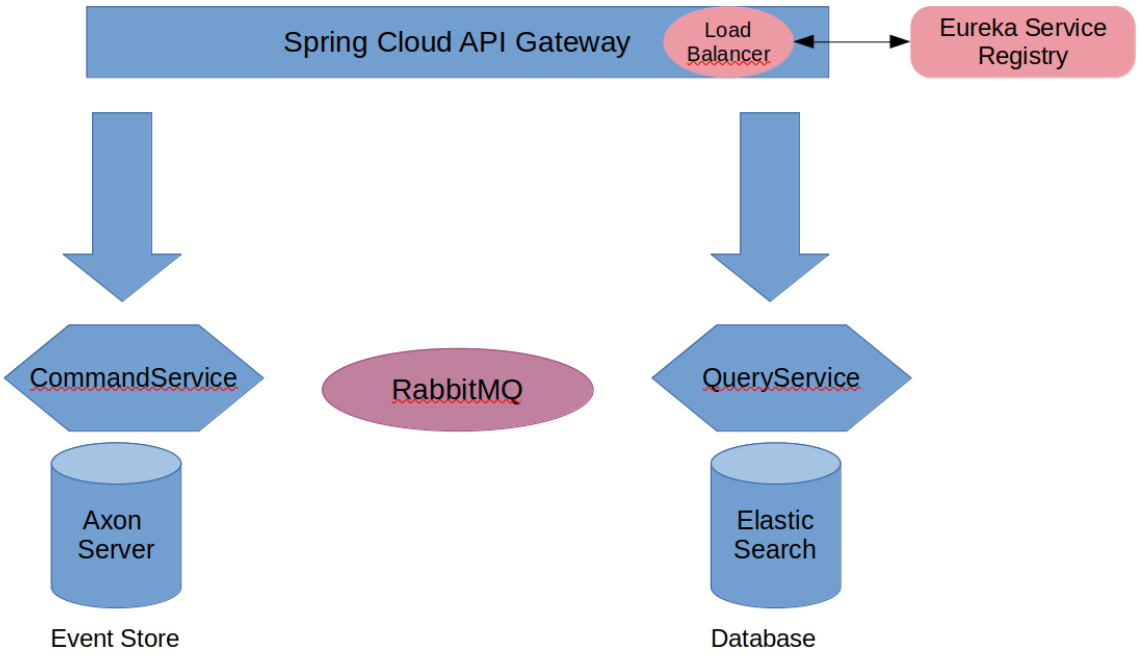
The project architecture uses multiple patterns :

- Event-Sourcing
- CQRS
- Event-Driven Design



I-API Gateway :

- It is responsible to route and map incoming HTTP requests from the client to the specific destination micro-service automatically. We just need to do the right configuration in application.properties file :

spring.cloud.gateway.discovery.locator.enabled=true

- It has a built-in Load Balancer
- It is a Eureka Client

Concerned dependency :

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-gateway</artifactId>

</dependency>

II-Eureka Server :

- Eureka Server is an application that holds information about all client-service applications. Every Microservice will register into the Eureka server and Eureka server knows all the client applications running on each port and IP address. Eureka Server is also known as Discovery Server.
- The code for main Spring Boot application class file is :

```
package com.tutorialspoint.eurekaserver;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.cloud.netflix.eureka.server.EnableEurekaServer;

@SpringBootApplication

@EnableEurekaServer

public class EurekaserverApplication {

public static void main(String[] args) {

SpringApplication.run(EurekaserverApplication.class, args);

}

}
```

- The code for Maven user dependency is :

```
<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-eureka-server</artifactId>

</dependency>
```

- By default, the Eureka Server registers itself into the discovery. We should add the followng configuration into application.properties file

```
eureka.client.registerWithEureka = false

eureka.client.fetchRegistry = false

server.port = 8761
```

III-Command Service :

Workflow of the command Service :

1- Receives commands to update the database (via rest API)

2- Create an event out of it.

3- Publish the event to the event store

4- Publish the event to rabbitmq broker

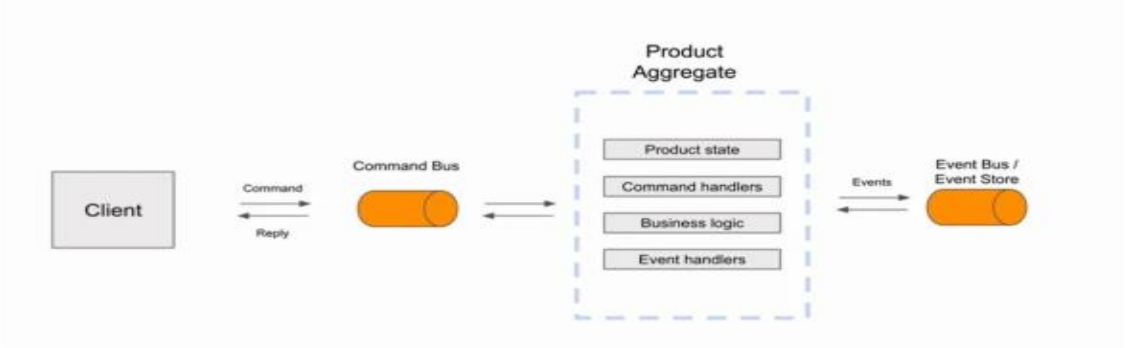
The command service is built following ***Event-Driven Architecture***.

- Event-driven architecture is a software architecture and model for application design. With an event-driven system, the capture, communication, processing, and persistence of events are the core structure of the solution. This differs from a traditional request-driven model.

Why Axon Framework?

- Axon Framework is a framework for building evolutionary, event-driven microservice systems, based on the principles of Domain Driven Design, Command-Query Responsibility Segregation (CQRS) and Event Sourcing.
- it provides you the necessary building blocks to follow these principles. Building blocks like Aggregate factories and Repositories, Command, Event and Query Buses, and an Event Store. The framework provides sensible defaults for all of these components out of the box.
- This set up helps you create a well-structured application without having to bother with the infrastructure. The main focus can thus become your business functionality.

Command Service Architecture:



IV-Query Service:

The workflow of the Query Service :

1. Receives read query via rest api

2. Query the database and return the result

3. All along, listen to published events in rabbitmq

4. Handle received events and updates the elasticsearch database

Why Elasticsearch?

Search has become a central idea in many fields with ever- increasing data. As most applications become data-intensive, it is important to search through a large volume of data with speed and flexibility. ElasticSearch offers both.

V-RabbitMQ:

RabbitMQ message broker is used to assure an asynchronous communication between the Command microservice and the Query microservice.

```
docker run --rm -it -p 15672:15672 -p 5672:5672 rabbitmq:3-management
```

- Configuration file for RabbitMQ:

```
@Configuration
public class MQConfig {
public static final String QUEUE = "event_queue";
@Bean
public Queue queue(){
return new Queue(QUEUE);
}
@Bean
public Jackson2JsonMessageConverter messageConverter(){
return new Jackson2JsonMessageConverter();
}
@Bean
```

```
public AmqpTemplate template(ConnectionFactory connectionFactory){
RabbitTemplate template = new RabbitTemplate(connectionFactory);
template.setMessageConverter(messageConverter());
return template;
}
}
```