Bus Drivers Schedule Microservice Design

1 Introduction

The design is for the technique evaluation of microservice developer. Thus, I will adopt microservice architecture.

1.1 Purpose

This document is to design the microservice application for Bus Driver Schedule System in the city.

- A bus has a capacity, a model, make, and an associated driver.
- A driver has a first name, last name, social security number and an email
- A schedule links the different buses with drivers on a given day/time.

A list of operations required by this application:

- The application should allow a user to manage the schedule and add new entries for buses, drivers and assign shifts to drivers.

> *For example, Driver John, will be driving Bus XYZ on Wednesday between 8:00 am and 2:00 pm.*

- The application should allow a user to retrieve the schedule for every driver for a given week.
- The application should allow a user to retrieve the schedule for every bus for a given week.

1.2 References

Requirements Document: https://github.com/b-yond-infinite-network/amaze-us/tree/master/intermediate/challenge-1

1.3 Important Terms

Term	Brief Definition
API Gateway	
Microservice architecture	
Spring Cloud Spring Security Spring JPA	

1.4 Tech stack

- Spring cloud: API gateway
- Spring Boot
- Spring Security: oauth2 and role based control
- Spring Kafka: microservice communication
- Kafka
- Zookeeper
- Postgres: database
- Docker Compose to link the containers.

2 Functional Summary

2.1 Usage Scenarios

The following scenarios are related to use of Bus Driver Schedule Application

The application should allow a user to manage the schedule and add new entries for buses, drivers and assign shifts to drivers.

Scenario 1: User adds a new driver

Scenario 2: User adds a new bus with **an associated** driver

Scenario 3: User adds the schedule, start time (time stamp), end time(time stamp), day in a week,

Scenario 4: A user retrieves a driver's week schedule.

Scenario 5: A user retrieves a bus's week schedule.

2.2 **Assumption**

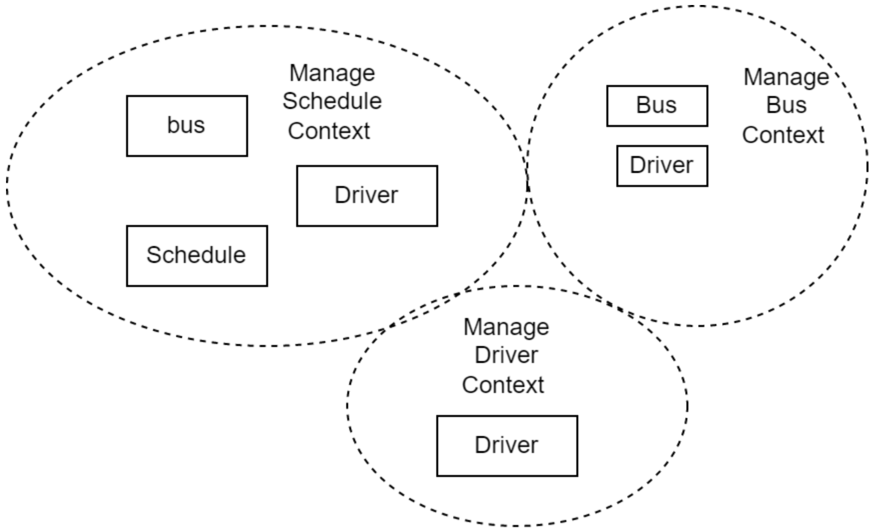
- User can only add a bus with an existing driver
- The bus and driver has the same schedule every week.
- No duplicated driver Name can be found in the driver data base.

