Mphasis
The Next Applied

Java Microservices Hands-on Mastery - S1

Manpreet Singh Bindra
Senior Manager

Do’s and Don’t

- Login to GTW session on Time
- Login with your Mphasis Email ID only
- Use the question window for asking any queries

Welcome

1. Skill - Proficiency Introduction
2. About Me - Introduction
3. Walkthrough the Skill on TalentNext

Overall Agenda

- Understanding CQRS and Event Sourcing Pattern
- Implementing CQRS with Event Sourcing with Axon Framework
- Distributed Transaction in Java Microservices using SAGA Pattern
- Introduce Docker and state its benefit over VM
- Understanding the Architecture of Docker and terminologies
- Describe what is Container in Docker, why to use it, and its scopes
- Deploy Microservice in Docker
- Deploy Microservice with H2 to AWS Fargate
- Implement Centralized Configuration Management with AWS Parameter Store
- Implement Auto Scaling and Load Balancing with AWS Fargate

Day - 1

Day – 1 Agenda

- Microservice vs Monolithic application
- Event-Driven Microservices
- Transactions in Microservices
- Choreography-Based Saga
- Orchestration-Based Saga
- Command Query Responsibility Segregation
- Types of Messaging in CQRS Pattern
- CQRS and Event Sourcing
- Building microservices with Spring Boot

Day - 1

Microservice vs Monolithic Application

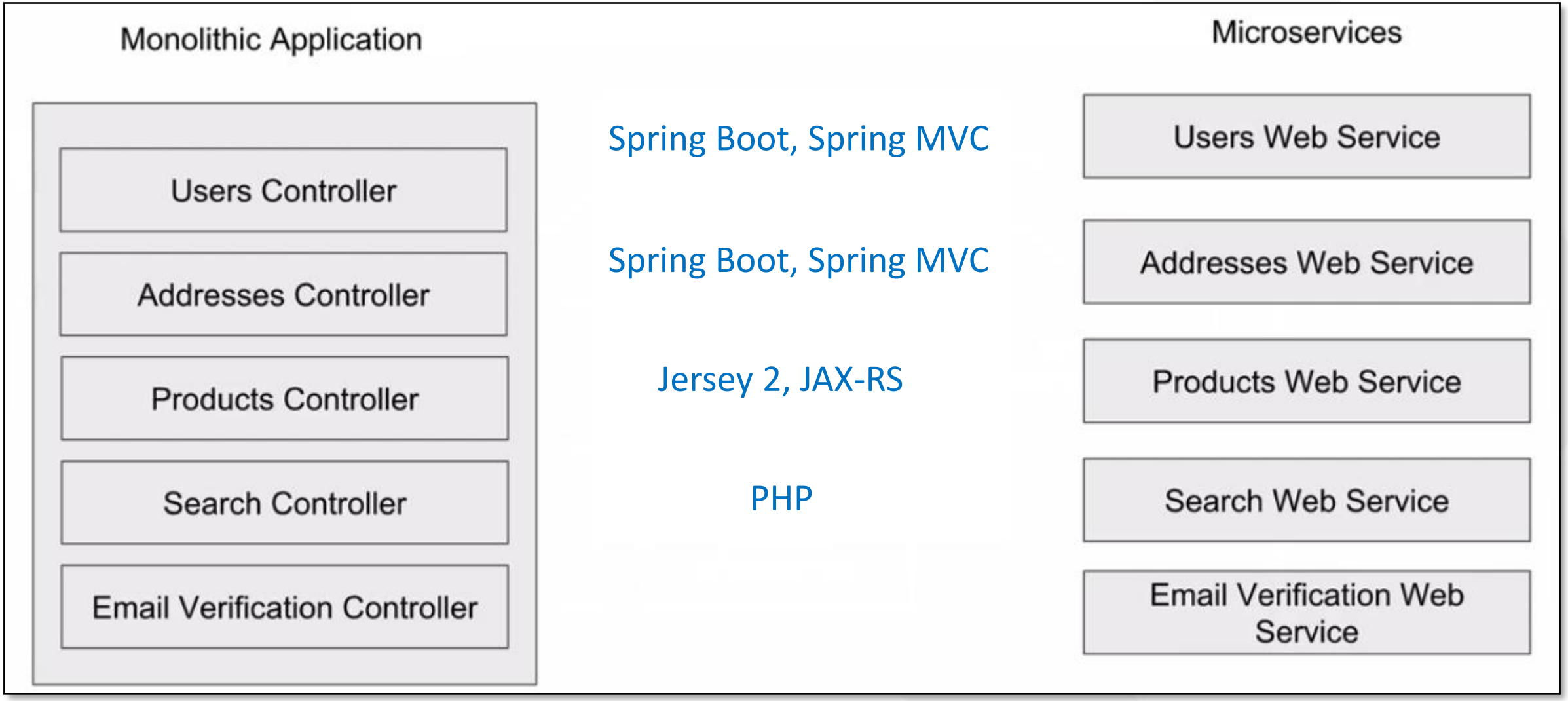
What are Microservice?

- Microservices are a software development technique – a variant of the service-oriented architecture (SOA) architectural style that structures an application as a collection of loosely coupled services…
- In a microservices architecture services are fine-grained…
- The benefits of decomposing an application into different smaller services is that it improves modularity. This makes the application easier to understand, develop, test, and become more resilient to architecture erosion.
- It parallelized development by enabling small autonomous teams to develop, deploy and scale their respective services independently.

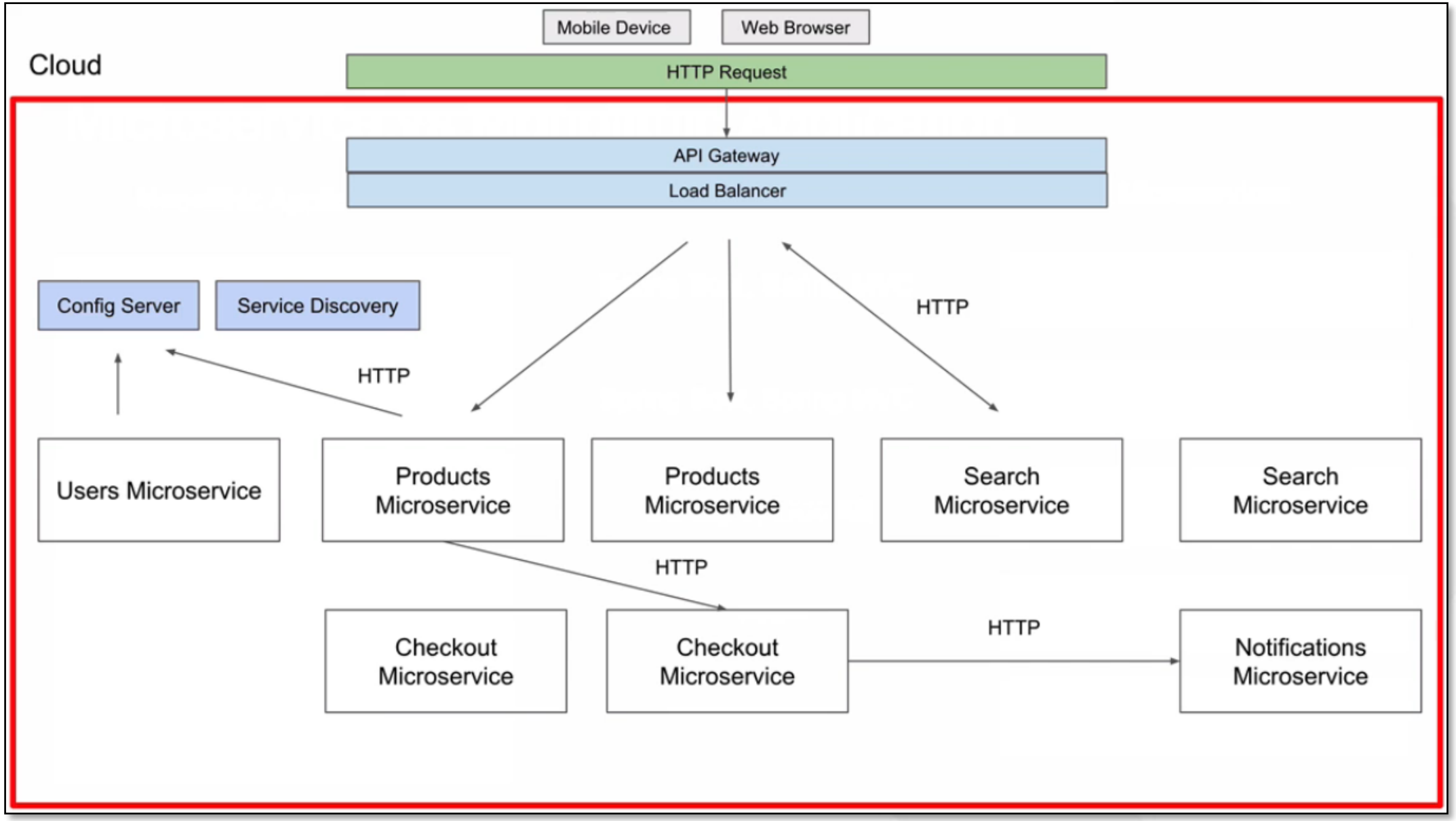
What are Microservice?

- A Web Service
- Small and Responsible for one thing (Search, Password Reset, Email Verification)
- Configured to work in the cloud and is easily scalable.

