wie bei der Nordpol-API
- `/flush`: GET-request zum manuellen entleeren einer Cloud-Datenbank-Instanz in die Nordpol-Datenbank

3.1.4 `wishservice`

`WishService` ist die API, die Menschen nutzen, um Wünsche zu senden. Der einzige von `WishController` definierte Endpunkt ist ein POST-request unter `/wish`. Da diese API noch kleiner ist als die anderen ist die Business Logic direkt im Controlle rimplementiert. Empfangene Wünsche werden **asynchron** an die Cloud weitergesendet.

3.1.5 `resources`

In der `application.properties` Datei werden Konfigurationen für die Microservices festgehalten. Für Development Zwecke hat eine Konfigurationsdatei für alle drei APIs gereicht.

`letters.csv` siehe Aufgabe 4.

3.2 Test

Ich habe einen simplen Pyhon-Client geschrieben, der asynchron Minimalbeispiel-Requests an einen einzigen `WishService` sendet. Bei 1.000 Anfragen „auf einmal“ kommt der Service noch klar. Bei 10.000 Anfragen ist er schon überfordert und keine Anfrage kommt durch. Das Senden der Anfragen dauert in allen Fällen 2 Sekunden.

4 Apache Camel

In package `camel` befinden sich eine Hauptklasse `CamelApp` und eine Klasse `CamelRoute`, die von Apache Camels `RouteBuilder` erbt. Die Route beginnt mit dem Lesen einer CSV-Datei, deren Pfad als Programmargument mitgegeben wird. Standardmäßig ist dieser Pfad `src/main/resources`, worunter sich die Datei `letters.csv` befindet. Die CSV beinhaltet Einträge der Form `name;text`. Eine CSV wird in der Route als `List<Wish>` zusammengefasst und per POST-request an die Nordpol Datenbank gesendet.