2.3 High-Level Design

Follow the DDD, we identify 3 context in the application

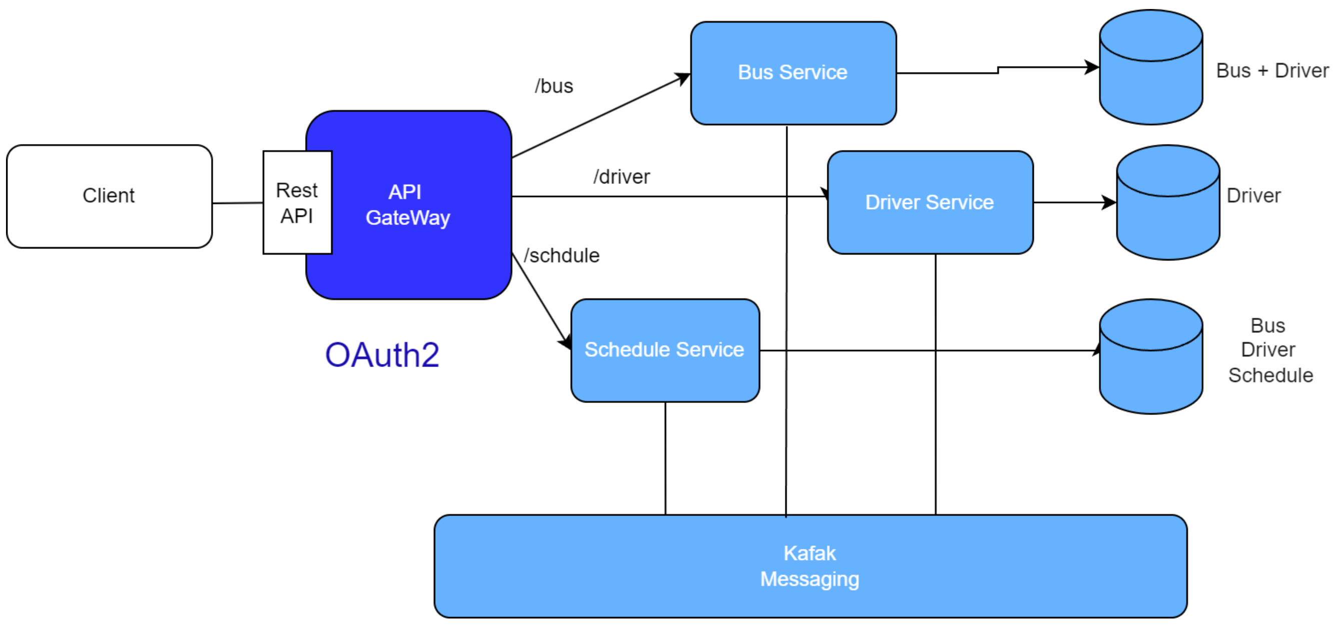
- Add a driver context, will involve driver entity
- Add a bus context, will involve driver and bus entities, where driver has a driverId
- Manage schedule context, involve driver, bus and schedule entities

We break the application into 3 microservice for each context.



2.4 Application Architecture

Following the above analysis, we design the following architecture.



2.5 Related Entities

2.5.1 Driver

Driver

```
public class Driver {
    private Long id; //primary key
    private String firstName;
    private String lastName;
    private String socicalNum;
    @Email
    private String email;
}
```

2.5.2 Bus

Bus

```
public class Bus {
    private Long id;
    private Integer capacity;
    private String model;
    private String maker;
    private Driver driver;
}
```

2.5.3 Schedule

Schedule

```
public class Schedule {
    @Id
    @Column(name = "ID", nullable = false)
    private Long id; //primary key

        @ManyToOne
    private Bus bus;

    @ManyToOne
    private Driver driver;

        Date start;

        Date end;

        DayofWeek day;
}
```

2.6 Kafka Topics

The application has 3 microservices: Driver, Bus, and Schedule and kafka ecosystem is used for the communication. We design 2 kafka topics for the messaging system: Topic "driver" and "bus".

Topic	Publishing	Subscriber	Comment
"Driver"	Driver microservice	Bus microservice	BUS add the driver in its database **only id is needed**. This is for the validate purpose. When add a bus, we will check if the driver exists or not.
		Schedule microservice	SCHEDULE add driver in its database and **only id and firstName and lastName** are needed.
"Bus"	Bus Microservice	Schedule Microservice	Schedule add the bus in its database and only **bus id** is needed.

- The Driver and Bus services send topic "driver"/"bus" messages to Kafka with using the 'KakfaTemplate'.
- The Bus and Schedule listen to "driver"/"bus" message with '@KafkaListener' and extract the information needed.

2.7 Rest APIs

The following rest APIs in DEMS use the baseline time series data CRUD operations.

REST API Action	REST API Path	Comment
POST	/bus	Add a bus with a associated driver if driver does not exist, then an error message is thrown
POST	/driver	Add a driver
GET	/schedule/bus/{busId}	retrieve the driver week schedule
GET	/schedule/driver/{name}	retrieve the bus week schedule
POST	/schedule	schedule a driver to a bus

2.8 Design Trade-Off Analysis

I propose the following trade-off design.

1. Schedule microservice only listens to the "bus" topic and updates its bus table; where the bus table embeds necessary information of driver and has bus id. This option is better the current design, the bus object is bigger. The kafka message sizes are larger.
2. Use CQRS architecture pattern with event source. With this option, we can use a unified database. It is better than Current design. But

3 Data Flow

The project creates Docker containers. It uses 3 microservice.

3.1 Add a driver data flow

3.2 Add a bus data flow

3.3 Add a schedule data flow

3.4 Retrieve schedule data flow

4 Configuration and Deployment

The application is deployed on the docker and have the following containers:

- Kafka
- Zookeeper
- postgres
- bus
- driver
- schedule

A docker-composer.yml has been generated for deployment.

5 Known Issues

- The description of the application asked to associate the driver with bus. The current design assumed that only one driver is associated with the bus. Have to refract with the a list of driver associated with bus consider the reality.

6 Design Issues

#	Design Issue	Priority	Status	Target	Resolution Plan	Owner
1	The map between bus and driver		✓		Made an assumption that one bus can have one driver	
2	Add schedule by name		✓		Made an assumption that no duplicated driver name	