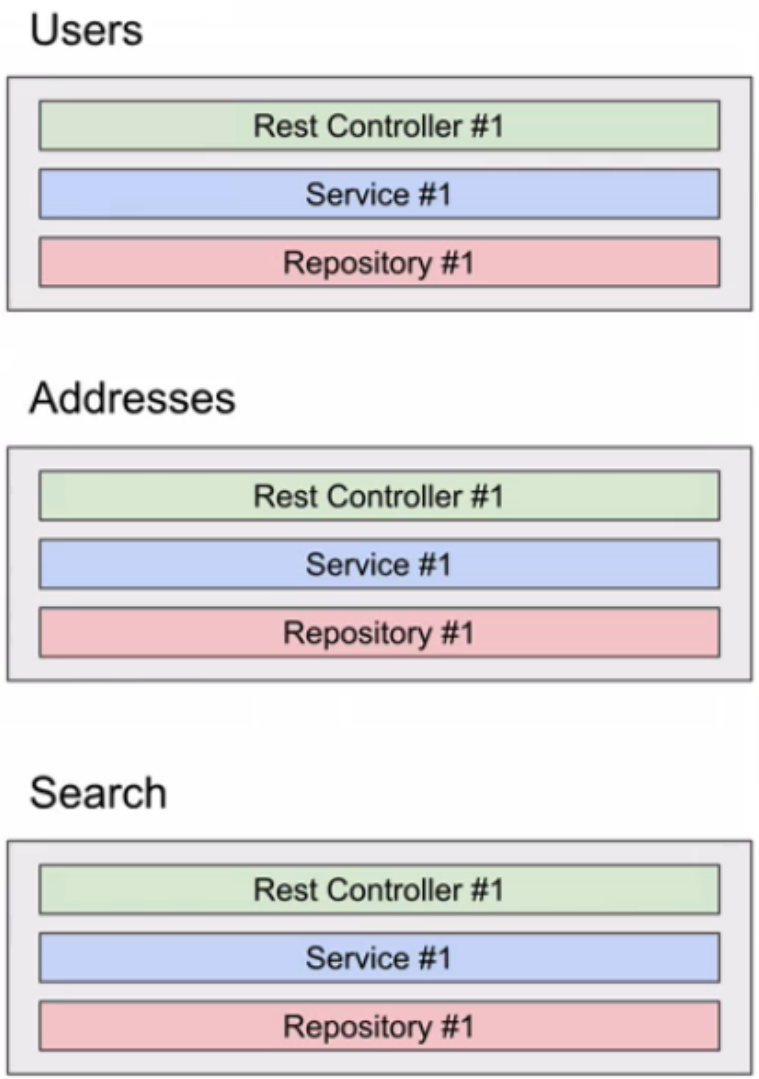
Microservice vs Monolithic Application



Application Diagram



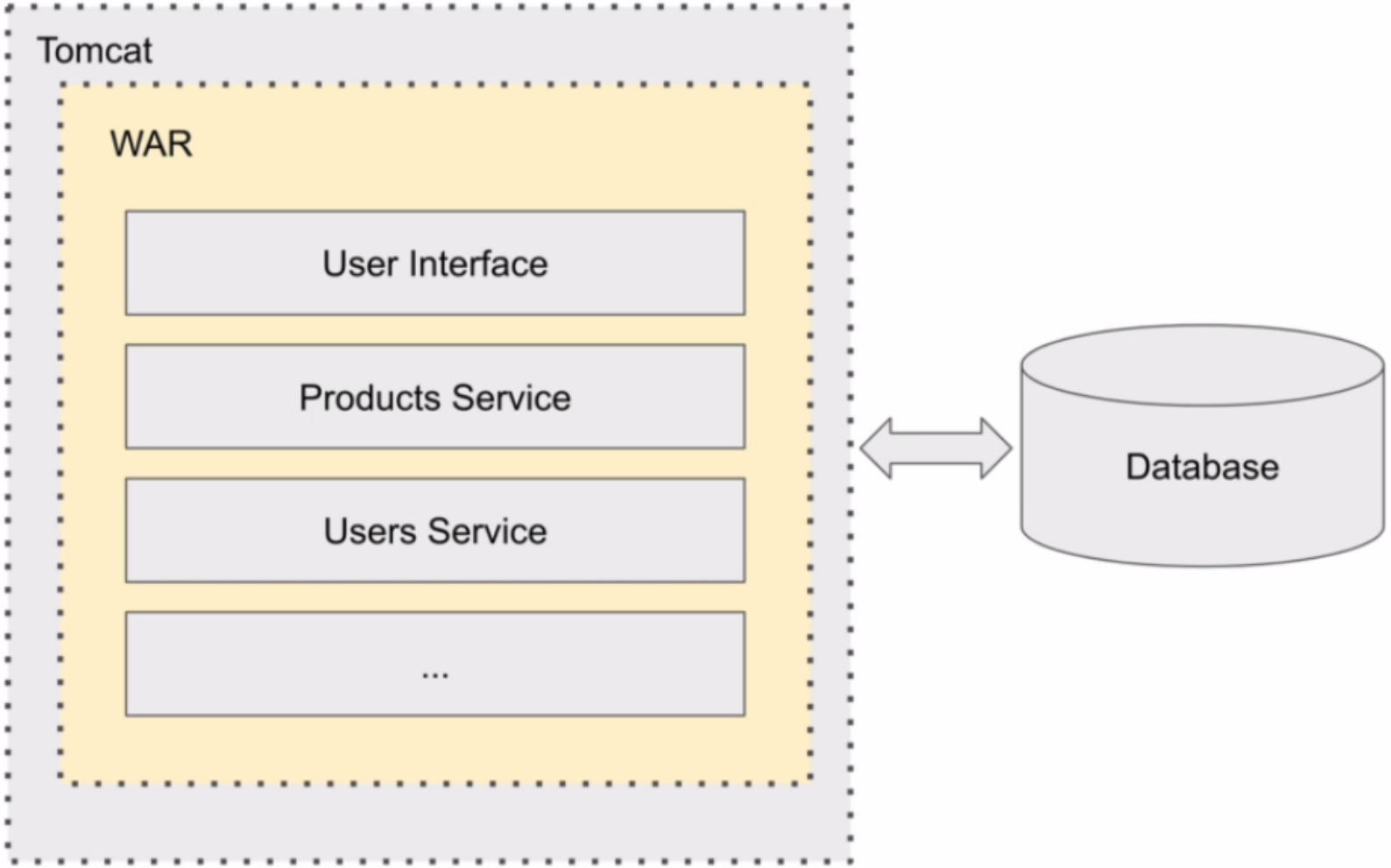
Rest Controller - Services



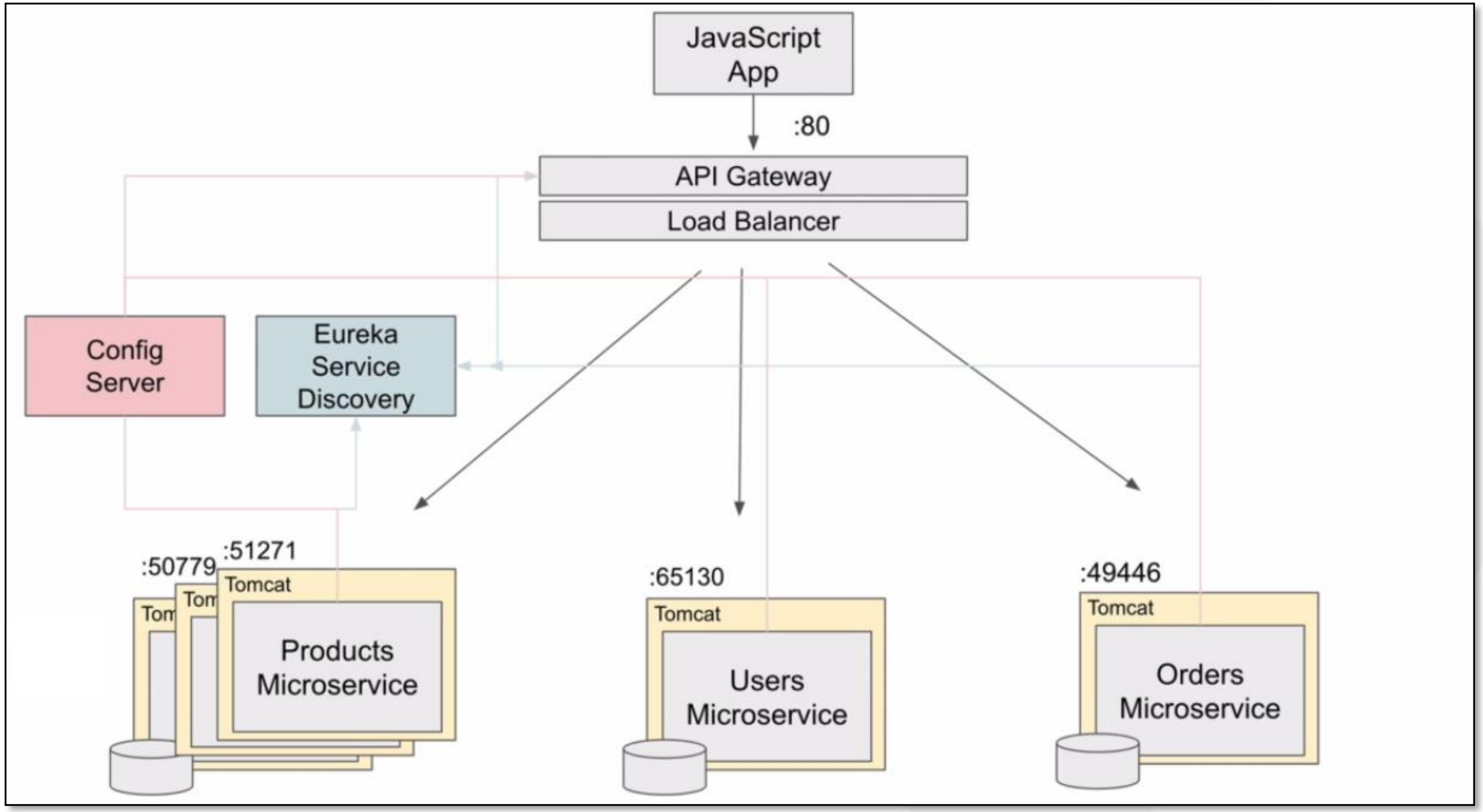
Day - 1

Microservices Architecture Overview

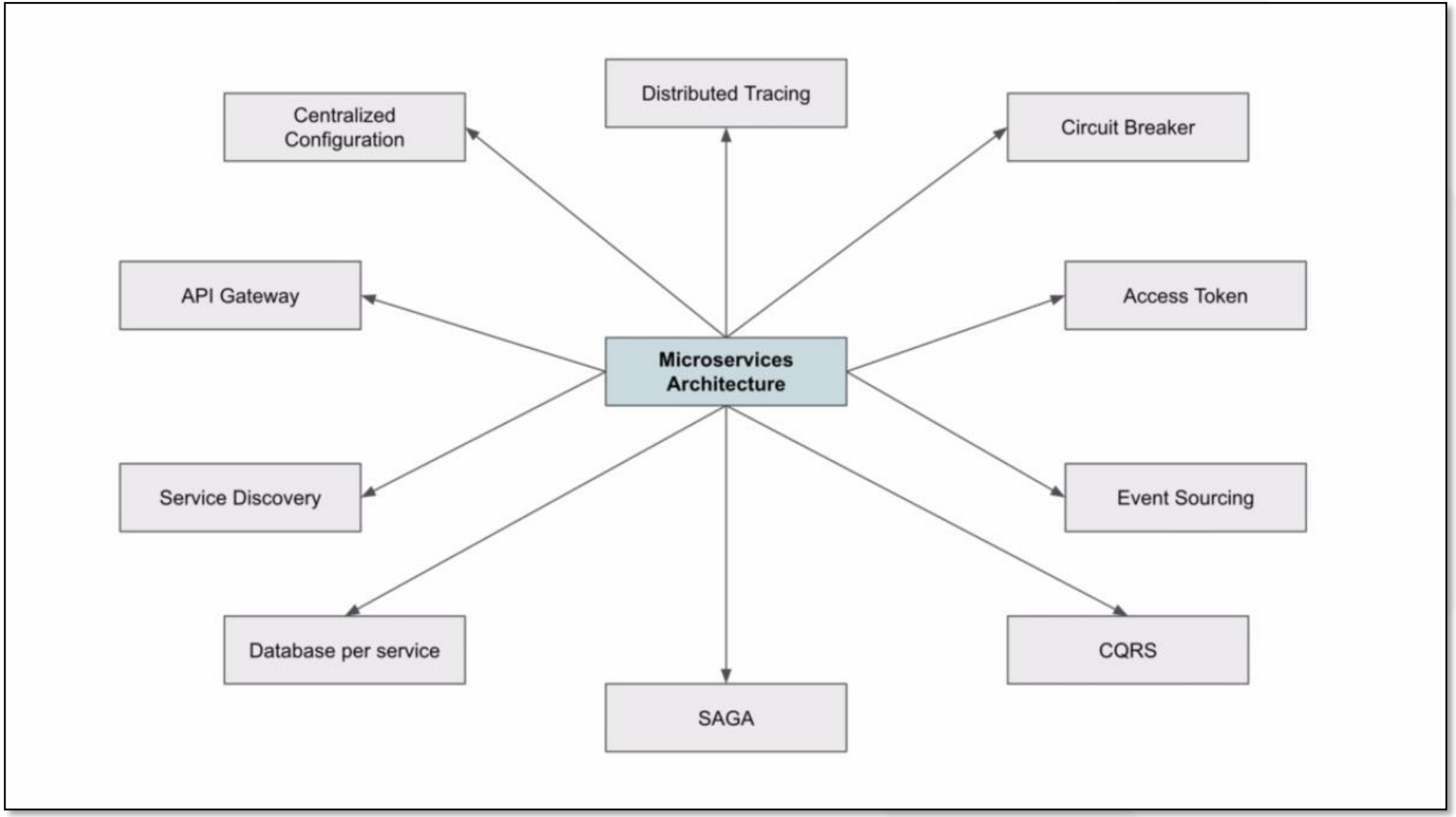
Monolithic Architecture



Microservices Architecture



Pattern Diagram



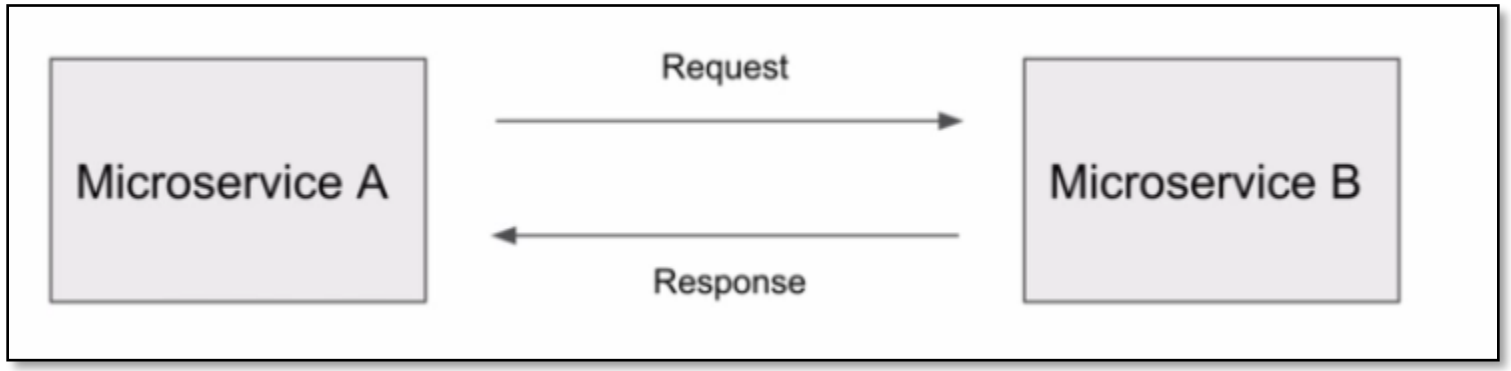
Day - 1

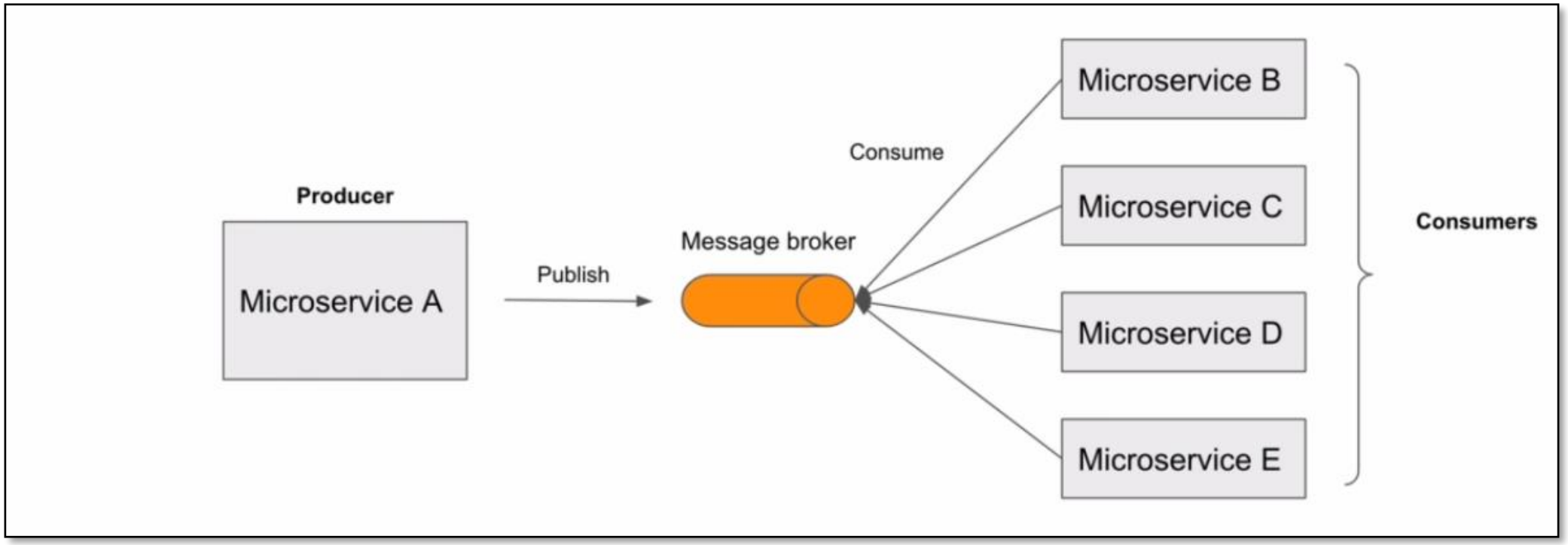
Event-Driven Microservices

Event-Driven Microservices

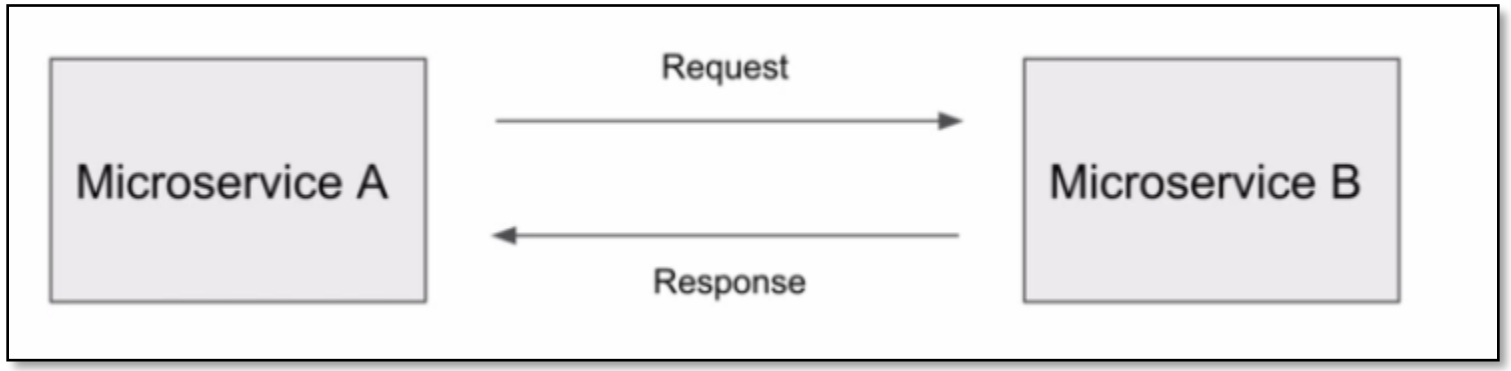
- It’s asynchronous.
- If your microservices are simple, follow request-response approach.
- If you don’t know how many microservices can be added/removed as consumer, we can prefer the event-driven microservices/message-driven microservices.

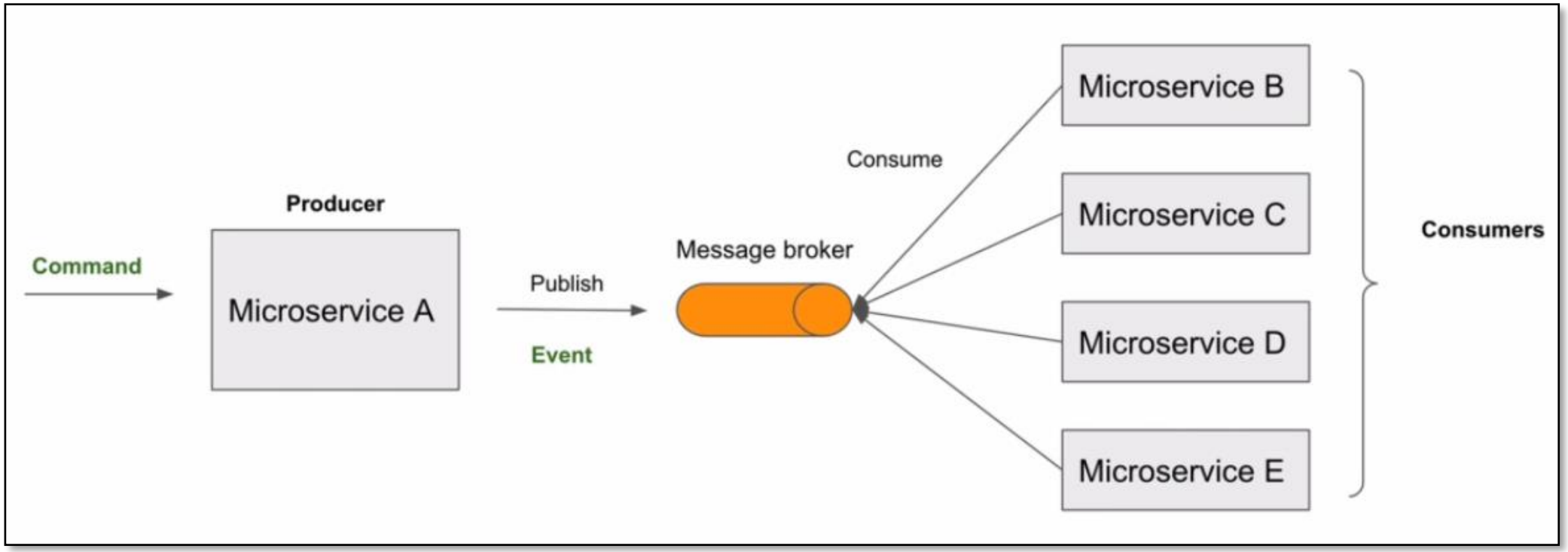
Event-Driven Microservices



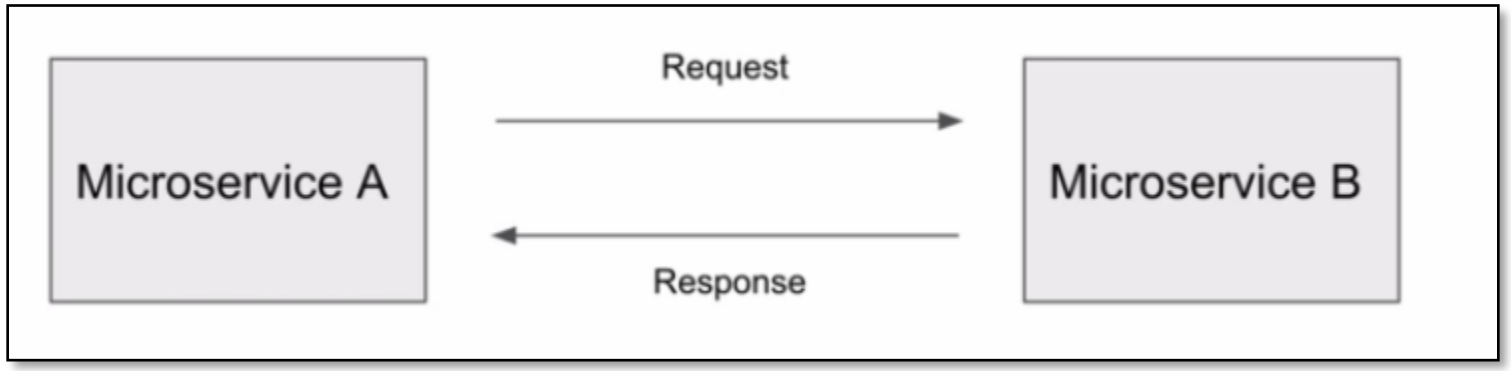


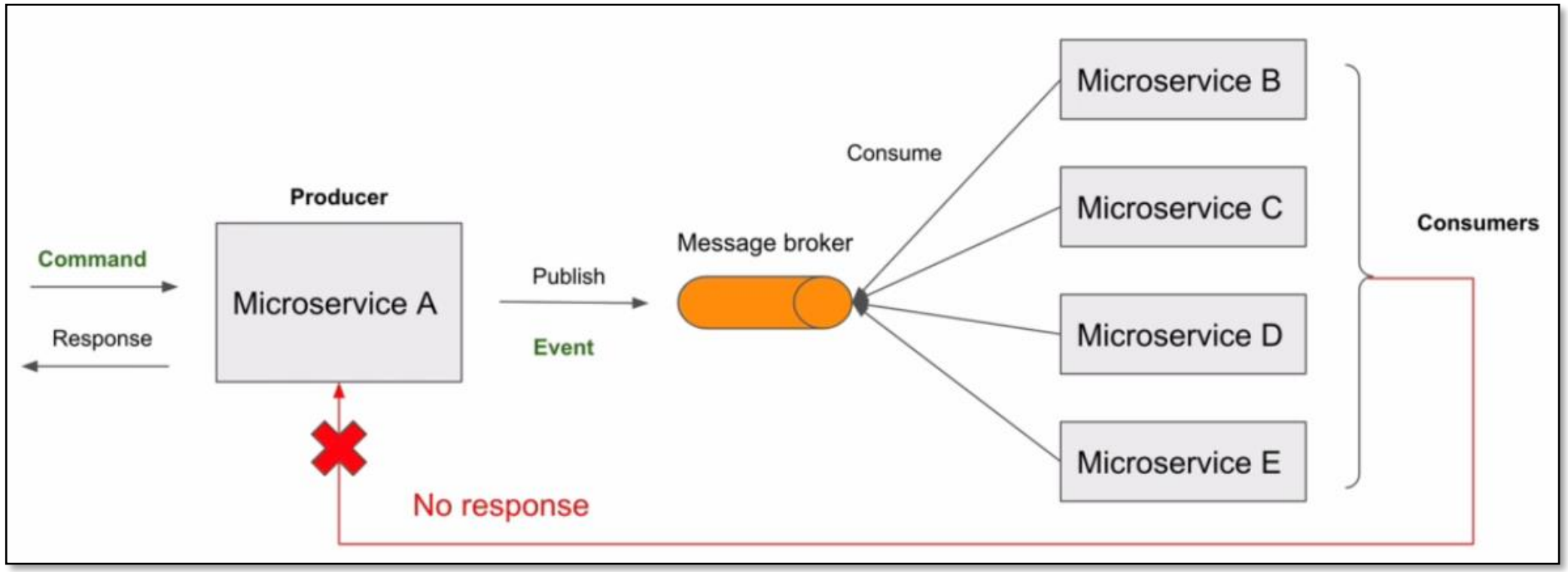
Event-Driven Microservices





Event-Driven Microservices



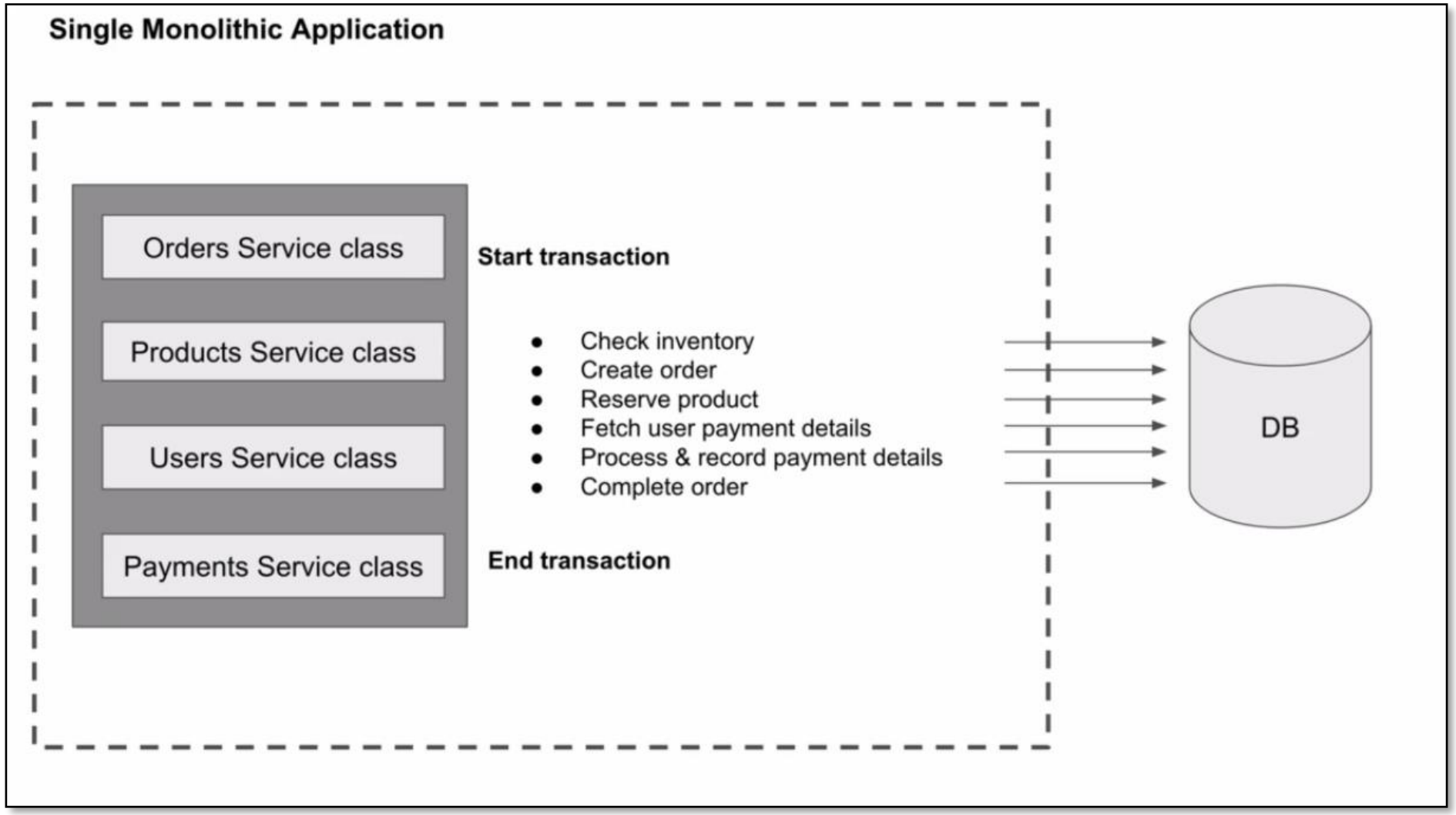


Day - 1

Transactions in Microservices

Transactions in Microservices

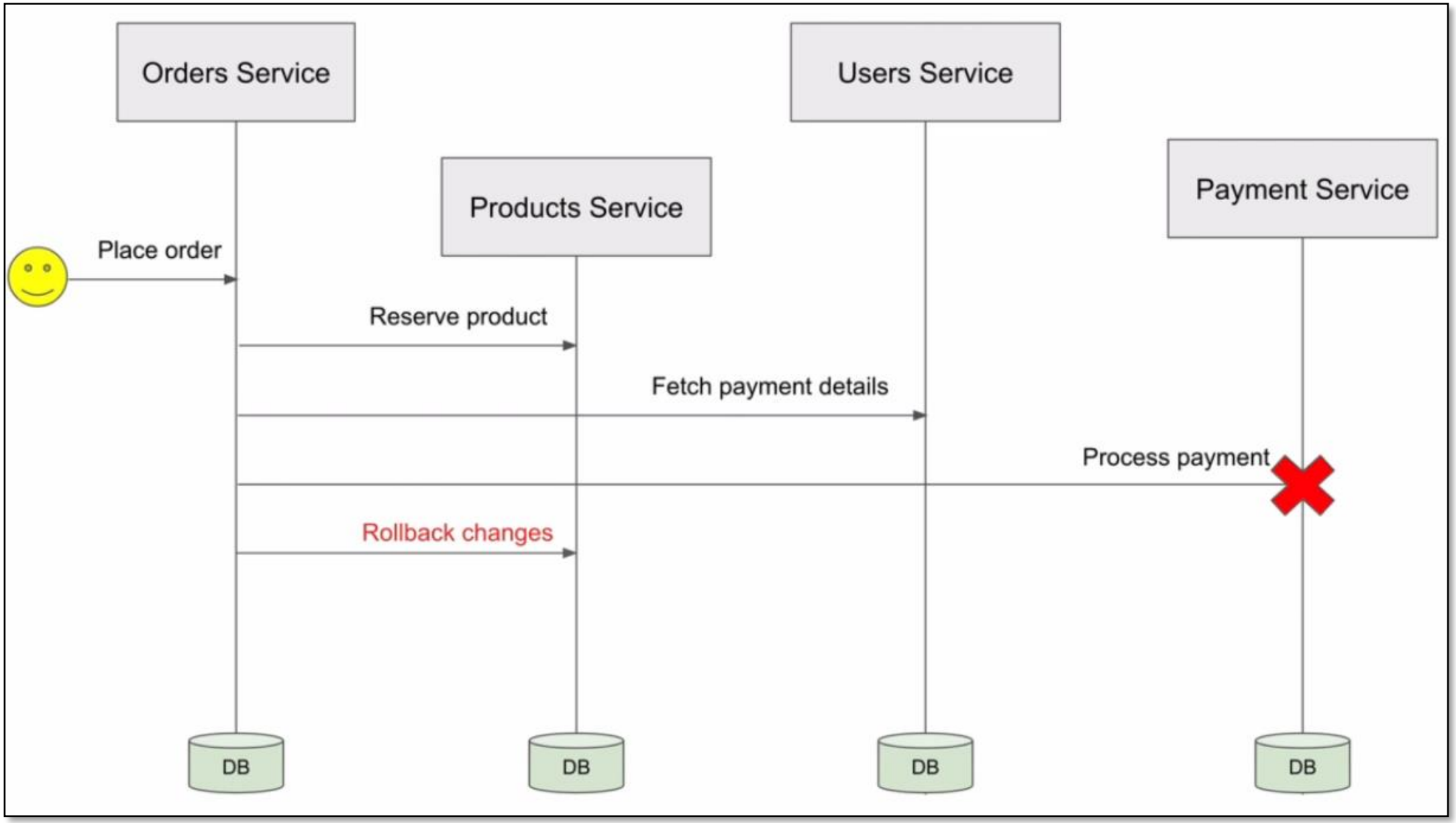
- In monolithic applications, implementing the transaction is very easy.



