Brno University of Technology
Faculty of Information Technology

PDB
Night snack
Final report

December 12, 2021

Svätopluk Hanzel

1 Project overview

The goal of this project is to implement a simple system that can act as a demo of a food delivery company specialized in late-night deliveries for programmers.

Since this is a demo application, the real-world usability is quite limited and this project mainly focuses on properly implementing CQRS and event sourcing patterns and using the correct DB systems for various types of data.

1.1 Application principle

Main principle of the application can be described as follows: there are restaurants, which have menu items organized into categories, which can be ordered by customers. Since the menu items' price, customer's location and delivery status must be freezed in each order, the orders are saved separately with all the data in a single object.

The main objective in this project was to optimize the application for scalability with regards to reads. The most common operation in such

2 Data

As described in Application principle (1.1), the application works with `Restaurants`, `Menu categories`, `Menu items`, `Orders`.

The first three are saved in CockroachDB for validation and administration purposes. These are read only by Command handlers. The Restaurant is then also saved in MongoDB along with its categories and items. This is done to optimize reads as more people would be browsing restaurant menus than updating them.

Figure 1: Relations betweeen entities in CockroachDB

2.1 MongoDB

MongoDB is used as NoSQL database to store Events, Restaurants, Stock and Orders. All of them are loaded from events to ensure consistency.

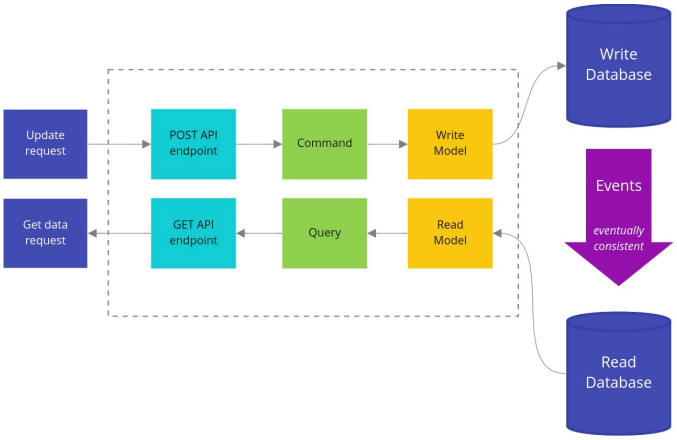

Figure 2: Principle of CQRS

2.2 Data consistency

Since the data is distributed across various databases, the event sourcing pattern has been used. All commands generate events, which are saved into a persistent storage – event store. All of these events are versioned. This ensures consistency since the aggregator cannot be updated if the updatee version doesn't correspond to the current version plus one.

3 Architecture

The project in its current state consists of one main service – *Snacker*. Which further encapsulates these gRPC services:

1. SnackerService
2. RestaurantCommandService
3. RestaurantQueryService
4. StockService
5. OrderService

All of these are available as gRPC services on the configured gRPC port or via HTTP gateway using simple REST API (see section 5 for more details).

Furthermore these services contains handlers for each RPC call (command). All Commands, Queries and Events are defined in protobuf to ensure consistent data types across systems.

All Commands are validated inside the handler and after that generate Events, which are both saved to Event store and sent via Event bus using

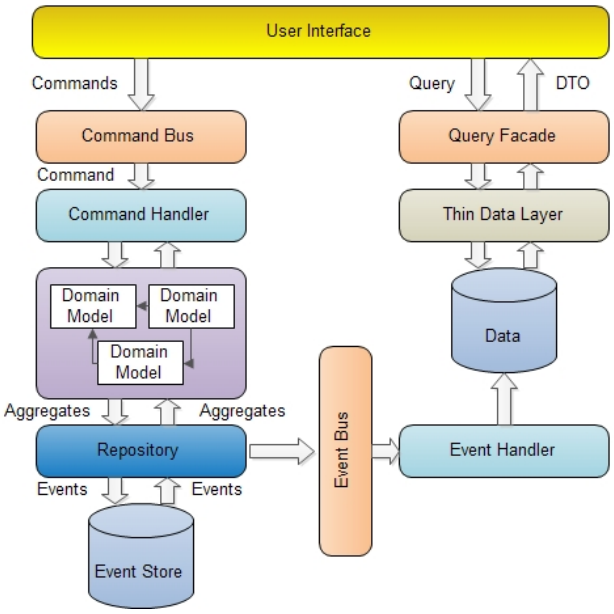

Figure 3: Event sourcing

NATS[1]. These events are then listened for and handled by any systems which manage them. Thanks to storing the Events in Event store, the systems, which are not online during the Event generation can then load them from Event store, thus ensuring eventual consistency (see 2.2).

4 How to use

The complete application is self-contained and thanks to using Docker[2] containers, it is easy to run using Docker compose[3]. For simplicity, one can use the prepared `make dev` command, which start all services.

5 API documentation

The API documentation can be generated using the `make docs` or `make run-docs` commands. This will generate Swagger[4] documentation from the code and the latter will also start a small HTTP server with GUI.

[1] `https://nats.io/`
[2] `https://www.docker.com/`
[3] `https://docs.docker.com/compose/`
[4] `https://swagger.io/`

6 Conclusion

The implemented project is a slightly cut-down version of the proposed solution due to technical difficulties few days before deadline. It however still implements CQRS pattern with event sourcing and eventual consistency making it highly scalable and consistent.