ACID Transactions

- Atomicity
- Consistency
- Isolation
- Durability

Transactions in Microservices



SAGA Design Pattern

- Is a way to manage data consistency across microservices in distributed transactions scenarios.
- There are two different ways to implement SAGA patterns:
  - Choreography-Based
  - Orchestration-Based

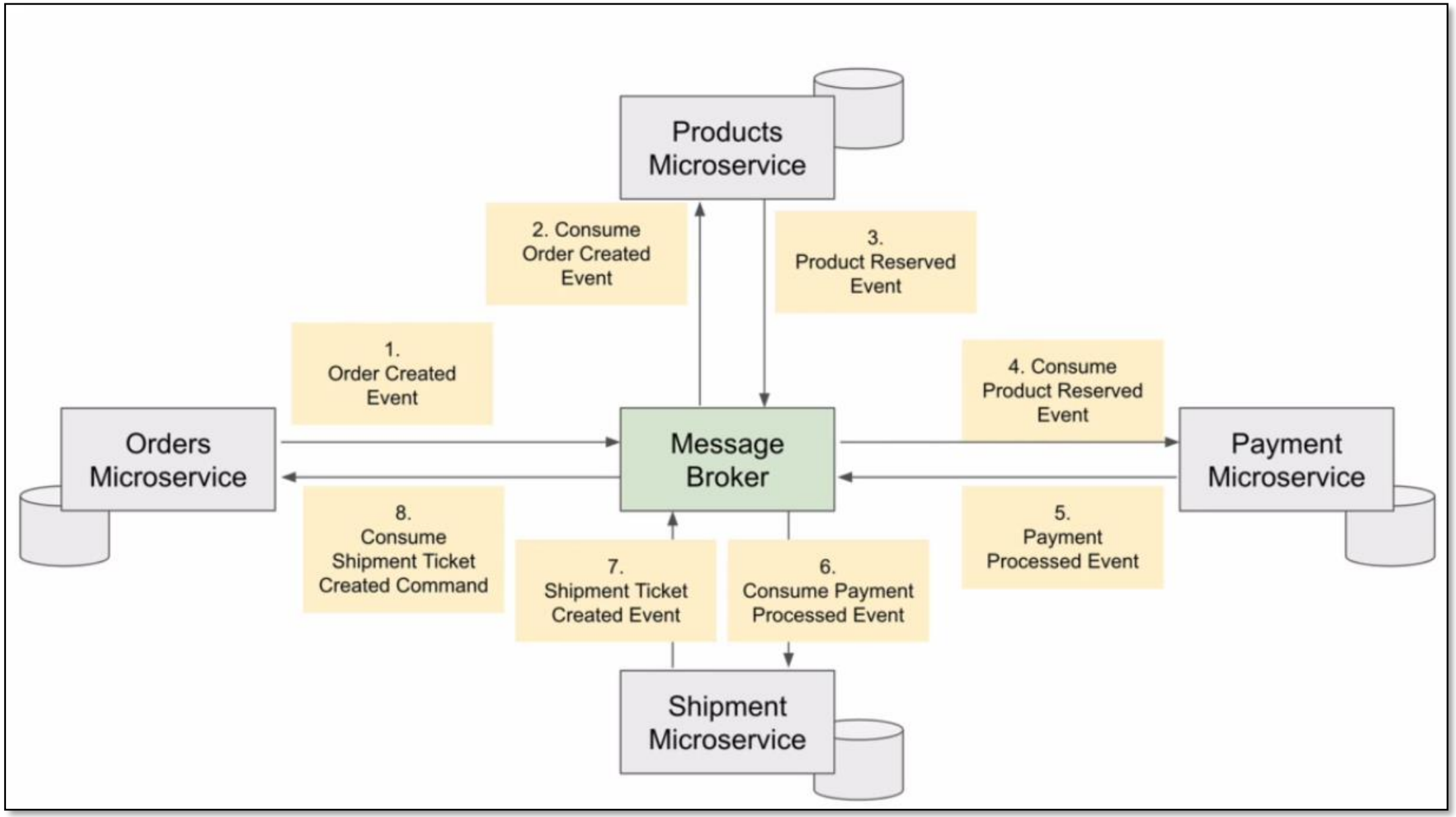
Day - 1

Choreography-Based SAGA

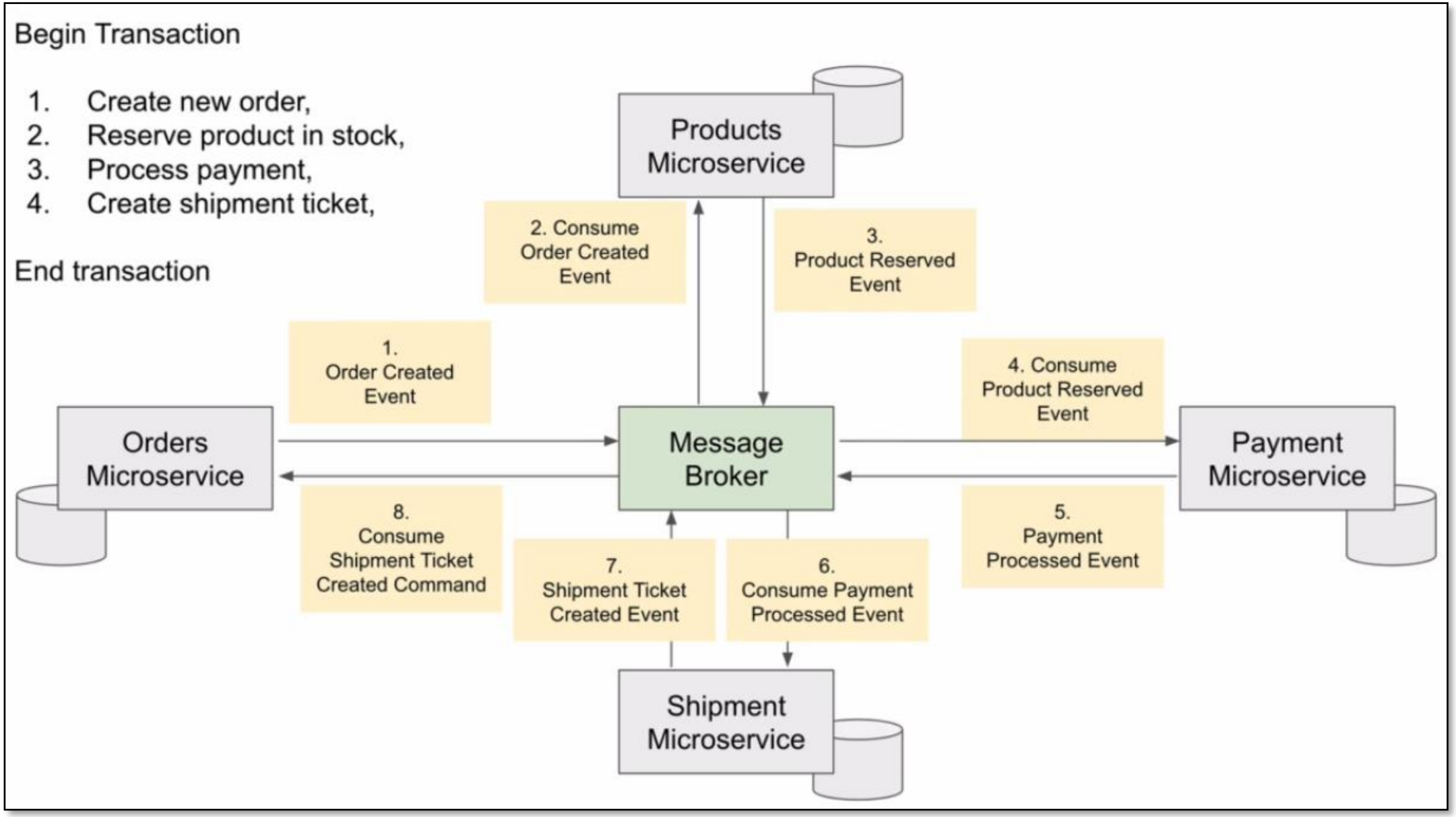
Choreography-Based SAGA

- In Choreography-Based SAGA, microservices communicate with each other by exchanging events.
- When microservices perform operations, it publishes the event message to a messaging system like Message Broker.

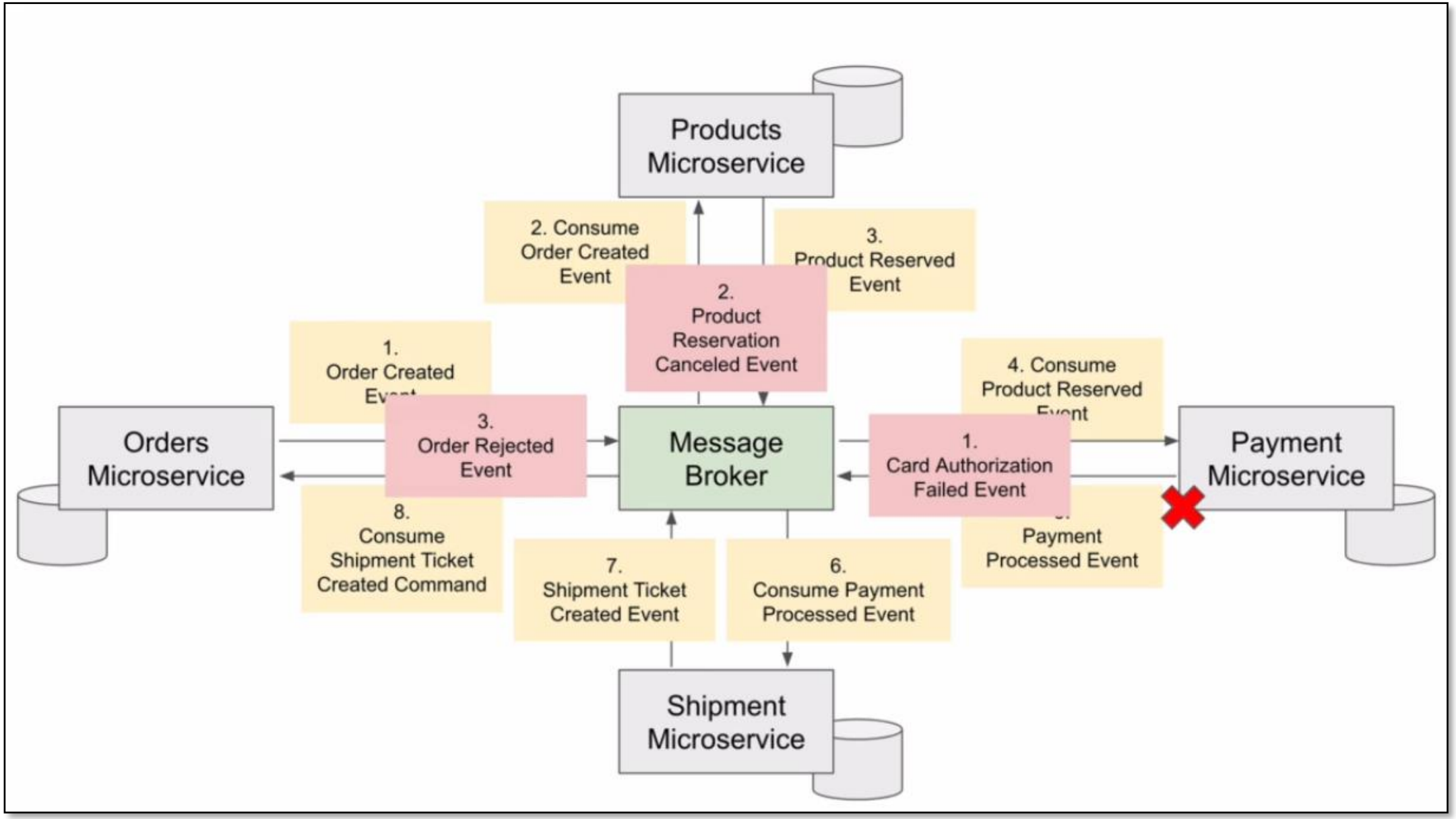
Choreography-Based SAGA



Choreography-Based SAGA



Choreography-Based SAGA



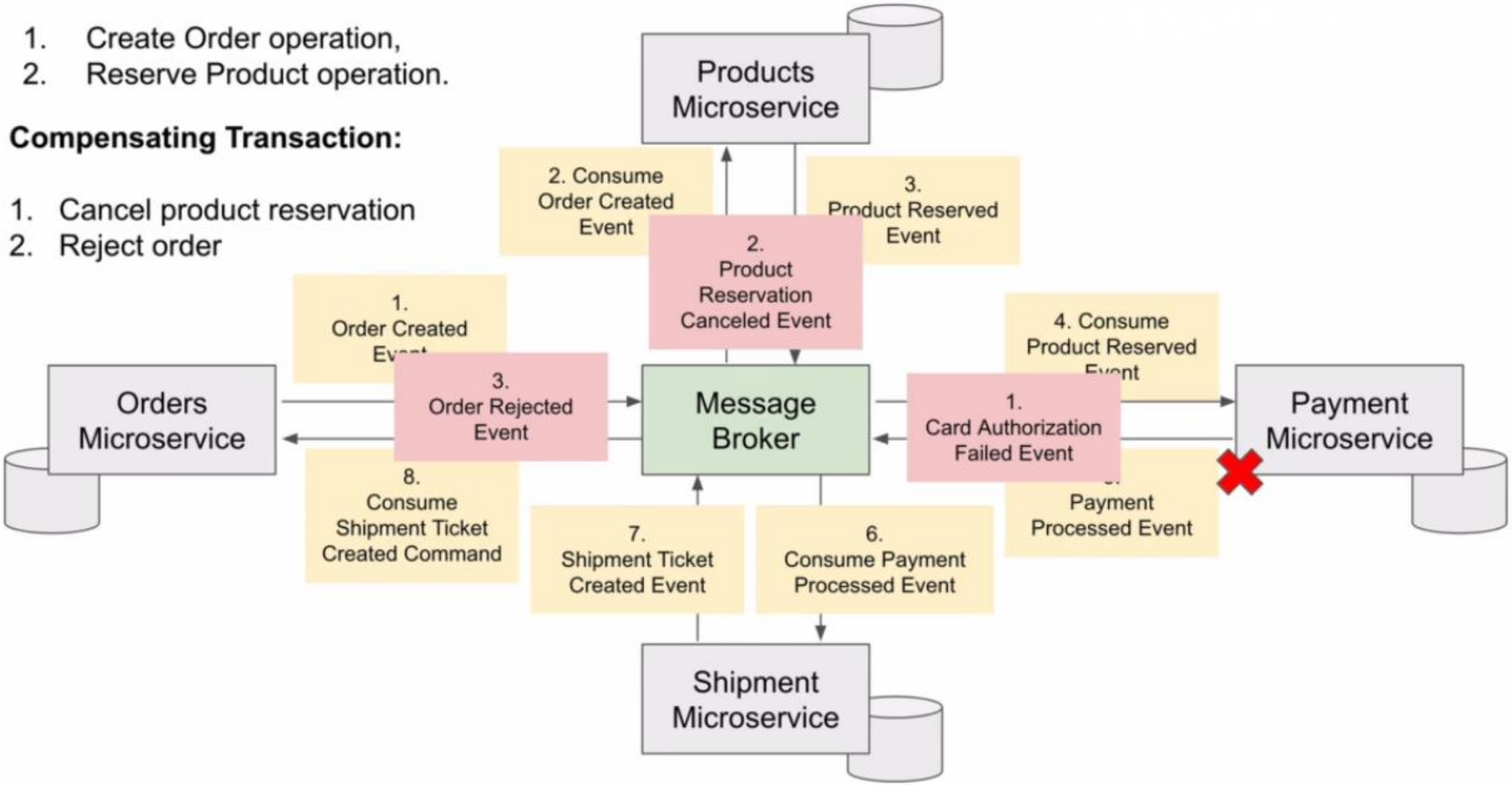
Choreography-Based SAGA

Initial Flow:

1. Create Order operation,
2. Reserve Product operation.

Compensating Transaction:

1. Cancel product reservation
2. Reject order



Choreography-Based SAGA

Initial Flow:

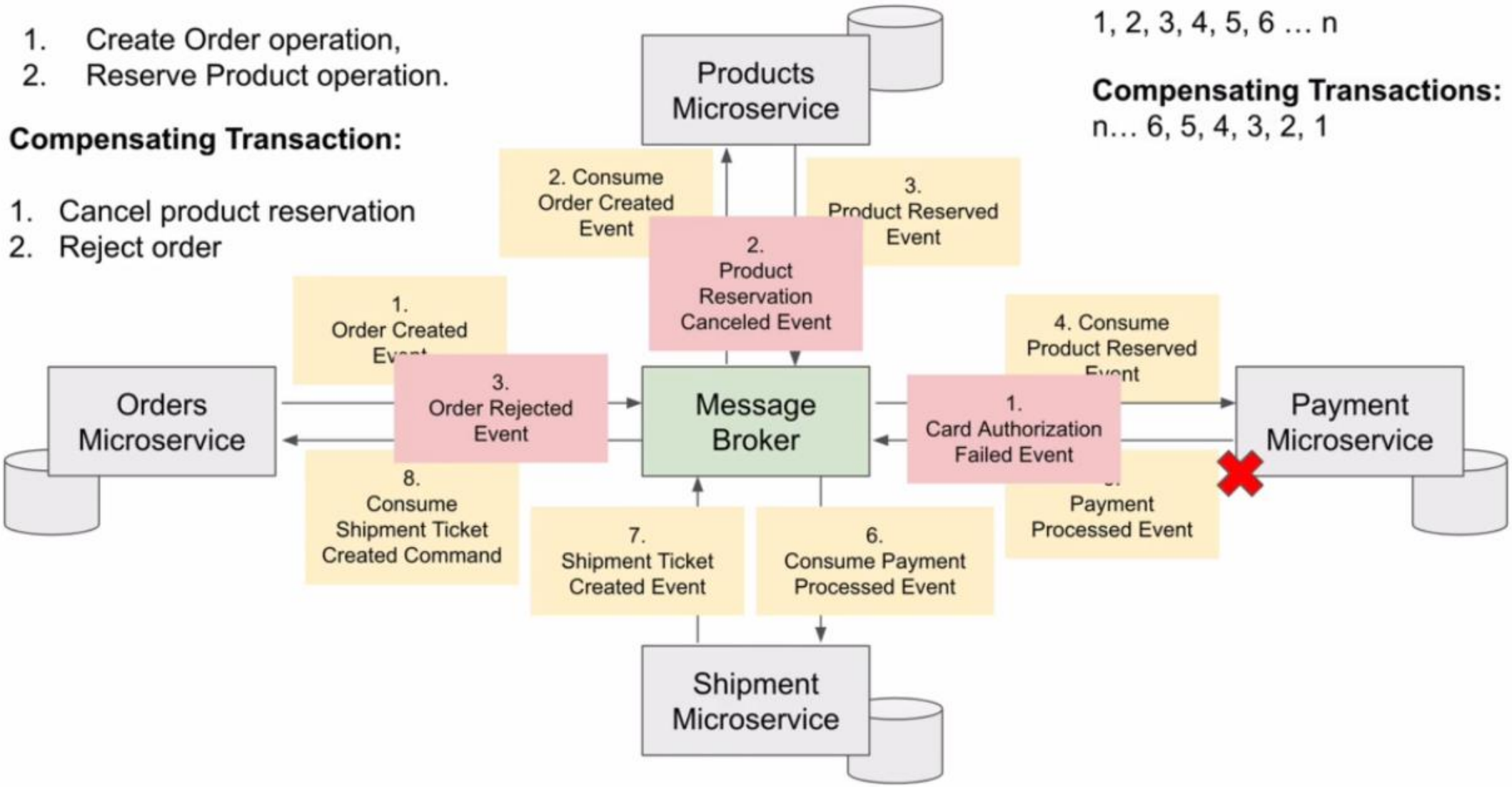
1. Create Order operation,
2. Reserve Product operation.

Compensating Transaction:

1. Cancel product reservation
2. Reject order

Initial Flow:
1, 2, 3, 4, 5, 6 … n

Compensating Transactions:
n… 6, 5, 4, 3, 2, 1



Choreography-Based SAGA

- In Choreography-Based SAGA, each microservice publishes the main event that triggers event in another microservice. You can think of these events as one transaction contains multiple local transactions

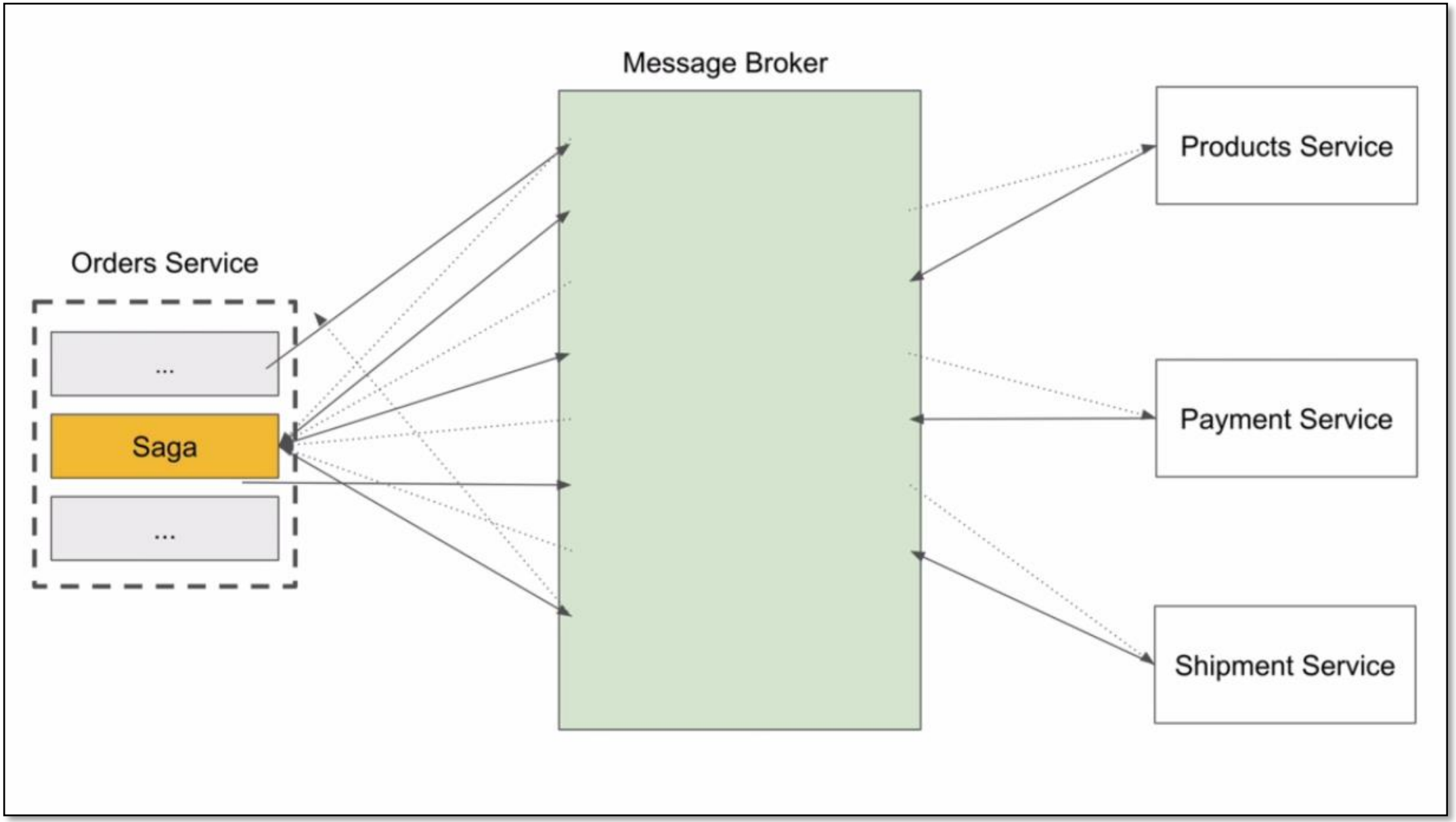
Day - 1

Orchestration-Based SAGA

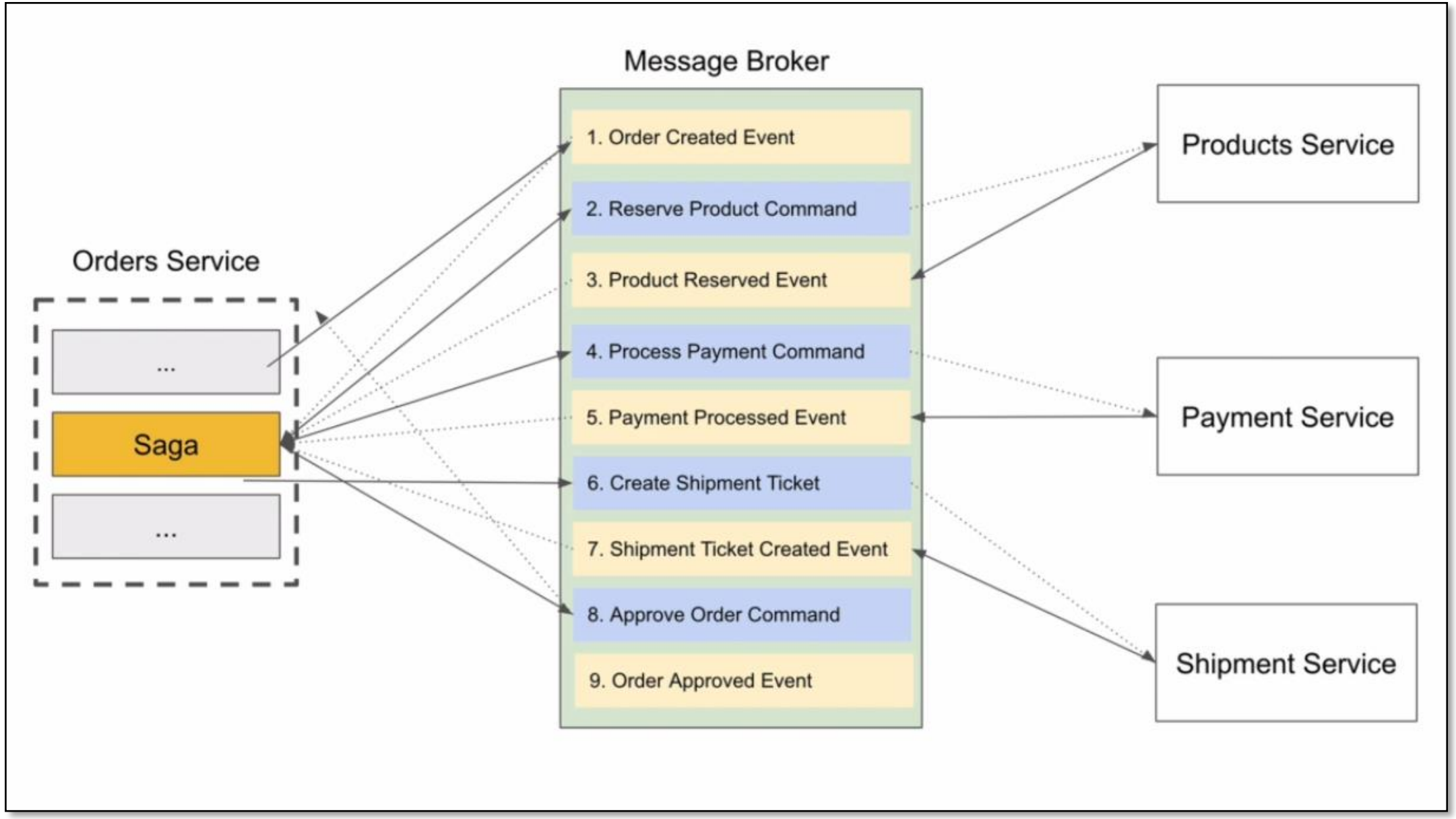
Orchestration-Based SAGA

- In Orchestration-Based SAGA, an orchestrator (object) tells the participants what local transactions to execute.

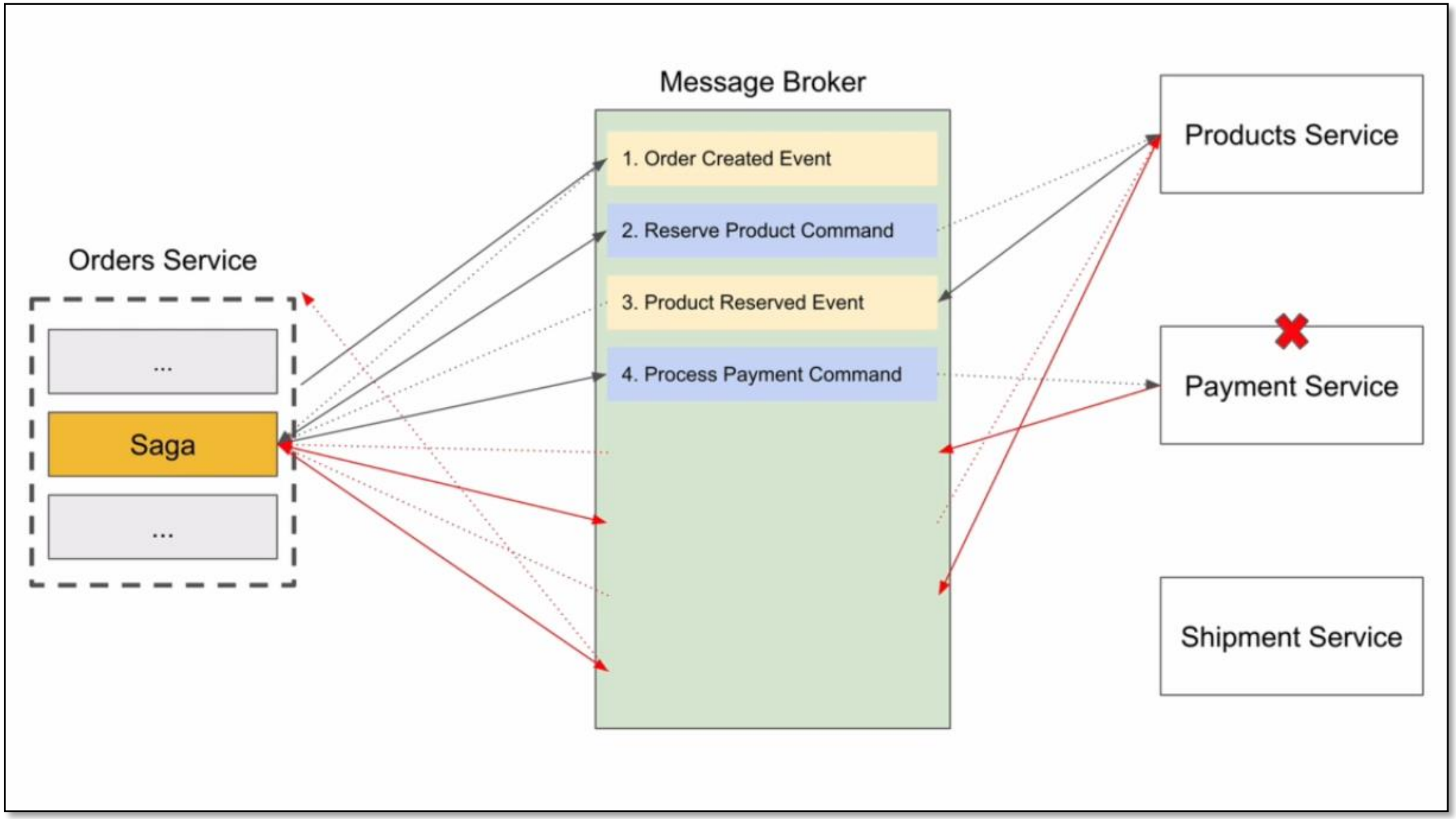
Orchestration-Based SAGA



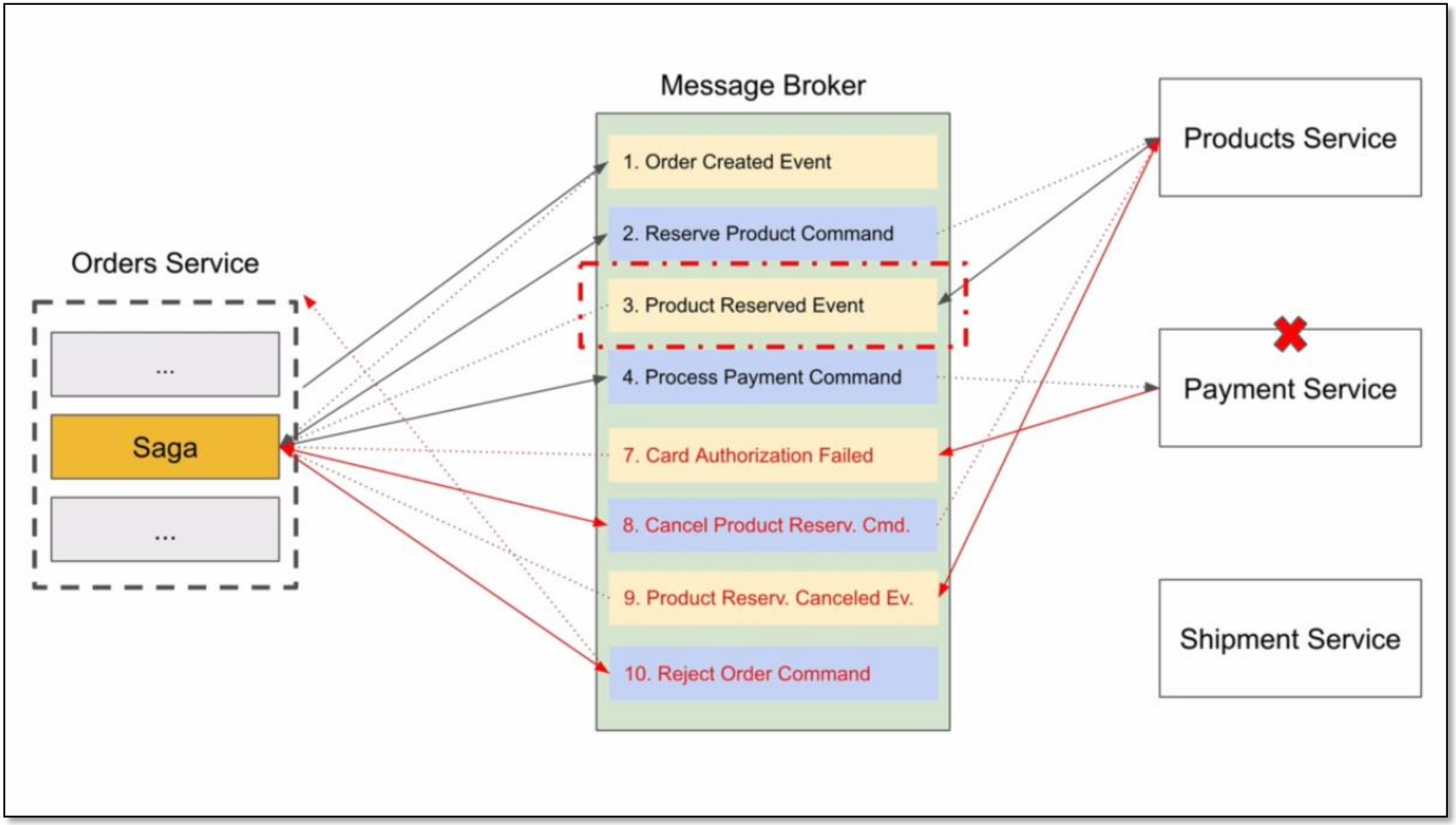
Orchestration-Based SAGA



Orchestration-Based SAGA



Orchestration-Based SAGA



Orchestration-Based SAGA

- SAGA will publish a Command, Microservices will consume this Command – process it and Publish an Event. SAGA will consume that Event and will decide what to do next. It can end SAGA or continue the process in the flow by publishing a New Command.

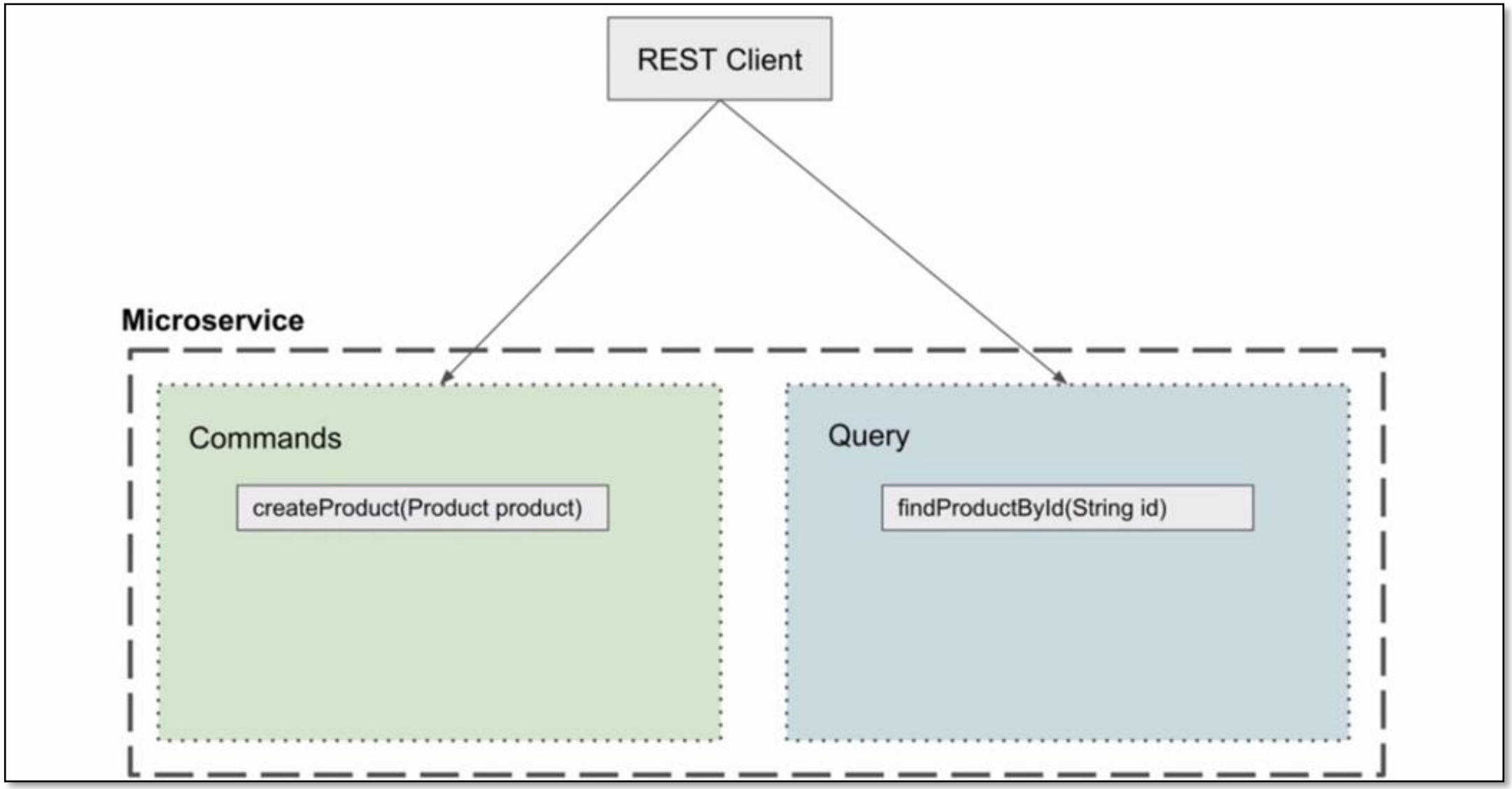
Day - 1

Command Query Responsibility Segregation

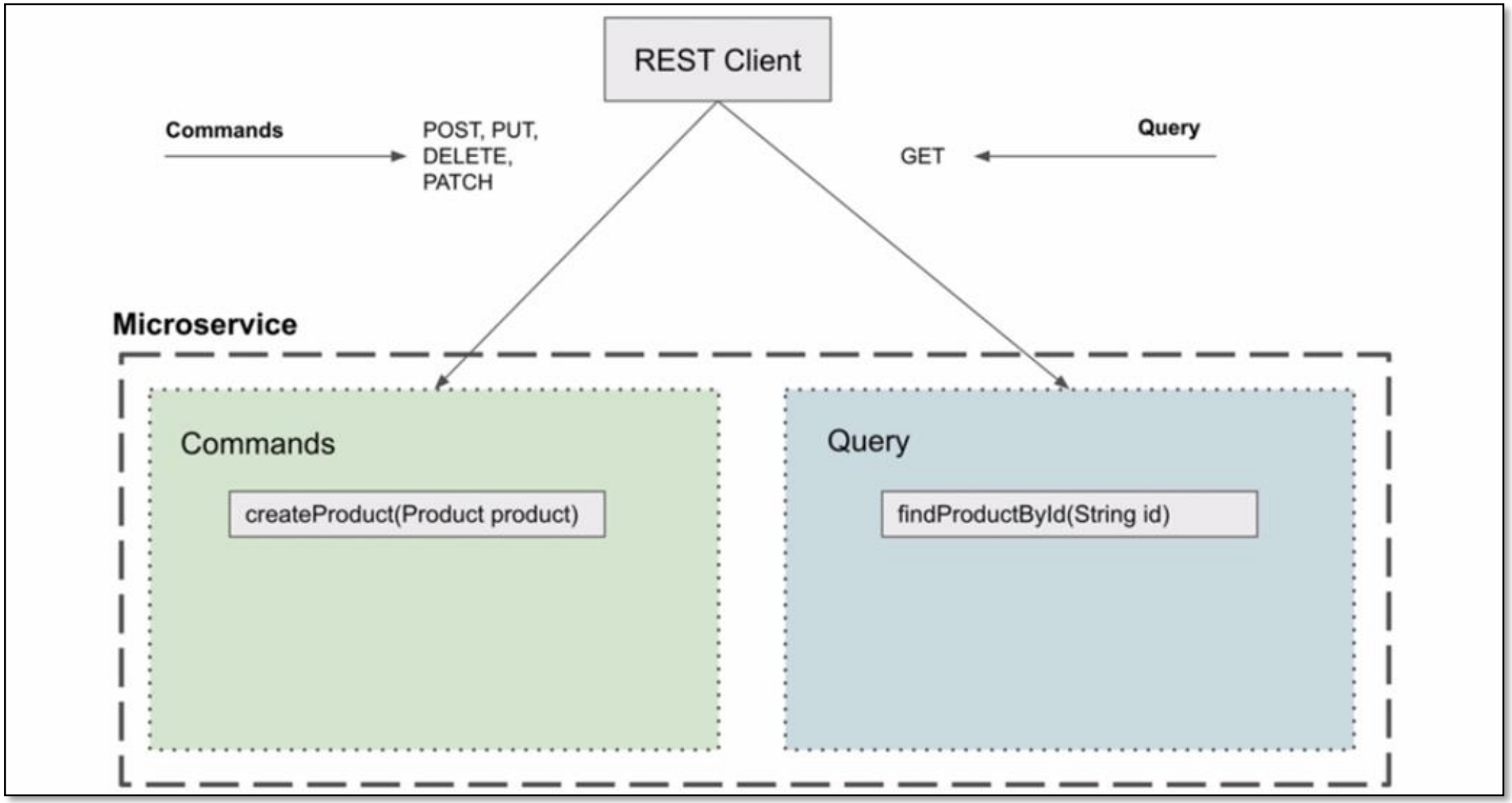
CQRS pattern

- The command query responsibility segregation (CQRS) pattern separates the data mutation, or the command part of a system, from the query part.
- You can use the CQRS pattern to separate updates and queries if they have different requirements for throughput, latency, or consistency.
- The CQRS pattern splits the application into two parts: the command side and the query side.
- The command side handles create, update, and delete requests.
- The query side runs the query part by using the read replicas.

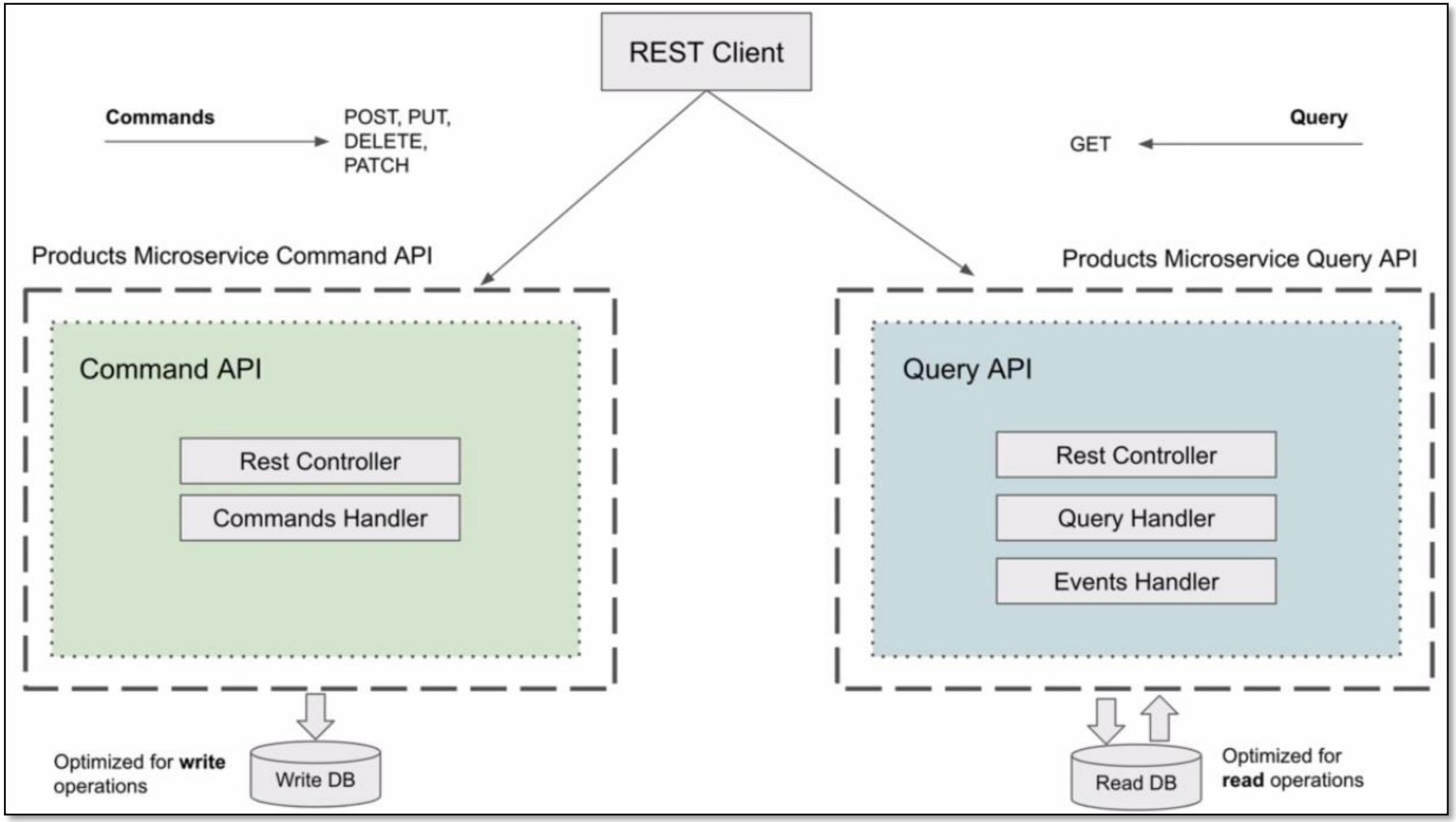
Command Query Responsibility Segregation



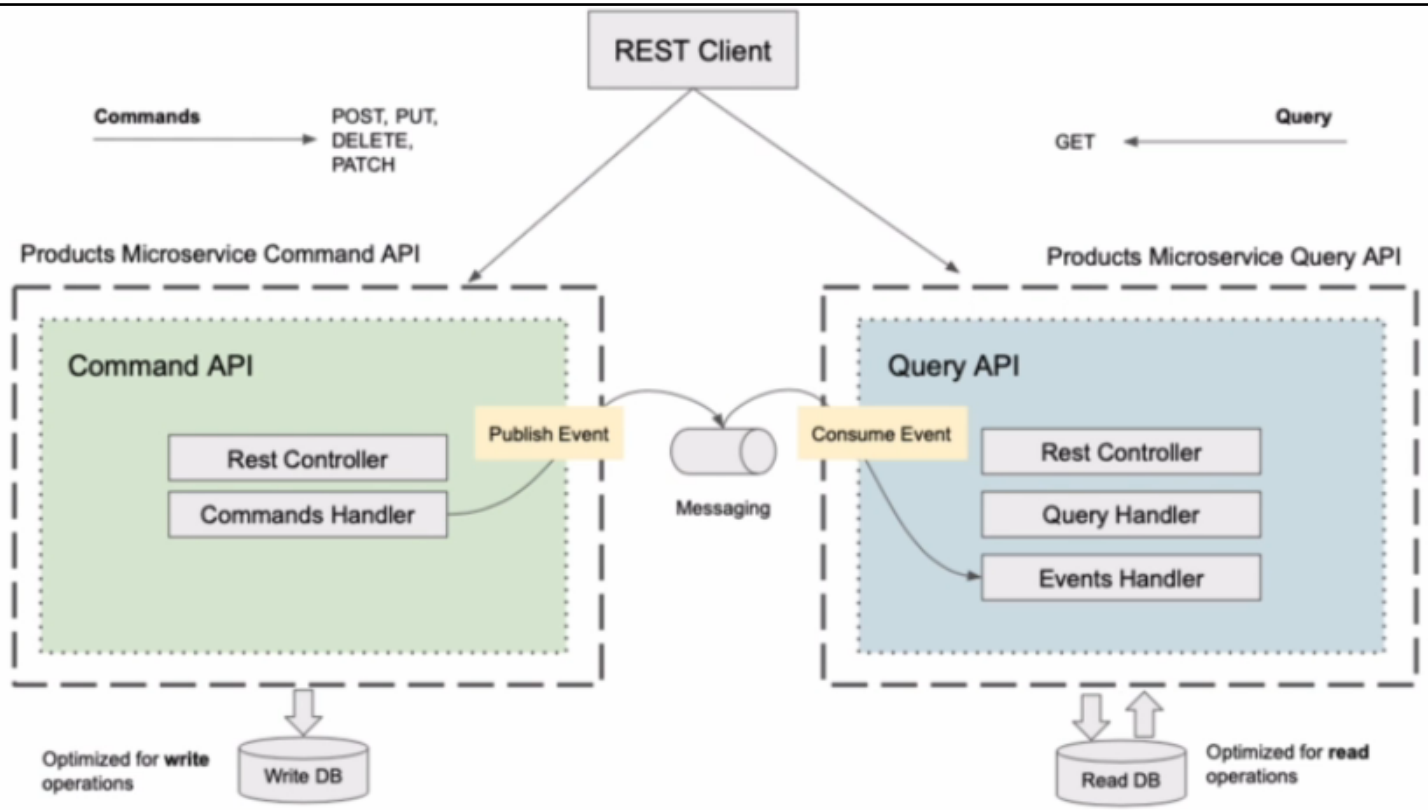
Command vs Query – Controllers



Read vs Write Traffic



Type of Messages



Type of Messages

- **Command** - express the intent to change the application's state.
  Ex. *CreateProductCommand, UpdateProductCommand, DeleteProductCommand*.
- **Query** - express the desire for information. Ex. *FindProductQuery, GetUserQuery*.
- **Event** - represent a notification that something relevant has happened. Ex. *ProductCreatedEvent, ProductUpdatedEvent*.

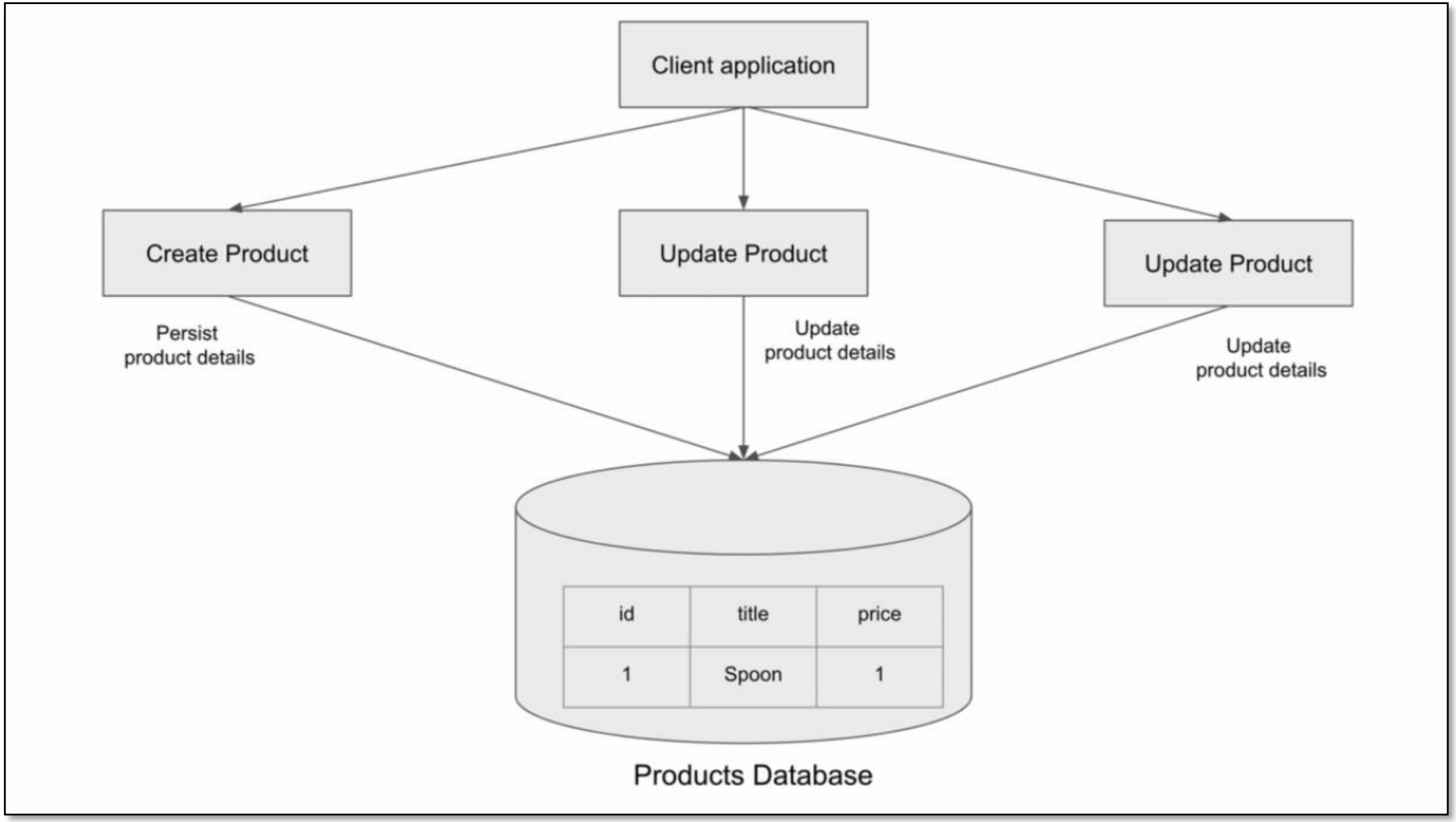
Day - 1

CQRS and Event Sourcing

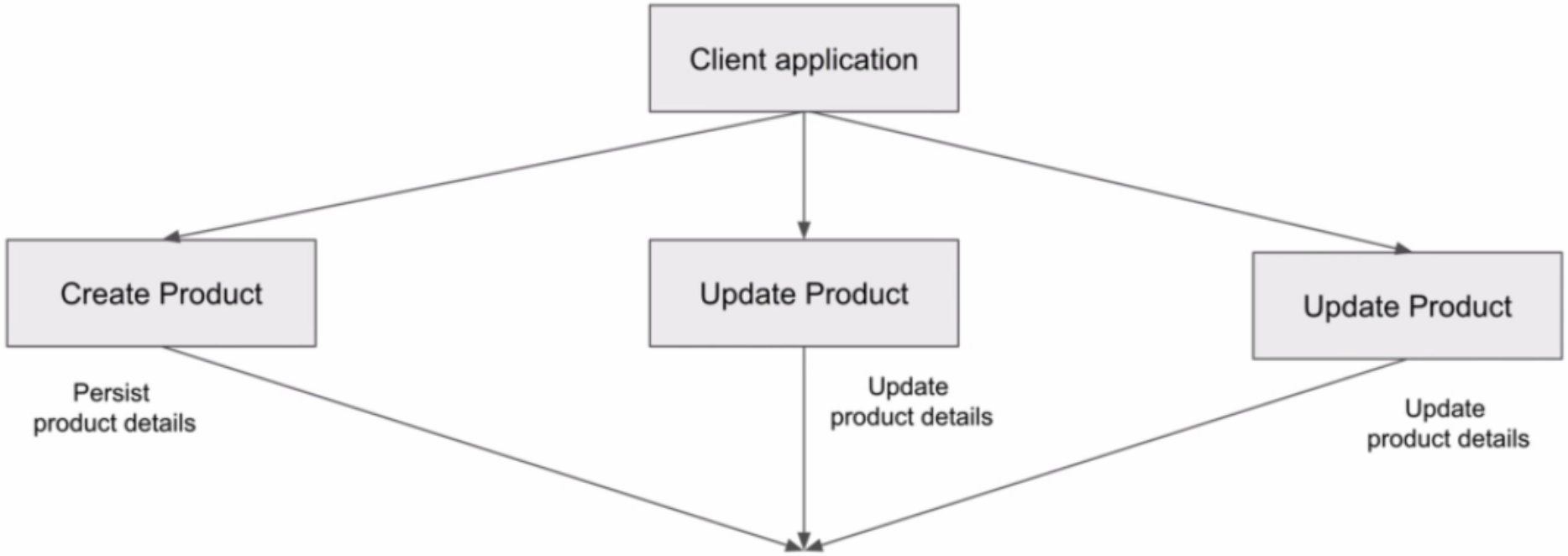
Event Sourcing

- In some scenarios though you would want more than the current state, you might need all the states which the customer entry went through.
- For such cases, the design pattern **“Event Sourcing”** helps.

Event Sourcing



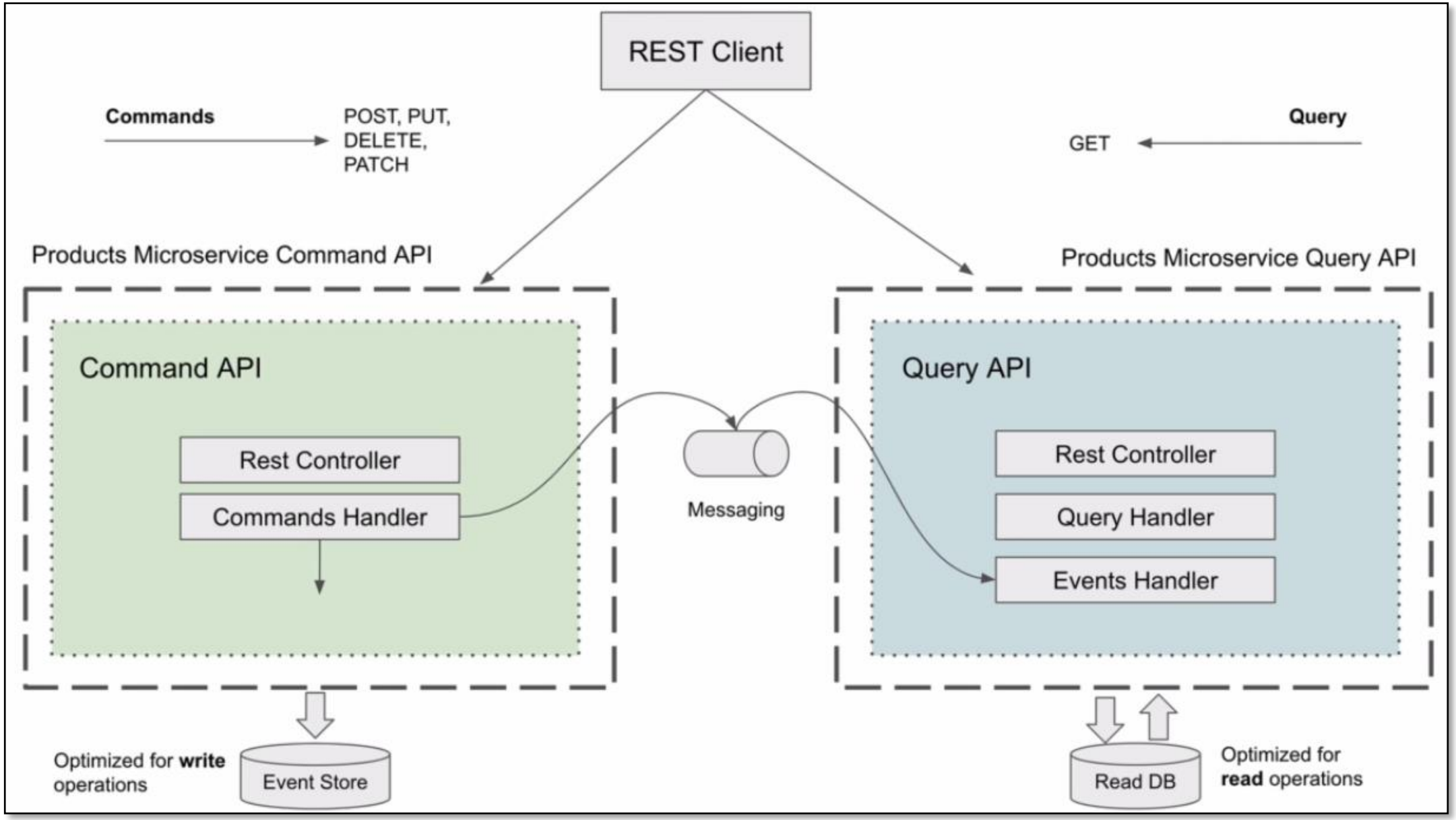
Event Sourcing



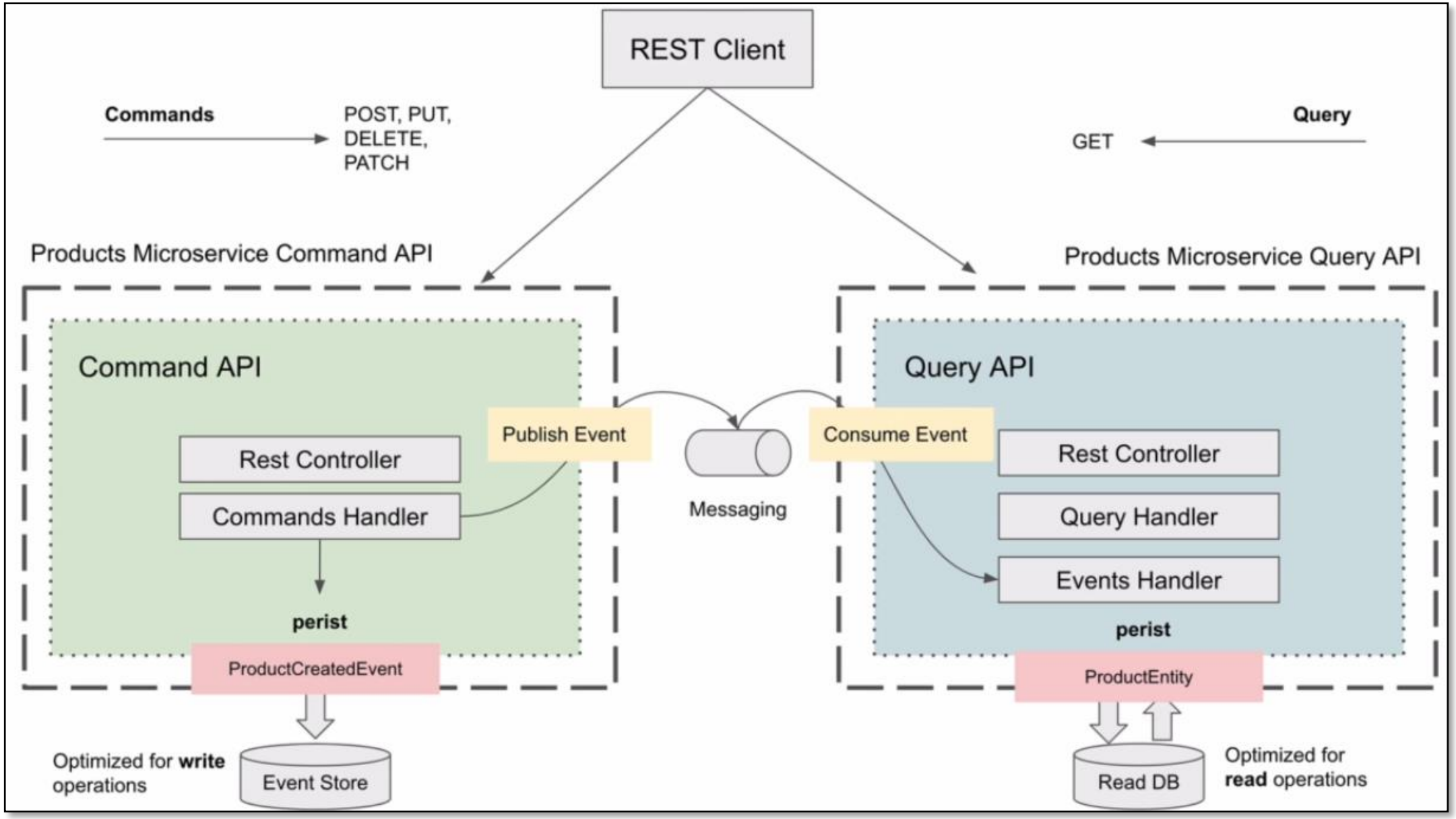
eventIdentifier	eventType	eventPayload
5e584f4e	ProductCreatedEvent	{"productId":"4ee43c56","title":"Spoon", "price":1}
60e74f5b	ProductUpdatedEvent	{"productId":"4ee43c56","title":"Spoon", "price":2}
d6f2d6c5	ProductUpdatedEvent	{"productId":"4ee43c56","title":"Spoon", "price":3}

Event Store

CQRS and Event Sourcing



CQRS and Event Sourcing



CQRS and Event Sourcing

Event Store

eventIdentifier	eventType	eventPayload
5e584f4e	ProductCreatedEvent	{"productId":"4ee43c56","title":"Spoon","price":1}
60e74f5b	ProductUpdatedEvent	{"productId":"4ee43c56","title":"Spoon","price":2}
d6f2d6c5	ProductUpdatedEvent	{"productId":"4ee43c56","title":"Spoon","price":3}
3e73f699	ProductPriceUpdatedEvent	{"productId":"4ee43c56", "price":3}

Read Database

id	title	price
1	Spoon	3

CQRS and Event Sourcing

Event Store

eventIdentifier	eventType	eventPayload
5e584f4e	ProductCreatedEvent	{"productId":"4ee43c56","title":"Spoon","price":1}
60e74f5b	ProductUpdatedEvent	{"productId":"4ee43c56","title":"Spoon","price":2}
d6f2d6c5	ProductUpdatedEvent	{"productId":"4ee43c56","title":"Spoon","price":3}
3e73f699	ProductPriceUpdatedEvent	{"productId":"4ee43c56", "price":3}

Replay

Read Database

id	title	price
1	Spoon	3

Read Database 2

title	price
Spoon	3

Day - 1

Building Microservices with Spring Boot

Building a RESTful web application

1. Create a new Project for **ProductService** using the Spring Initializr (start.spring.io).
2. Select the Spring Boot Version - 2.7.13.
3. Select the **Spring Web, Lombok, and Eureka Client** starter.
4. Create a ProductController will have the following Uri's:

URI	METHODS	Description
/products	POST	Return a String - "HTTP POST Method Handled"
/products	PUT	Return a String - "HTTP PUT Method Handled"
/products	GET	Return a String - "HTTP GET Method Handled"
/products	DELETE	Return a String - "HTTP DELETE Method Handled"

Configure Microservices with Eureka Service Registry Server

- The “ProductService" instance will exposes a remote API such as HTTP/REST at a particular location (host and port). To overcome the challenge of dynamically changing service instances and their locations. The code deployers intended to create a **Service Registry**, which is a database containing information about services, their instances, and their locations.

Configure Microservices with Eureka Service Registry Server

1. Create a new Project for **DiscoveryServer** using the Spring Initializr (start.spring.io).
2. Select the Spring Boot Version - 2.7.13.
3. Select the **Eureka Server** starter.
4. In the Application class, Add **@EnableEurekaServer** annotation.
5. Ensure the server is running on 8761.

application.properties

```
server.port=8761
eureka.client.register-with-eureka=false
eureka.client.fetch-registry=false
eureka.instance.prefer-ip-address=true
#eureka.instance.hostname=localhost
eureka.client.service-url.defaultZone=http://localhost:8761/eureka
```

6. Start the Application.
7. Verify the Eureka Server: http://localhost:8761/

Enable Dynamic Registration to Product Microservice

1. Refer the **ProductService** created previously.
2. Include the **application name** and **eureka.client.serviceUrl.defaultZone** in the application.properties files. For the Product Service application to dynamically register to Discovery Server.

application.properties

```
1
2 eureka.client.service-url.defaultZone=http://localhost:8761/eureka
3 spring.application.name=products-service
4
```

3. In the Application class, Add **@EnableEurekaClient/@EnableDiscoveryClient** annotation.
4. Ensure that the Discovery Server and Product Service is running.
5. Again, verify the Eureka Server: http://localhost:8761/

Implementing Spring Cloud Gateway in Microservices

1. Ensure the Discovery Server and Product Service is running.
2. Now let's implement an API gateway that acts as a single-entry point for a collection of microservices.
3. Create a new Project for **ApiGateway** using the Spring Initalizr (start.spring.io).
4. Select the Spring Boot Version - 2.7.13.
5. Select the **Gateway, Spring Web and Eureka Discovery Client** starter.
6. Add **@EnableEurekaClient/@EnableDiscoveryClient** in the Application class.
7. In application.properties file, enable the automatic mapping of gateway routes and add the application name and eureka client serviceUrl.

application.properties

```
1
2 spring.application.name=api-gateway
3 server.port=8082
4 eureka.client.service-url.defaultZone=http://localhost:8761/eureka
5
6 spring.cloud.gateway.discovery.locator.enabled=true
7 spring.cloud.gateway.discovery.locator.lower-case-service-id=true
8
```

Implementing Spring Cloud Gateway in Microservices

8. Start the ApiGateway.
9. Check the proxy running instances is also registered with the Eureka Server.
10. Test the Proxy: http://localhost:8082/products-service/products

Recap of Day – 1

- Microservice vs Monolithic application
- Event-Driven Microservices
- Transactions in Microservices
- Choreography-Based Saga
- Orchestration-Based Saga
- Command Query Responsibility Segregation
- Types of Messaging in CQRS Pattern
- CQRS and Event Sourcing
- Building microservices with spring boot

Day - 2

Day – 2 Agenda

- Introduction to Axon Server
- Download and run Axon Server as JAR application
- Axon Server configuration properties
- Run Axon Server in a Docker container
- Bringing CQRS and Event Sourcing Together with Axon Framework
- Accept HTTP Request Body
- Adding Axon Framework Spring Boot Starter
- Creating a new Command class
- Send Command to a Command Gateway

Day – 2 Agenda

- Introduction to Aggregate
- Creating the Aggregate class
- Validate the command class
- Creating the event class
- Apply and Publish the Created Event
- @EventSourcingHandler Annotation
- Previewing Event in the EventStore

Day - 2

Axon – Getting Started

Axon

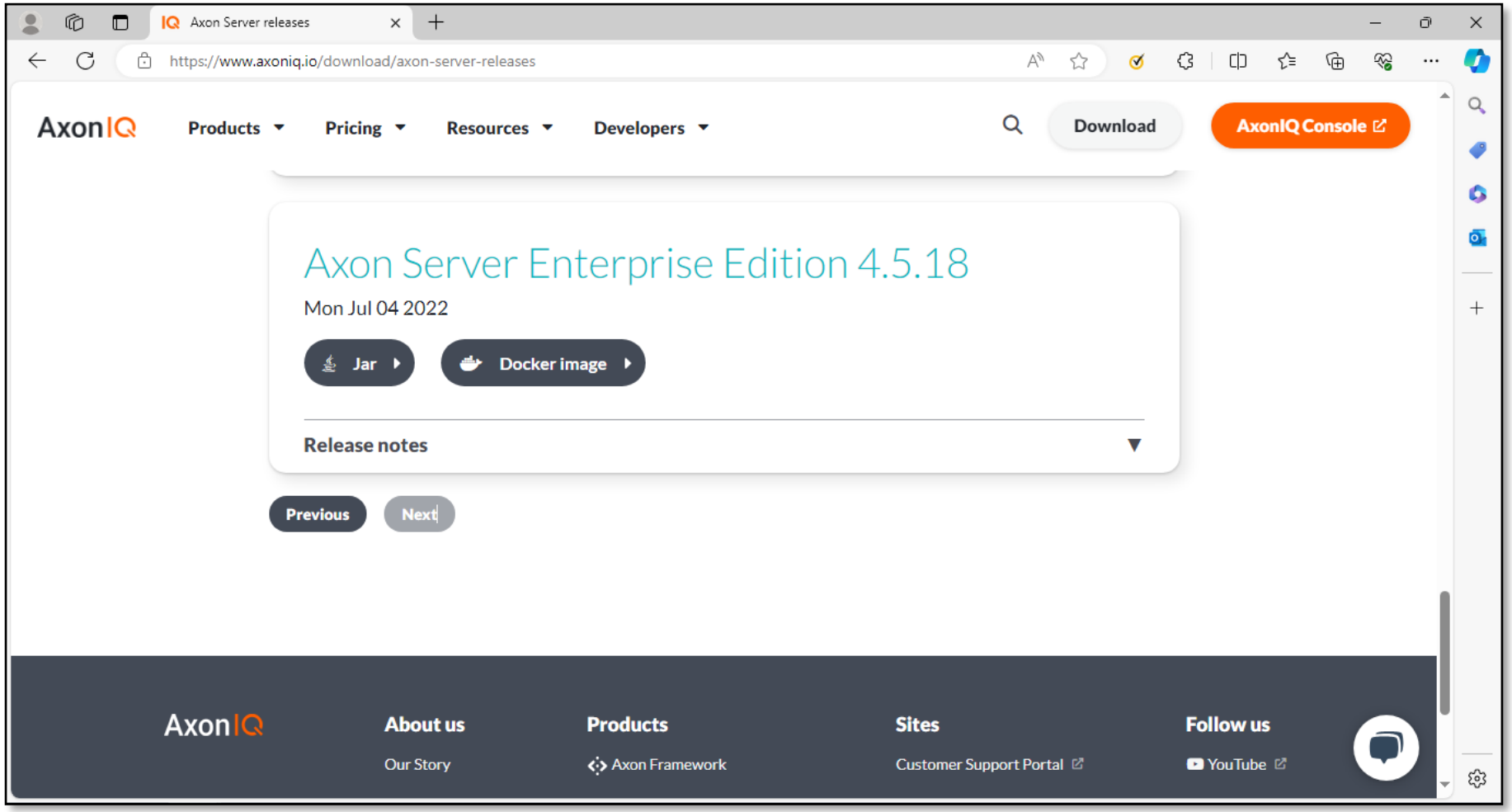
- Axon provides the **Axon Framework** and the **Axon Server** to help build applications centered on three core concepts - **CQRS** (Command Query Responsibility Segregation) / **Event Sourcing** and **DDD** (Domain Driven Design).
- While many types of applications can be built using Axon, it has proven to be very popular for microservices architectures. Axon provides an innovative and powerful way of sensibly evolving to event-driven microservices within a microservices architecture.

Day - 2

Axon Server

Download and run Axon Server as JAR application

- Go to https://developer.axoniq.io/download
- Select Axon Server Enterprise Edition 4.5.18, download the JAR and execute.

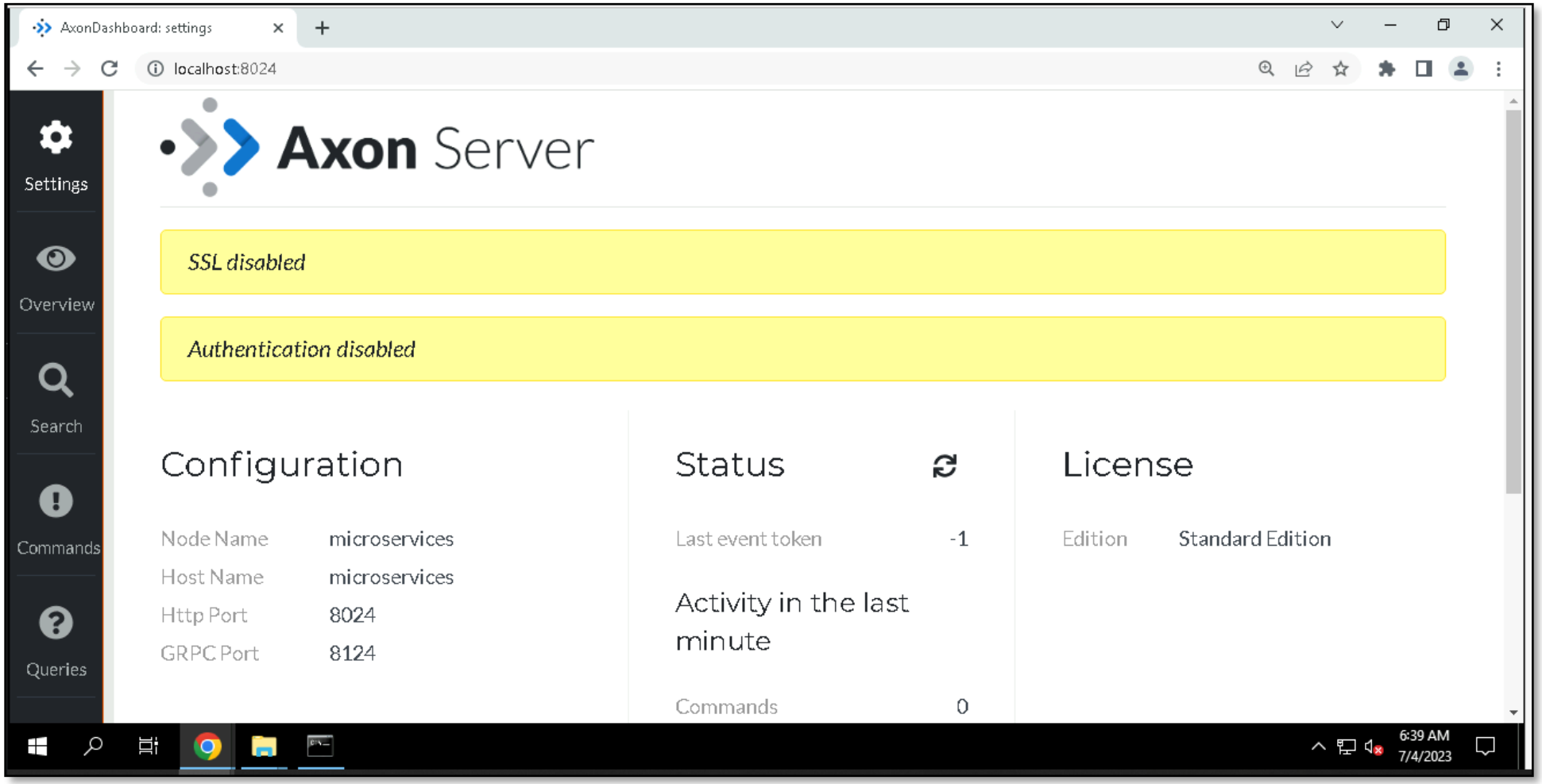


Download and run Axon Server as JAR application

- Run java -jar axonserver.jar

Download and run Axon Server as JAR application

- Access http://localhost:8024



Day - 2

Run Axon Server in a Docker Container

Run Axon Server in a Docker Container

- Docker Command:

docker run -d --name axonserver -p 8024:8024 -p 8124:8124 axoniq/axonserver:4.5.8

Command Prompt

```
C:\Users\labuser>docker run -d --name axonserver -p 8024:8024 -p 8124:8124 axoniq/axonserver:4.5.8
Unable to find image 'axoniq/axonserver:4.5.8' locally
4.5.8: Pulling from axoniq/axonserver
ec52731e9273: Pull complete
8907fc4ab049: Pull complete
a1f1879bb7de: Pull complete
5347aaf66df0: Pull complete
2bdcc0330791: Pull complete
b1bb79b3cfdb: Pull complete
191655b822f9: Pull complete
11e10d67a998: Pull complete
d3d5ea2e5b82: Pull complete
07ff33aa477f: Pull complete
338c3c26206a: Pull complete
Digest: sha256:7b004facd1da1fa791e4a2752b6cdec68ae7aeb2cd216a2c4dd95b26beeb6ab1
Status: Downloaded newer image for axoniq/axonserver:4.5.8
a2006525b087bd82d16e57e92b6ed5f786f93364b20638006e992b64753c52b5

C:\Users\labuser>docker ps
CONTAINER ID   IMAGE                    COMMAND                  CREATED              STATUS              PORTS
                                                                                 NAMES
a2006525b087   axoniq/axonserver:4.5.8   "java -cp /app/resou…"   About a minute ago   Up About a minute   0.0.0.0:8024-
>8024/tcp, :::8024->8024/tcp, 0.0.0.0:8124->8124/tcp, :::8124->8124/tcp   axonserver

C:\Users\labuser>_
```

Start, Stop, Delete Axon Server Docker Container By ID

- docker ps -a
- docker start <container-id>
- docker stop <container-id>
- docker rm <container-id>

Day - 2

Bringing CQRS and Event Sourcing Together with Axon Framework

Introduction

- Apply the CQRS design pattern to our Products Microservices.

Accept HTTP Request Body

- Add the Lombok dependency:

```
<dependency>
    <groupId>org.projectlombok</groupId>
    <artifactId>lombok</artifactId>
    <scope>provided</scope>
</dependency>
```

Accept HTTP Request Body

- Create a **CreateProductRestModel** class for Accept HTTP Request Body:

CreateProductRestModel.java

```java
package com.mphasis.controller;

import java.math.BigDecimal;

import lombok.Data;

@Data
public class CreateProductRestModel {

    private String title;
    private BigDecimal price;
    private Integer quantity;
}
```

Accept HTTP Request Body

- Apply **CreateProductRestModel** to the Controller:

ProductController.java

```java
package com.mphasis.controller;

import org.springframework.beans.factory.annotation.Autowired;

@RestController
@RequestMapping("/products")
public class ProductController {

    @Autowired
    private Environment env;

    @PostMapping
    public String createProduct(@RequestBody CreateProductRestModel createProductRestModel) {
        return " HTTP POST Handled " + createProductRestModel.getTitle();
    }

    @GetMapping
    public String getProduct() {
        return " HTTP GET Handled: " + env.getProperty("local.server.port");
    }
```

Adding Axon Framework Spring Boot Starter

- We will add **axon-spring-boot-starter** starter to ProductService/pom.xml:

```
<dependency>
    <groupId>org.axonframework</groupId>
    <artifactId>axon-spring-boot-starter</artifactId>
    <version>4.5.8</version>
</dependency>
```

Day - 2

Product Service API - Commands

Creating a new Command class

- **Command** - express the intent to change the application's state.
- In terms of CQRS design pattern, when there’s a modifying request, this is a Command. Because the Command is intended to make a change, but in this case Command in creating a Product.
- The name of Command should be in below format:

<Verb><Noun>Command

CreateProductCommand

UpdateProductCommand

Target Aggregate Identifier

TargetAggregateIdentifier

The TargetAggregateIdentifier annotation tells Axon that the annotated field is an id of a given aggregate to which the command should be targeted.

Creating a new Command class

- The CreateProductCommand will be read-only:

CreateProductCommand.java

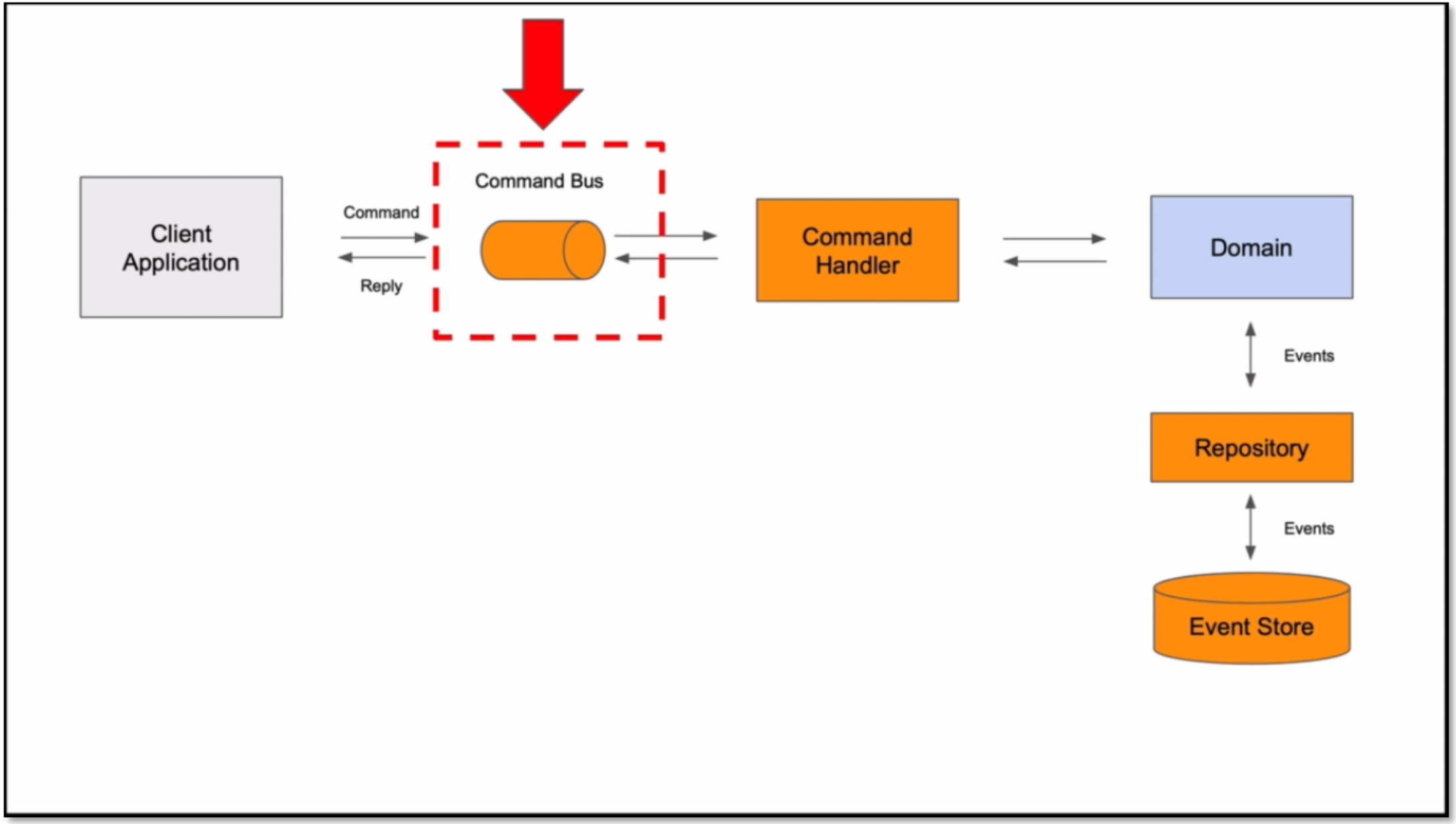
```java
package com.mphasis.command;

import java.math.BigDecimal;

import org.axonframework.modelling.command.TargetAggregateIdentifier;

import lombok.Builder;
import lombok.Data;

@Builder
@Data
public class CreateProductCommand {

    @TargetAggregateIdentifier
    private final String productId;
    private final String title;
    private final BigDecimal price;
    private final Integer quantity;

}
```

Send Command to a Command Gateway

CommandGateway:

Interface towards the Command Handling components of an application. This interface provides a friendlier API toward the command bus. The CommandGateway allows for components dispatching commands to wait for the result.

Send Command to a Command Gateway



Send Command to a Command Gateway

ProductController.java

```java
    @Autowired
    private CommandGateway commandGateway;

    @PostMapping
    public String createProduct(@Valid @RequestBody CreateProductRestModel createProductRestModel) {

        CreateProductCommand createProductCommand = CreateProductCommand.builder()
        .price(createProductRestModel.getPrice())
        .quantity(createProductRestModel.getQuantity())
        .title(createProductRestModel.getTitle())
        .productId(UUID.randomUUID().toString())
        .build();

        String returnValue;

        try {
            returnValue = commandGateway.sendAndWait(createProductCommand);
        }catch (Exception ex) {
            returnValue = ex.getLocalizedMessage();
        }
        return returnValue;
    }
```

Day - 2

Product Service API - Events

Creating a new Event class

- Event - represent a notification that something relevant has happened.
- To publish an Event first we must create Event class.
- Naming conventions for Event:

<Noun><PerformedAction>Event

ProductCreatedEvent

ProductShippedEvent

ProductDeletedEvent

Creating a new Event class

- Our aggregate will handle the commands, as it's in charge of deciding if a Product can be created, deleted, or shipped.

- It will notify the rest of the application of its decision by publishing an event. We'll have three types of events — ProductCreatedEvent, ProductDeletedEvent, and ProductShippedEvent.

Creating a new Event class

- Let’s create the ProductCreatedEvent class:

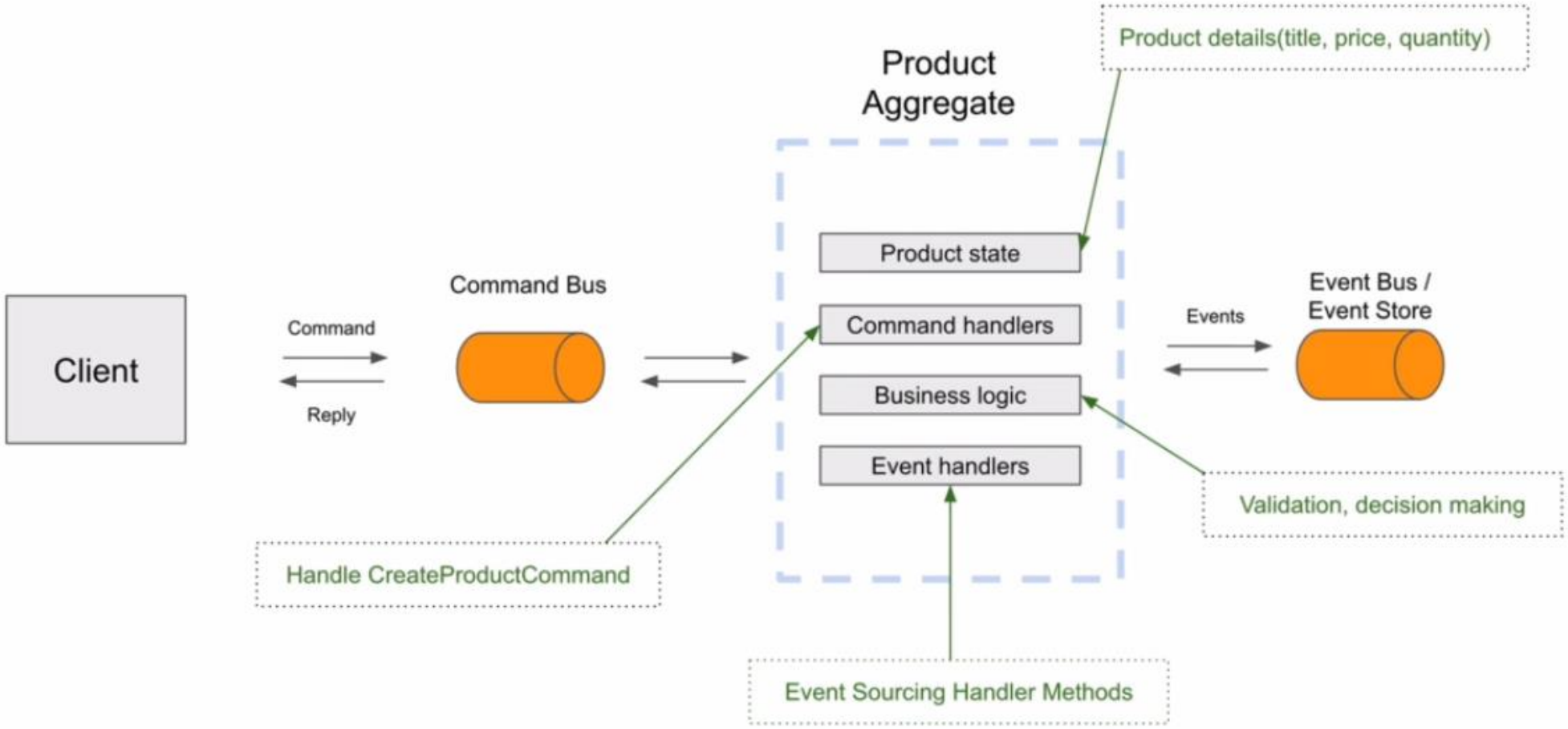
ProductCreatedEvent.java

```java
package com.mphasis.events;

import java.math.BigDecimal;

import lombok.Data;

@Data
public class ProductCreatedEvent {

    private String productId;
    private String title;
    private BigDecimal price;
    private Integer quantity;
}
```

Day - 2

The Command Model – Product Aggregate

Product Aggregate - Introduction



Product Aggregate - Introduction

- Aggregate class is at the core of your microservice.
- It holds the current state of the main object.
- In this section we are working on Product Microservice, and this means our aggregate class will be called **ProductAggregate**.
- This Product Aggregate object will hold the current state of Product object.
- It will hold the current value of the Product details(title, price, quantity).
- Additionally, the Product Aggregate will contain methods that can handle commands.
- The Product Aggregate class will also have the business logic.
- The Product Aggregate class will contain the Event Sourcing Handler Methods.
- Every time the state of the Product changes an Event Sourcing Handler Method will be invoked.

Aggregate

Aggregate

The Aggregate annotation is an Axon Spring specific annotation marking this class as an aggregate. It will notify the framework that the required **CQRS** and **Event Sourcing** specific building blocks need to be instantiated for this *ProductAggregate*.

CommandHandler

Marker annotation to mark any method on an object as being a CommandHandler. Use the AnnotationCommandHandlerAdapter to subscribe the annotated class to the command bus. This annotation can also be placed directly on Aggregate members to have it handle the commands directly.

Create ProductAggregate class

- Product Aggregate class is annotated with **@Aggregate** annotation.
- Second constructor with CreateProductCommand argument is annotated with **@CommandHandler** annotation.

ProductAggregate.java

```java
package com.mphasis.command;

import org.axonframework.commandhandling.CommandHandler;
import org.axonframework.spring.stereotype.Aggregate;

@Aggregate
public class ProductAggregate {

    public ProductAggregate() {

    }

    @CommandHandler
    public ProductAggregate(CreateProductCommand createProductCommand) {
        // Validate Create Product Command
    }
}
```

Validate the CreateProductCommand

- Product Aggregate class can be used to validate the CreateProductCommand.

ProductAggregate.java

```java
@Aggregate
public class ProductAggregate {

    public ProductAggregate() {

    }

    @CommandHandler
    public ProductAggregate(CreateProductCommand createProductCommand) {
        // Validate Create Product Command

        if (createProductCommand.getPrice().compareTo(BigDecimal.ZERO) <= 0) {
            throw new IllegalArgumentException("Price cannot be less or equal than zero");
        }

        if(createProductCommand.getTitle() == null
                || createProductCommand.getTitle().isEmpty()) {
            throw new IllegalArgumentException("Title cannot be empty");
        }
    }
}
```

AggregateLifeCycle.apply(Object payload)

AggregateLifeCycle.apply(Object payload)

Apply a **DomainEventMessage** with given payload without metadata. Applying events means they are immediately applied (published) to the aggregate and scheduled for publication to other event handlers.

EventHandler

EventHandler

Annotation to be placed on methods that can handle events. The parameters of the annotated method are resolved using parameter resolvers.

Apply and Publish the Product Created Event

ProductAggregate.java

```java

    @CommandHandler
    public ProductAggregate(CreateProductCommand createProductCommand) {
        // Validate Create Product Command

        if (createProductCommand.getPrice().compareTo(BigDecimal.ZERO) <= 0) {
            throw new IllegalArgumentException("Price cannot be less or equal than zero");
        }

        if(createProductCommand.getTitle() == null
                || createProductCommand.getTitle().isEmpty()) {
            throw new IllegalArgumentException("Title cannot be empty");
        }

        ProductCreatedEvent productCreatedEvent = new ProductCreatedEvent();

        BeanUtils.copyProperties(createProductCommand, productCreatedEvent);

        AggregateLifecycle.apply(productCreatedEvent);
    }
}
```

@EventSourcingHandler

- **AggregateLifecycle.apply(Object payload)** – Apply an DomainEventMessage with given payload without metadata. Applying events means they are immediately applied (published) to the aggregate and scheduled for publication to other event handlers.
- **@TargetAggregateIdentifier** in CreateProductCommand and **@AggregateIdentifier** in ProductAggregate will help Axon Framework to associate the dispatch command with the right aggregate.

AggregateIdentifier

AggregateIdentifier

Field annotation that identifies the field containing the identifier of the Aggregate.

@EventSourcingHandler

ProductAggregate.java

```java
package com.mphasis.command;

import java.math.BigDecimal;

@Aggregate
public class ProductAggregate {

    @AggregateIdentifier
    private String productId;
    private String title;
    private BigDecimal price;
    private Integer quantity;

    public ProductAggregate() {

    }

    @CommandHandler
    public ProductAggregate(CreateProductCommand createProductCommand) {
        // Validate Create Product Command
```

@EventSourcingHandler

ProductAggregate.java

```java

        if (createProductCommand.getTitle() == null || createProductCommand.getTitle().isEmpty()) {
            throw new IllegalArgumentException("Title cannot be empty");
        }

        ProductCreatedEvent productCreatedEvent = new ProductCreatedEvent();

        BeanUtils.copyProperties(createProductCommand, productCreatedEvent);

        AggregateLifecycle.apply(productCreatedEvent);
    }

    @EventSourcingHandler
    public void on(ProductCreatedEvent productCreatedEvent) {
        this.productId = productCreatedEvent.getProductId();
        this.price = productCreatedEvent.getPrice();
        this.title = productCreatedEvent.getTitle();
        this.quantity = productCreatedEvent.getQuantity();
    }
}

```

Adding Additional Dependency

- When using Axon with Spring Cloud – Maven start demanding for the Google Guava dependency (due to transitive dependency).

```
<dependency>
    <groupId>org.axonframework</groupId>
    <artifactId>axon-spring-boot-starter</artifactId>
    <version>4.5.8</version>
</dependency>

<dependency>
    <groupId>com.google.guava</groupId>
    <artifactId>guava</artifactId>
    <version>30.1-jre</version>
</dependency>
```

Trying how it works

1. Run the AxonServer using Docker command.
2. Ensure the Discovery Server and Product Service is running.
3. Ensure the ApiGateway is running.

Send a POST request

Home Workspaces Explore Search Postman Sign In Create Account

POST http://localhost:8082/p

http://localhost:8082/products-service/products Add to collection

POST http://localhost:8082/products-service/products Send

Params Authorization Headers (8) Body Pre-request Script Tests Settings Cookies

none form-data x-www-form-urlencoded raw binary JSON Beautify

```
1 {
2     "title":"iPhone 3",
3     "price":300,
4     "quantity":2
5 }
6
```

Body Cookies Headers (3) Test Results Status: 200 OK Time: 1 m 18.78 s Size: 152 B Save Response

Pretty Raw Preview Visualize Text

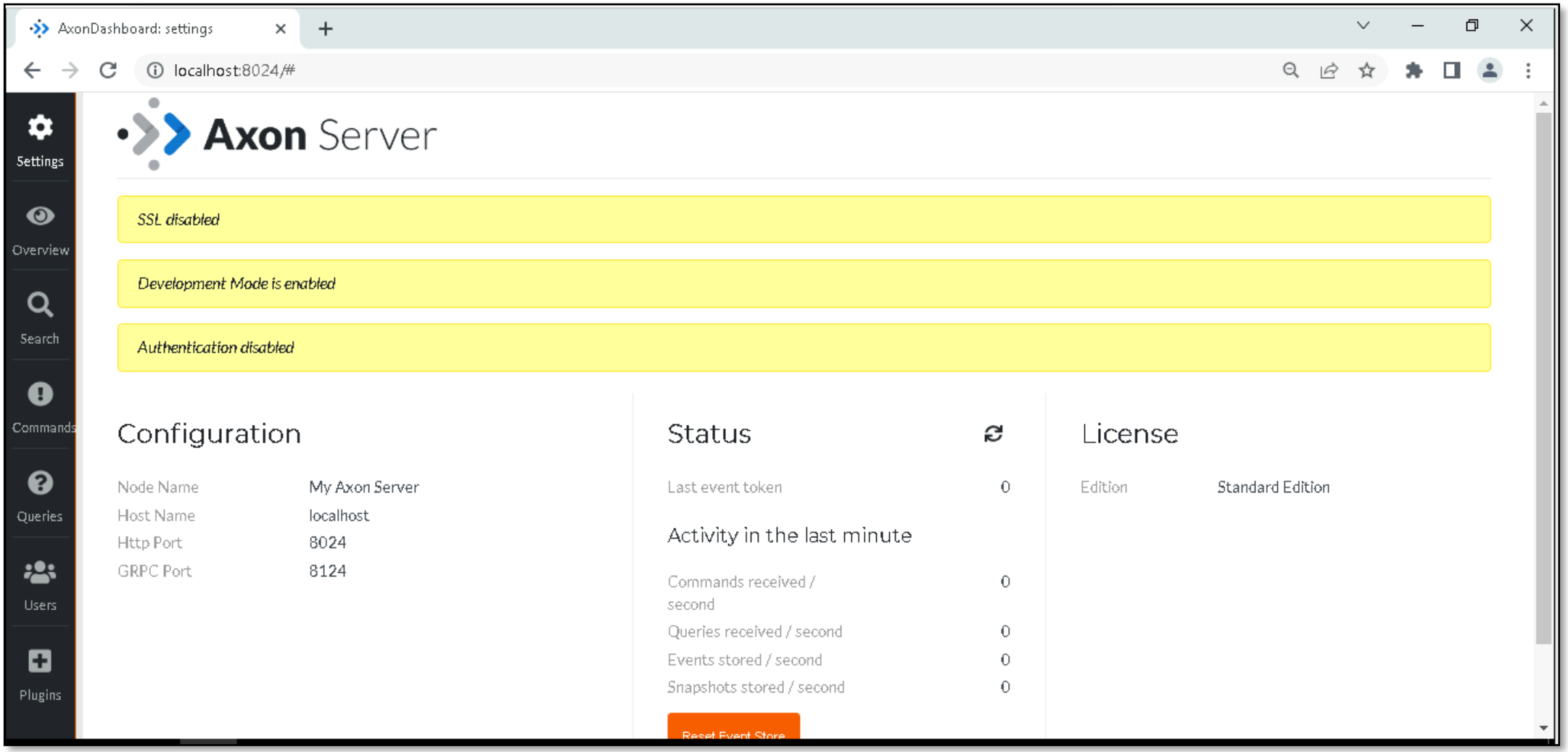
```
1 51a8bedc-b3ca-436c-b44e-64b70d0750d3
```

Console

6:49 PM 7/4/2023

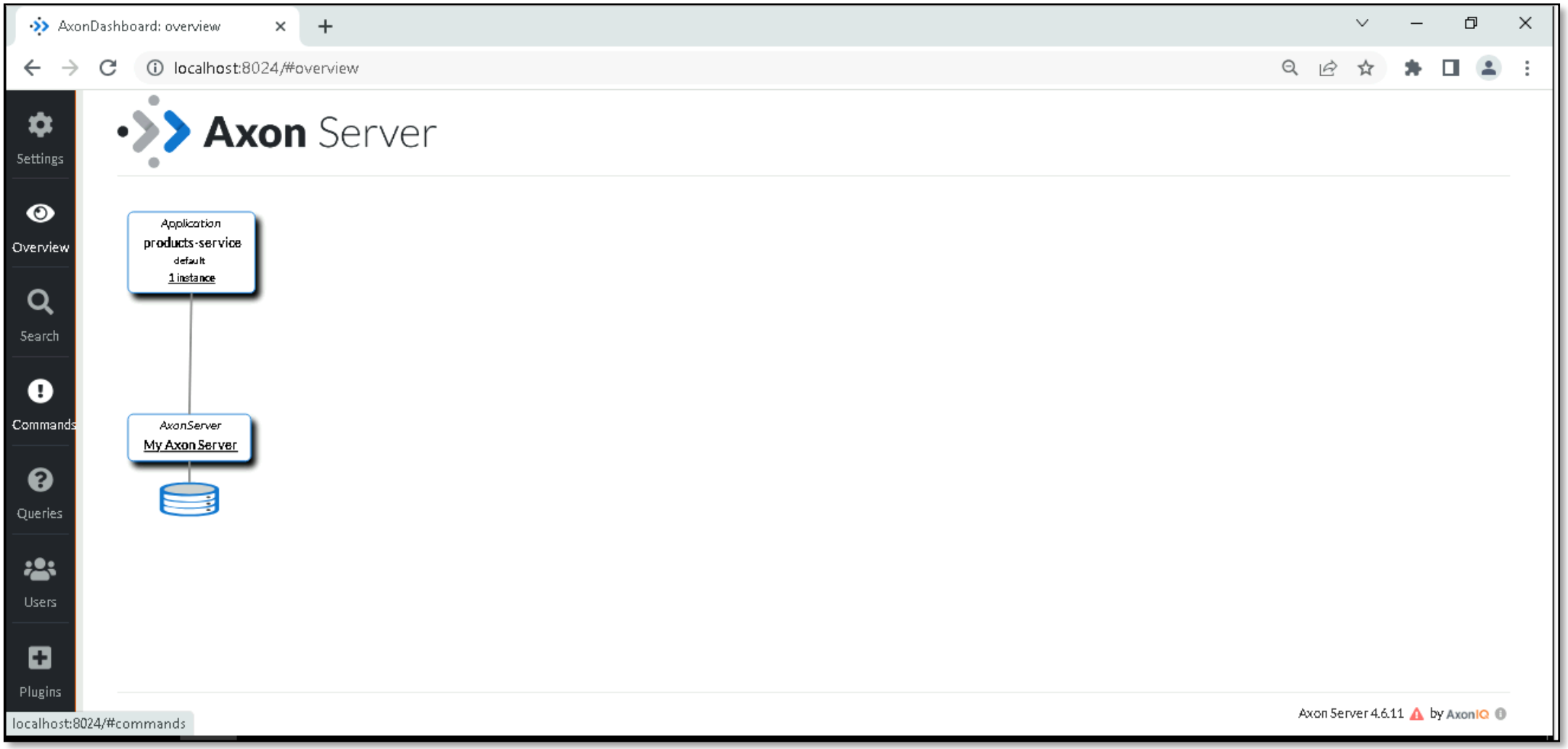
Previewing Event in the EventStore

- Go to Axon Server:



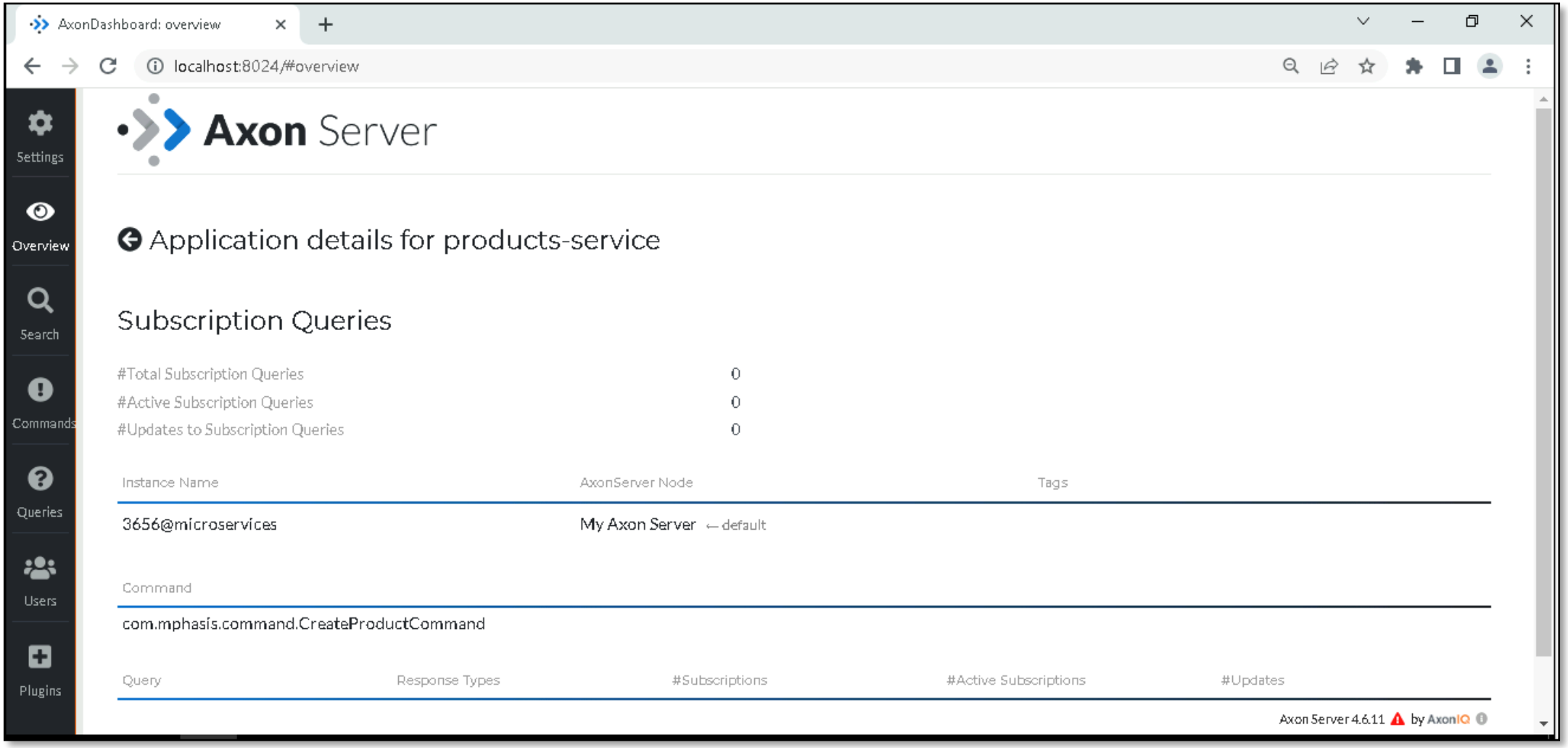
Previewing Event in the EventStore

- Click on **Overview** tab:



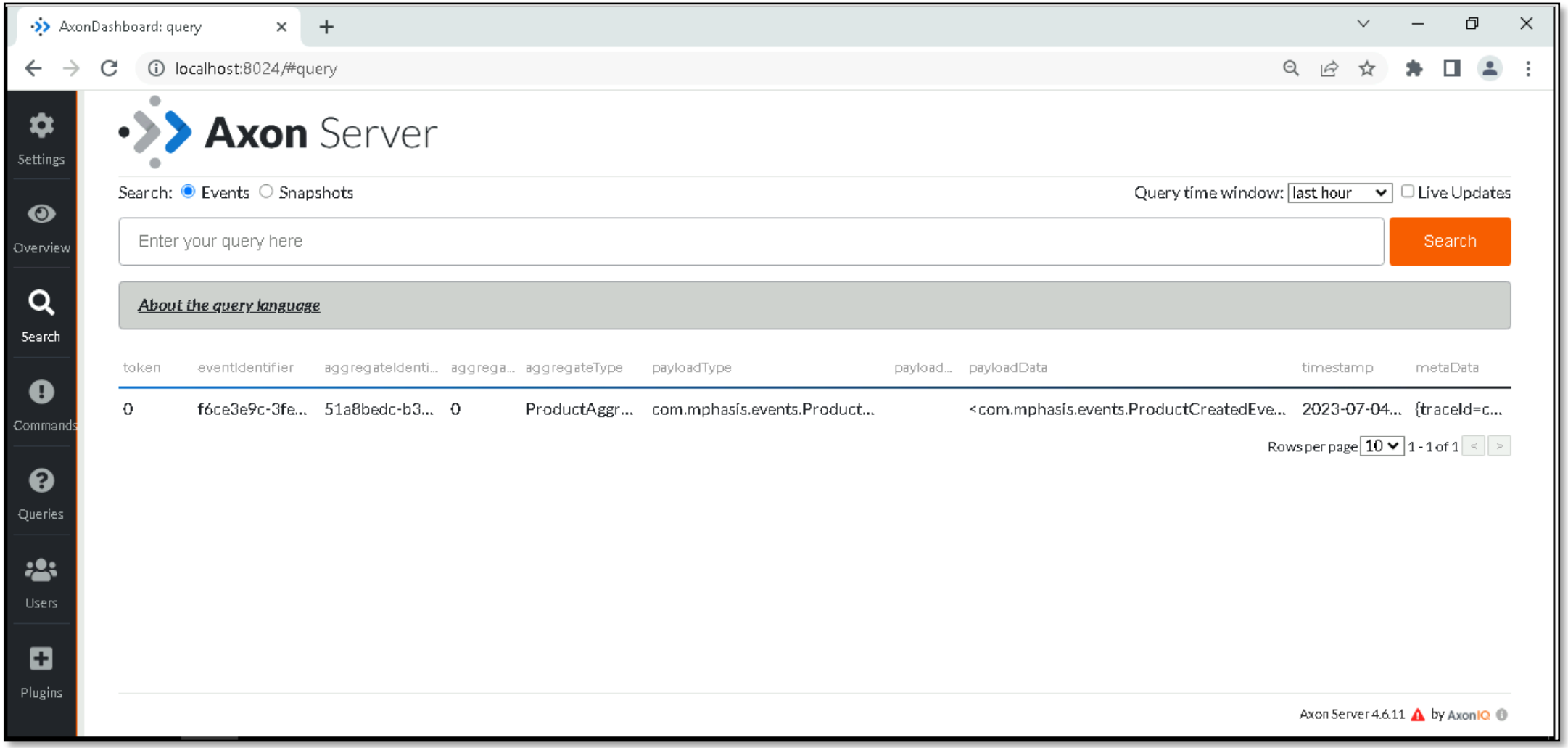
Previewing Event in the EventStore

- Select the **products-service** Application:



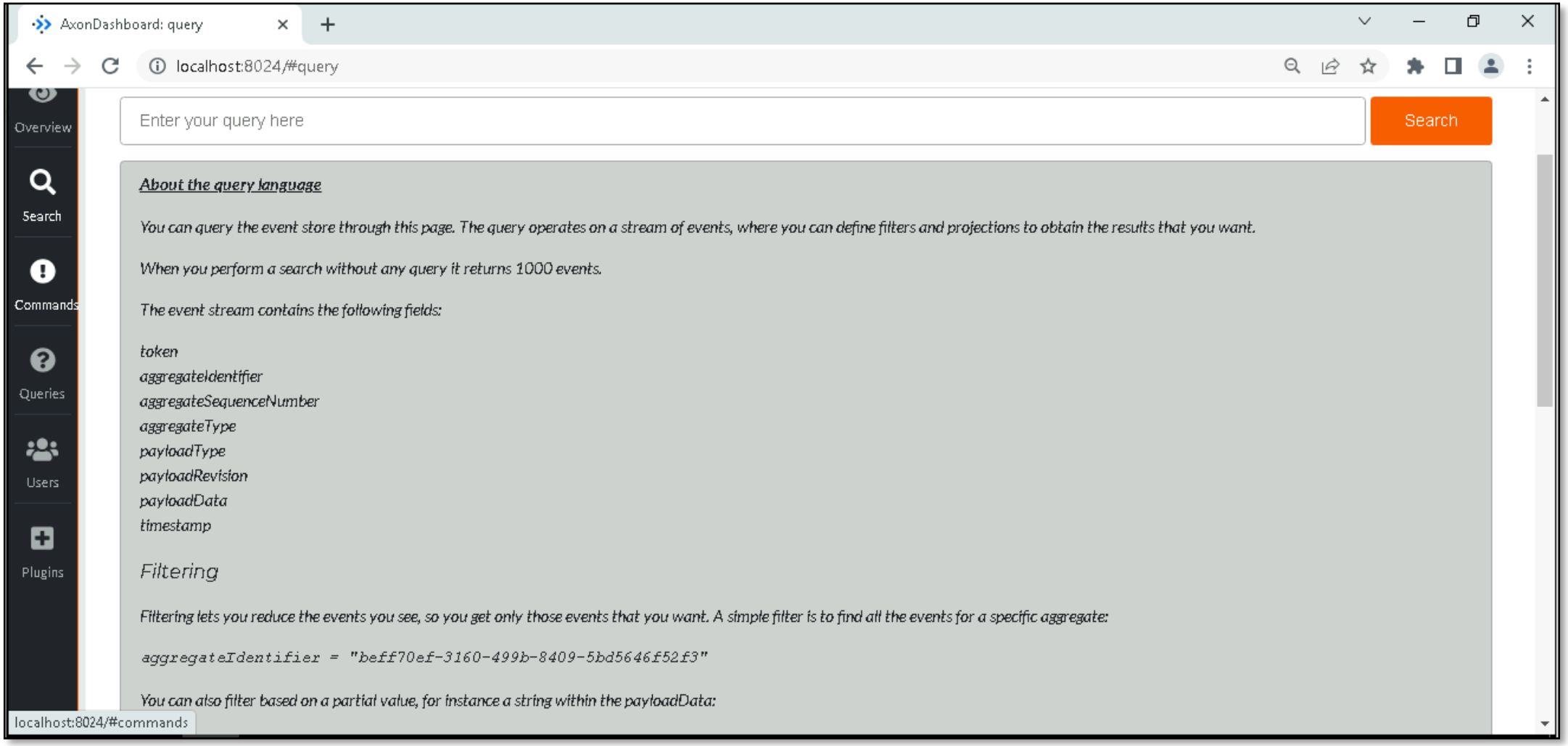
Previewing Event in the EventStore

- Go to **Search** tab and click on **Search** button:



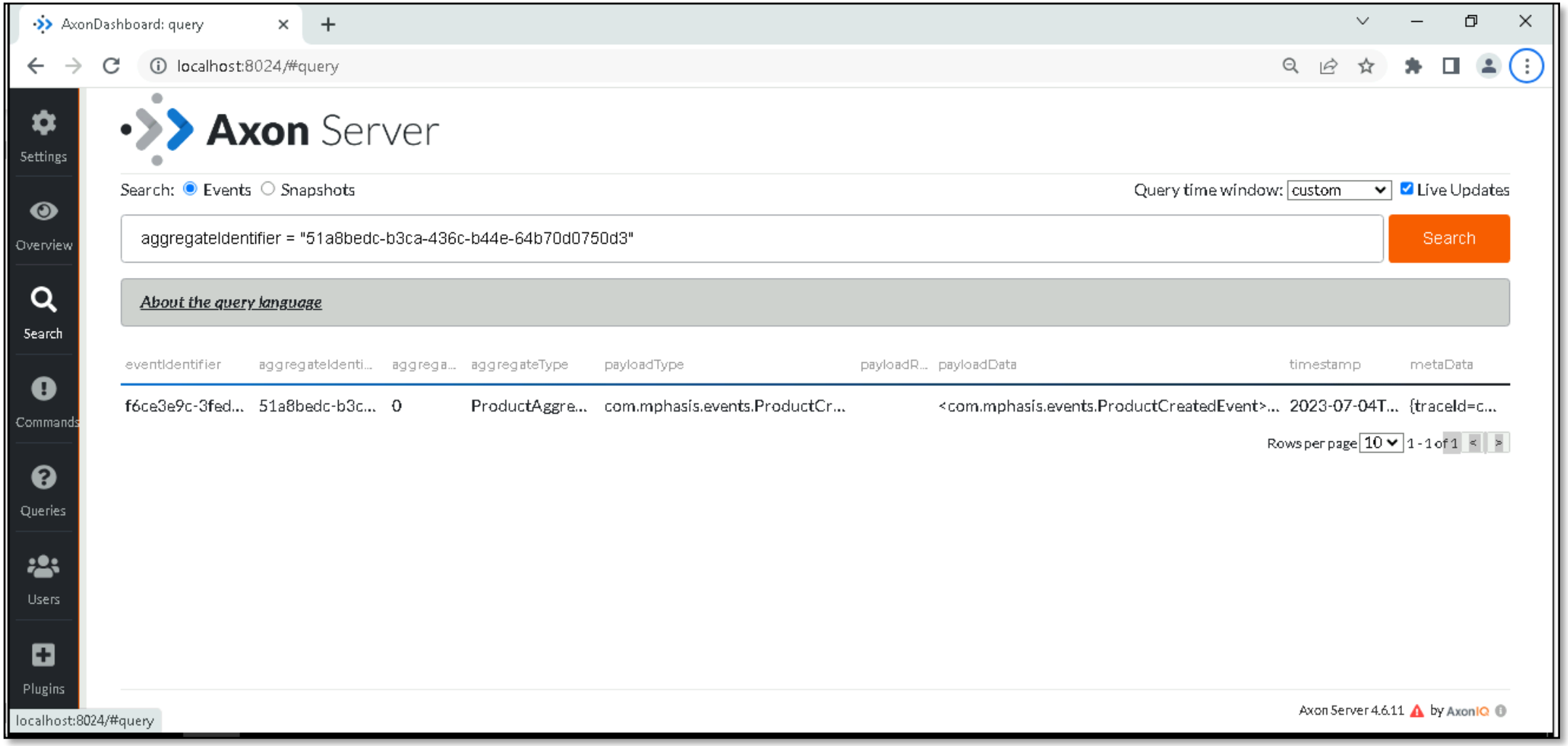
Previewing Event in the EventStore

- About the **query** language:



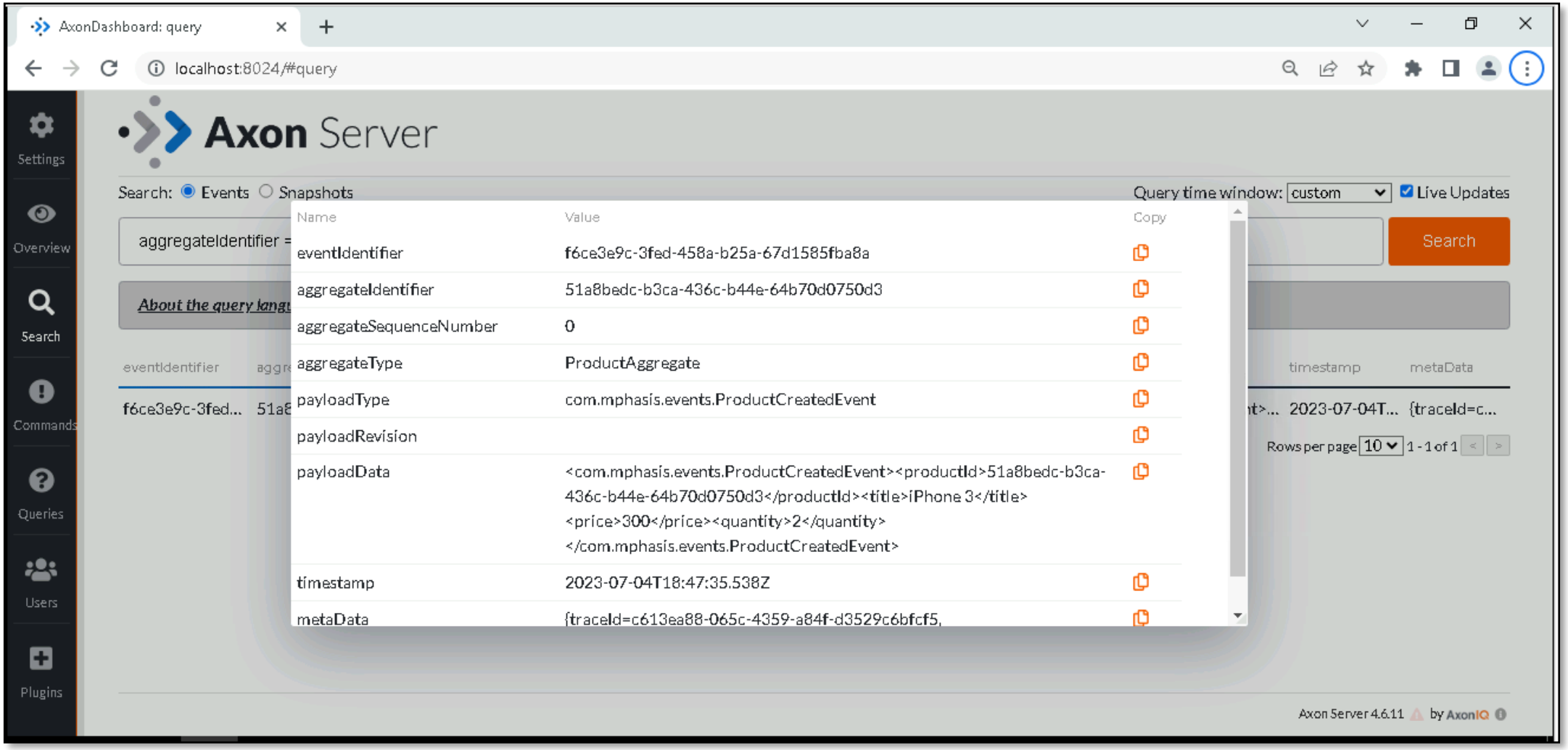
Previewing Event in the EventStore

- Search using the **aggregateIdentifier**:



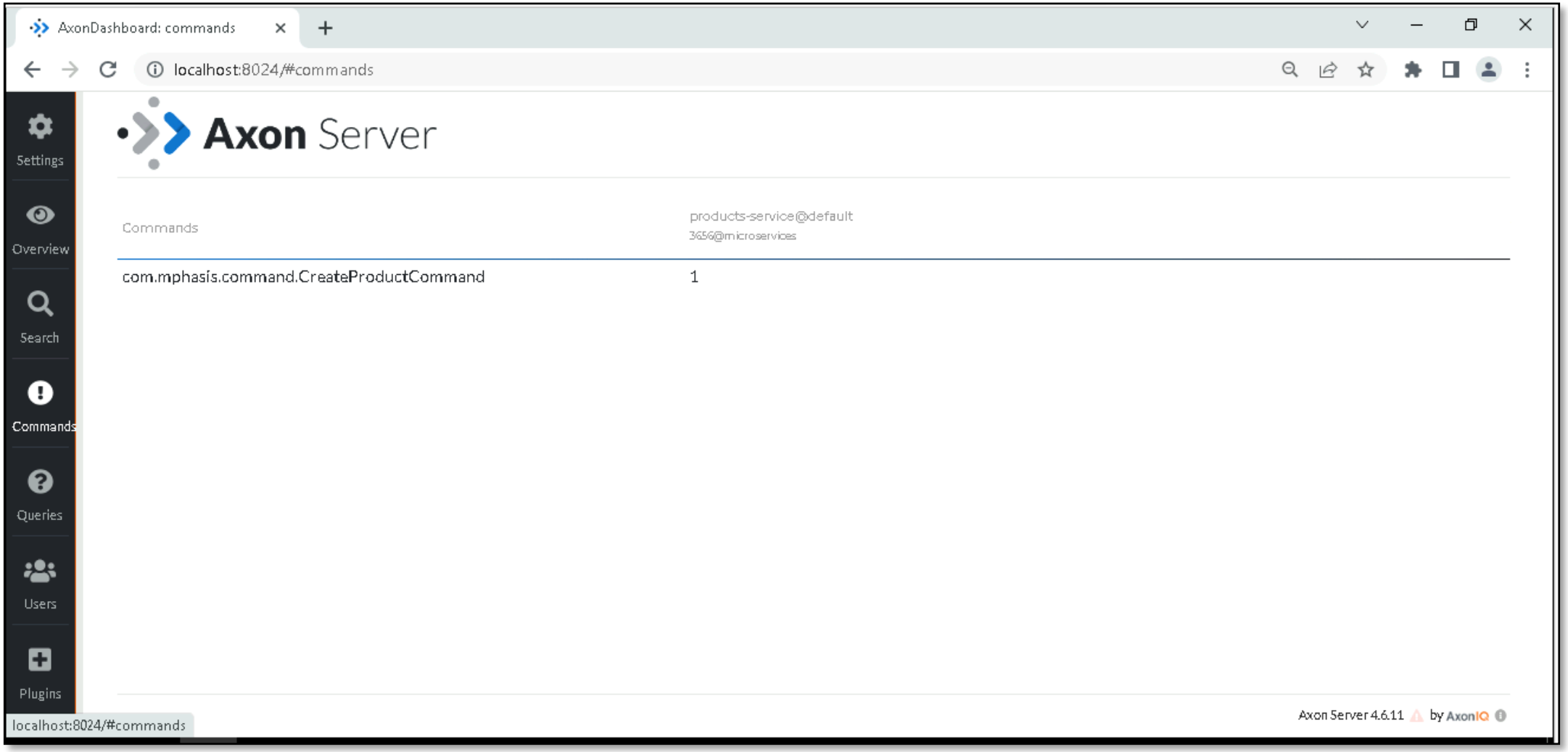
Previewing Event in the EventStore

- View the event details available in Event Store:



Previewing Event in the EventStore

- Go to Commands tab:



Recap of Day – 2

- Introduction to Axon Server
- Download and run Axon Server as JAR application
- Axon Server configuration properties
- Run Axon Server in a Docker container
- Bringing CQRS and Event Sourcing Together with Axon Framework
- Accept HTTP Request Body
- Adding Axon Framework Spring Boot Starter
- Creating a new Command class
- Send Command to a Command Gateway

Recap of Day – 2

- Introduction to Aggregate
- Creating the Aggregate class
- Validate the command class
- Creating the event class
- Apply and Publish the Created Event
- @EventSourcingHandler Annotation
- Previewing Event in the EventStore

Mphasis
The Next Applied

THANK YOU

About Mphasis

Mphasis (BSE: 526299; NSE: MPHASIS) applies next-generation technology to help enterprises transform businesses globally. Customer centricity is foundational to Mphasis and is reflected in the Mphasis' Front2Back™ Transformation approach. Front2Back™ uses the exponential power of cloud and cognitive to provide hyper-personalized ($C=X2C^2{}_{TM}=1$) digital experience to clients and their end customers. Mphasis' Service Transformation approach helps 'shrink the core' through the application of digital technologies across legacy environments within an enterprise, enabling businesses to stay ahead in a changing world. Mphasis' core reference architectures and tools, speed and innovation with domain expertise and specialization are key to building strong relationships with marquee clients. Click here to know

Important Confidentiality Notice

Any Questions?

Manpreet.Bindra@mphasis.com

FOLLOW US IN THE LINK BELOW

@Mphasis